Data Science Coding Challenge: Twitter Sentiment Analysis

Vasco Fernandes

August 26, 2018

Contents

1 Introduction

Hate speech is an unfortunately common occurrence on the Internet. Often social media sites like Facebook and Twitter face the problem of identifying

and censoring problematic posts while weighing the right to freedom of speech. The importance of detecting and moderating hate speech is evident from the strong connection between hate speech and actual hate crimes.

Early identification of users promoting hate speech could enable outreach programs that attempt to prevent an escalation from speech to action. Sites such as Twitter and Facebook have been seeking to actively combat hate speech. In spite of these reasons, NLP research on hate speech has been very limited, primarily due to the lack of a general definition of hate speech, an analysis of its demographic influences, and an investigation of the most effective features.

2 Problem Statement

The objective of this project is to find the best classification model to classify tweets.

The objective of this task is to detect hate speech in tweets. For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets.

Formally, given a training sample of tweets and labels, where label '1' denotes the tweet is racist/sexist and label '0' denotes the tweet is not racist/sexist, your objective is to predict the labels on the test dataset.

3 Data Set

3.1 Train Set

The data set provided (https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/) is composed by two different files, the train set ("*train_E6oV3lV.csv*") and test set ("*test_tweets_anuFYb8.csv*"). The first file is composed by 31621 observations, with 3 columns ("id", "label" and "tweet"), with an uneven label (or class) distribution, as can be seen in Figure 1:

3.2 Test Set

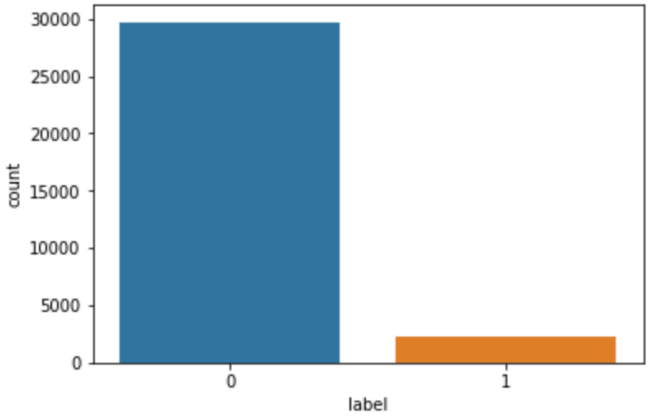
The test set is composed by 17197 observations, with only two columns ("id" and "tweet"). Given that this challenge is an open challenge, makes sense no "label" column, to make users submit their results.

4 Initial Feature Engineering

4.1 Pre-clean Tweet length

Given that in NLP tasks the most difficult challenge is the feature engineering part, and after applying algorithms out-of-box **after** cleaning and I was not able

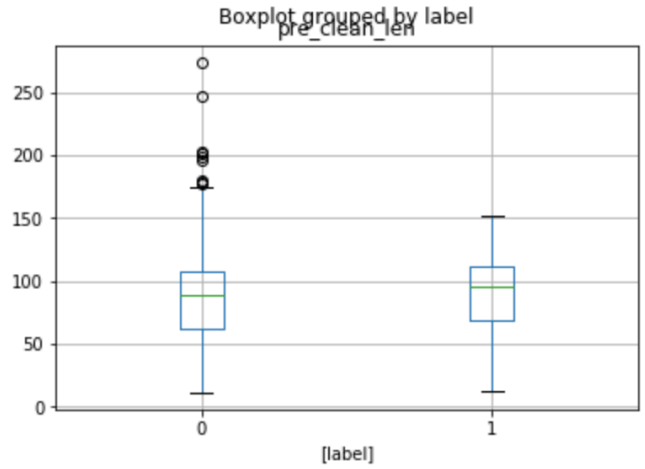
Figure 1: Class distribution



to produce good results, I tried to create my own features.

So the first hypothesis that I wanted to test was if one class of tweets has a statistically different pre-clean length that the other one. As we can see in Figure 2:

Figure 2: Pre-clean length

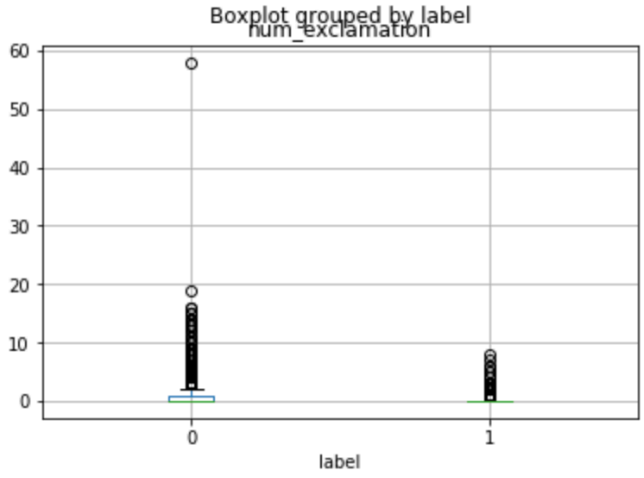


So, in terms of the median value, there is not a significant difference, but the 75^{th} percentile is somewhat deviated in class 0, comparing to class 1.

4.2 Number of exclamation marks

So following the same line of thought of the last subsection, I thought that exclamation marks could be a discriminative feature. The hypothesis was that hate-speech tweets had more exclamation marks than non-hate speech tweets. The boxplot for each class in terms of the number of number of exclamation points.

Figure 3: Number of Exclamation Marks



Again, there is not a significant difference between the two, but we can see, opposite to what I thought could be possible, the non-hate speech class has more exclamation marks.

4.3 Post-cleaning length

Skipping some steps, lets now check the post-cleaning length. Again, there is no statistically significant differences between one class and the other.

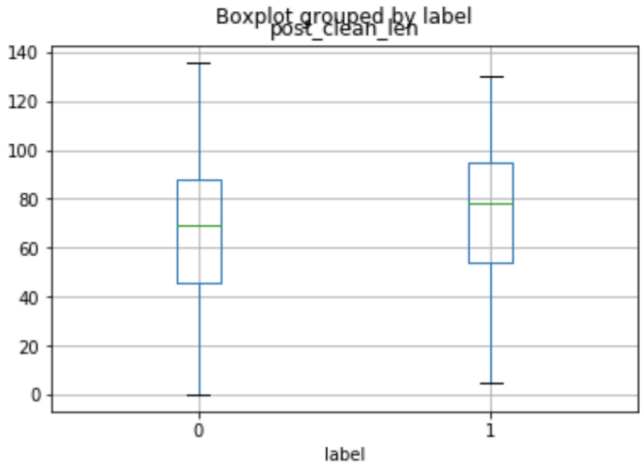
5 Data Cleaning

In this section, I will refer the cleaning steps that I took.

Given that the dataset already had user mentions anonymised, these mentions, in my opinion, did not had any predictive power. Given this, I wrote a function, called `remove_pattern(text, pattern)` that can remove any pattern of a *regex* expression. I applied this function to the data set column `tweet`, creating a new column, named `new_tweet`.

The second step of the cleaning part was to lowercase each word of the column. This operation was applied to the `new_tweet` column.

Figure 4: Post-cleaning length



The cleaning part of this project was also accomplished using another function `clean_tweet(text)`, that perform the following steps:

- Remove HTML characters.
- Encoding normalization.
- Check for common abbreviation in english and transform them to the long version (like "aren't" to "are not").
- Remove non-character symbols.
- Tokenize each tweet using the NLTK Tokenizer.

Then checked for missing values in all columns, that produced no results.

6 Exploratory Data Analysis

6.1 Word Cloud

For the positive class (non-hate speech), this is the word clouds after removing the stop words (Figure 5).

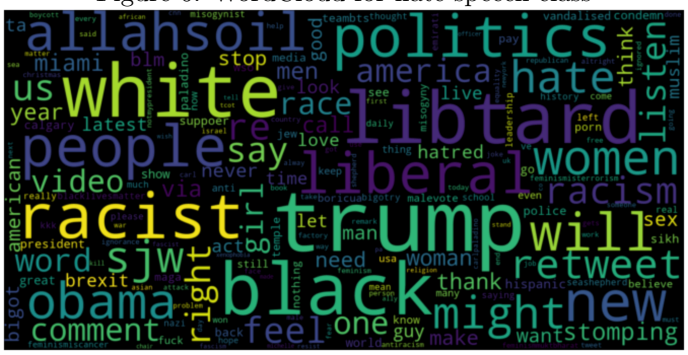
Now for the negative class (hate speech tweets), the word cloud can be seen in Figure 6.

6.2 Zipf's Law

Zipfs Law states that a small number of words are used all the time, while the vast majority are used very rarely. There is nothing surprising about this, we

Figure 5: WordCloud for non-hate speech class

Figure 6: WordCloud for hate speech class

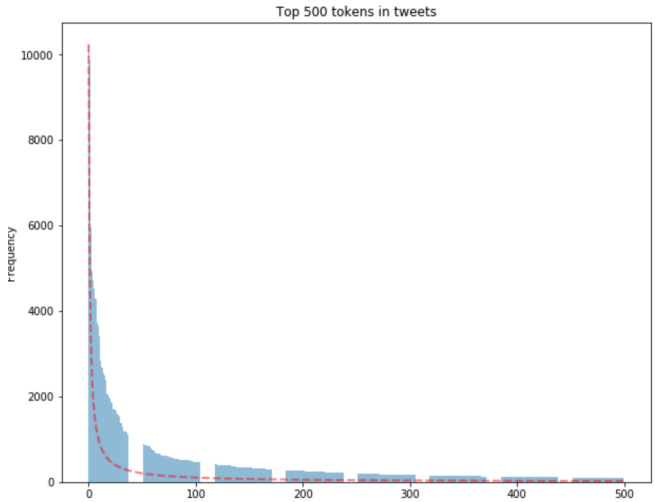


know that we use some of the words very frequently, such as the, of, etc, and we rarely use the words like aardvark (aardvark is an animal species native to Africa). However, whats interesting is that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. Thus the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.

In Figure 7, the x-axis is the rank of the frequency from highest rank from left up to 500th rank to the right. Y-axis is the frequency observed in the corpus (in this case, our dataset). One thing to note is that the actual observations in most cases does not strictly follow Zipfs distribution, but rather follow a trend of near-Zipfian distribution.

Another way to look at this data is to plot the features at a log-scale. This can be seen in Figure 8.

Figure 7: Approximate Zipf Law



6.3 Token Frequency

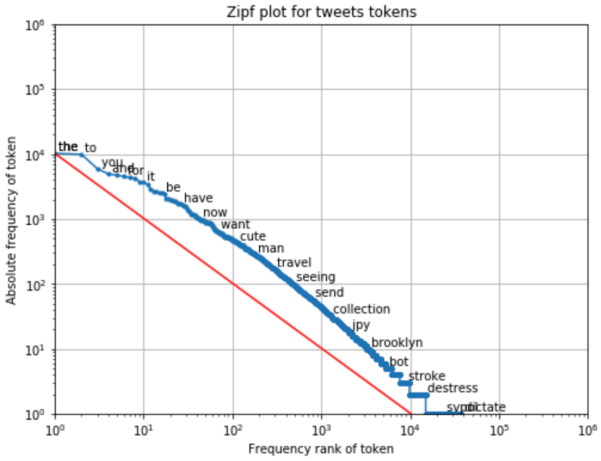
Now let's go deeper into token frequency and test several metrics in order to see if we can extract meaningful information from this set of metrics.

Token count per class After removing the stop words, the first question was "how many times a certain token appears in one class?". But lets look at the frequency values of the minority class. Everything is pretty normal, right? (In terms of the ratio between the positives and negatives). What about words like "like", "just"? (Figure 9).

Most common tokens per class So I started to look for the most common tokens after removing the stop words. In Figure 10 and Figure 11, this can be seen.

After this, I tried to plot the frequency of the positive and negative tokens in a scatter plot (Figure 12):

Figure 8: Log-scaled approximate Zipf Law



7 Algorithms & Feature Engineering

7.1 Logistic Regression

7.2 Algorithm Comparison

7.3 Deep Learning

8 Discussion and Future Work