Capstone 2: Milestone Report #1

Christos Magganas - Springboard 2019

Objective: Twitter Classification – gather twitter data from two similar groups of users with a clear identifiable keyword in their profile, specifically “Students” and “Teachers.” Then create an Natural Language Processing prediction model that attempts to identify them based on tweet text.

Data Wrangling

- Source of the Data and Method of Acquirement

Links:
https://twitter.com/,
https://followerwonk.com/,
https://github.com/Jefferson-Henrique/GetOldTweets-python/blob/master/LICENSE

The data was scraped and collected from followerwonk and Twitter with some code taken from the github link above.

The data gathered includes the 1000 usernames with either the keyword “student” or “teacher” in their profile (500 each). The other search parameter used was to have a follower range between 500-600, … , 1400-1500 so that a more diverse sample set with enough credibility could be collected.

Scraping multiple tweets from multiple usernames over a short period of time from Twitter’s page triggers a warning such that Twitter will block your actions temporarily. Similarly, followerwonk limits the number of pages per search and the number of searches that can be made at once for free without a subscription. To surpass this roadblock, for the collection of usernames from followerwonk I used the time module in python to make the scraping function sleep for a designated amount of time before searching again. This method was intrinsically time consuming, but avoided blank pages being scraped via less time expensive methods.

I used some code from the github link above because scraping twitter data without an API was too difficult and gathering data with the API may have lead to more limitations of what I could search and collect. Once the list of usernames was gathered, I scraped the first 5 tweets of each user from the list of usernames. I stored each tweet

as a datapoint into a dataframe and then into a csv file for each class (student/teacher) to be used for further analysis.

- Merging and Sorting

The student and the teacher DataFrame would have to be merged together such that no data was lost or changed. But some of the usernames appeared in both DataFrames before the merge. I decided that these users would be too ambiguous to assign to one class, so I removed them from both. I assigned a classification and code column to each DataFrame so that they would be identifiable after the merge. At this point, the data was ready for cleaning and pre-processing.

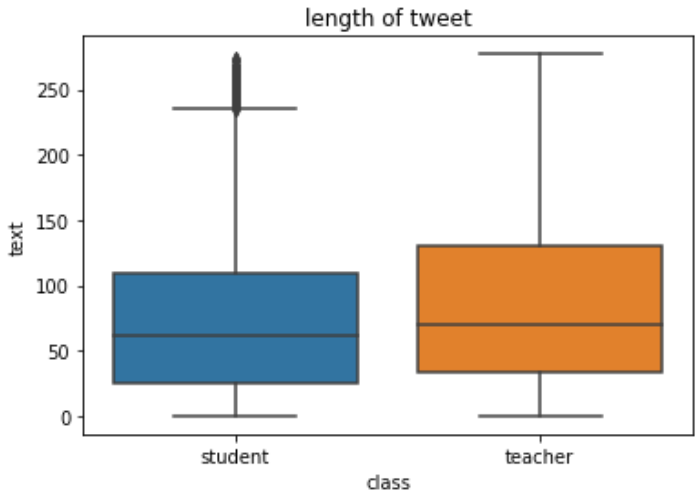
- Cleaning Steps and Pre-Processing

The first cleaning step I made was to ensure that all of the tweets were in english. I used a language detector to remove non-english tweets. The next cleaning step was to remove any url whether it was a link or a picture. Then I removed all punctuation except for the '#' and '@' symbols which give different meaning to the words that they connect to. Then I made each word lowercase for easier comparison.

The two most common pre-processing methods that are used in NLP are called stemming and lemmatization. Stemming takes and replaces a word with its root, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. In a few test runs, stemming seemed to change too many words for the worse, while lemmatization did not. For this reason I chose to only use lemmatization during pre-processing.

Exploratory Data Analysis

- Finding trends / Visual Analysis

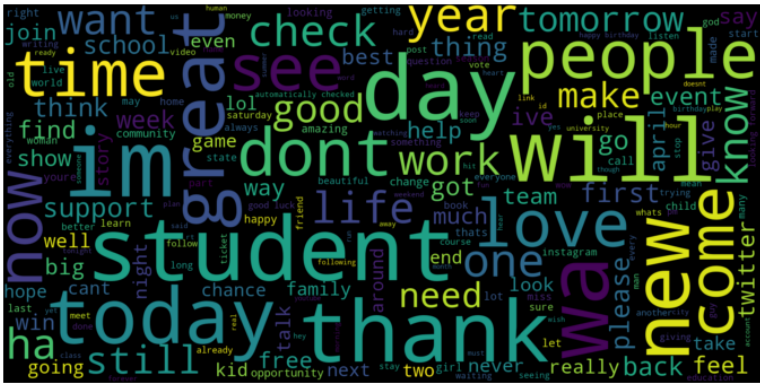


(Left: box plot comparing length of tweets for students and teachers)

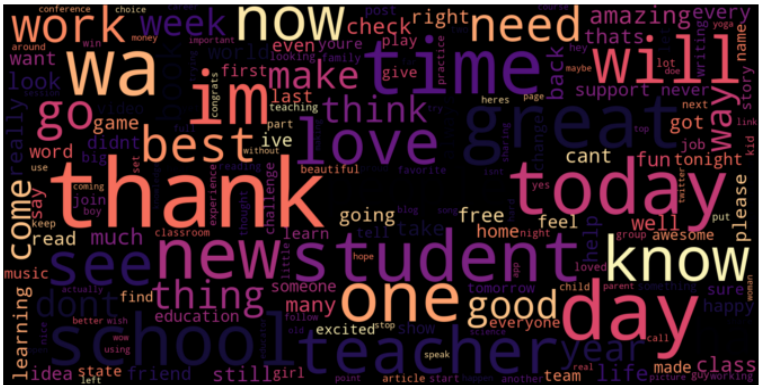
If we look at the graph on the left, we see that the lengths of student's tweets are a bit shorter than the lengths of teacher's tweets. But the differance does not seem to be significant enough to be useful.

To briefly visualize which words appear most often in each class, we can use a WordCloud. A WordCloud represents word usage in a document by resizing individual words proportionally to its frequency, and then presenting them in random arrangement.

Here we see the most commonly used words tweeted by students (not including words from the built-in stop word list such as "the" or "and" etc.)



Here we see the most commonly used words tweeted by teachers (not including words from the built-in stop word list)



Inferential Statistics

- Initial Analysis / Term Frequency

At this point in our Natural Language Processing Analysis, we want to turn the text into quantitative data that can be measured and modelled. First, the entire set of tweets is tokenized, as mentioned in the pre-processing section. Each token, which in this case is either a word in lowercase form, a number, or some combination (meant to include times like “11pm” for example) is put into the dictionary of term that will be henceforth referenced.

	students	teachers	total
the	710	843	1715
to	651	478	1494
a	474	52	1141
and	427	419	998
of	334	3	812

As we can see from the *term-frequency table* on the left, all the terms are what are known as stop words. Stop words are the most common words in a language usually filtered out during pre-processing due to how little they contribute to the meaning and how often they occur, diluting the important information. However, for our purposes, since each tweet possesses so few words, it will be more advantageous if we remove the stop words methodically based on their non-predictiveness to maximize our prediction score.

- Train Test Split

Before we can train any model, we first consider how to split the data. Here I chose to split the data into a training and testing chunk. The training set data is used for learning, tuning parameters of the classifier, and 5-fold cross validation so that an unbiased evaluation of a model can be made. The testing data is used only to assess the performance of the best models.

- Vectorizers (CountVectorizer & TfidfVectorizer)

When using a CountVectorizer, it merely counts the number of words in the document. More specifically, it converts a collection of text documents to a matrix of the counts of occurrences of each word in the document. TF-IDF (stands for Term-Frequency-Inverse-Document Frequency) weighs down the common words occurring in almost all the documents and give more importance to the words that appear in a subset of documents. TF-IDF works by penalising these common words by assigning them lower weights while giving importance to some rare words in a particular document. The TfidfVectorizer also converts the collection of text documents to a matrix, but uses weighted vectored instead. The TfidfVectorizer is more often than not a better choice, but considering the smaller size of the dataset and the few number of words in each tweet, common words that may be more helpful in distinguishing between classes would be down weighted in TF-IDF, but given equal weight to rare words in countvectorizer. This will be something to consider going forward.

- Classifiers (LogisticRegression & RandomForestClassifier & MultinomialNB)

Logistic Regression (LR) is an easy, fast and simple classification method. Naive bayes (NB) is a prime model for text classification, especially when featureset is very large. NB works well with small datasets and is a generative model whereas LR is a discriminative model. LR performs better than NB with collinearity, because NB expects all features to be independent. Random Forest (RF) is an ensemble model that is robust, accurate and powerful. RF handles overfitting efficiently and derives feature selection and importance. RF is however computationally complex and slower than LR and NB.

I have chosen these three classifiers for the majority of my analysis for the reasons stated above as they are ideal for my smaller NLP Classification dataset.