Homework 2 and 3

Kyle Walter

10/29/2021

# Introduction

In continuing to pursue techniques for analyzing public sentiment of AI, this paper will explore some of the techniques used in prepossessing text. Text comes in many forms and task at hand requires various forms of techniques to work with the data the has been collected and the task at hand can drastically change the technique utilized.

A commonly sighted example when looking at techniques is the discovery of authors. Writers tend to use function words differently from one another. Many of these words are often recognized as stop words and in sentiment analysis would be removed, however; when looking at who may have written an assignment this technique may actually remove the ability of an algorithm to decern differences between the writers and thus make it less capable of identifying the author.

These steps are important however because text comes with many forms of words that are often similar in meaning and can create problems of too many tokens with out anything providing meaningful results.

# Analysis and Models

## About the Data

The data for this analysis contained 4 files collected from Kaggle. 3 for this analysis which were text files that contained a review on Amazon, Yelp, and imdb. Each review was followed by sentiment of 1 or 0 for negative or positive review.

In addition the data set used from the previous analysis was intitally added, that data set came from twitter contained sentiment for 1.6 million tweets. The data set due to its size was later dropped as memory issues caused many issues. It was brought in to show that data can be read from multiple sources per the assignment specifications.

### Reading in the data

The text data was read in via a for loop, split using the tabe in the data set seperating the text from the sentiment, and added to a dictionary with 2 values. Sent for Sentiment Marker and text for the text of the informaiton. Lastly the data set was transformed into a pandas data frame.

The csv file was read in using the code from assignment one, though the headers were modifed to use the same names as the columns in the text file data frame. Pandas can natively parse the csv directly into a data frame so no additional work was required at this point.

Since the columns had the same name, the additional columns from the twitter data were removed and the two data frame were intiall concatednated so that they could be run together.

As mentioned previous, when vectorizing the data length of the twitter data presented problems and the steps were commented out in the code, if the comment are removed the entire set can run through then entire code.

Sampling the data set for records was considered, however given the large size of the twitter set at 1.6 million records vs the 3000 records and the time alotted for the assignment finding a balance was nixed as a full sentiment was done on the twitter data. Data from Twitter will be talked about through the preprocessing as it presents some intersting additional steps.

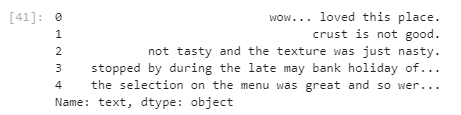
## The Analysis

Now that the data was read in and transformed into a data frame it was checked to see if the sentiment tags were balanced. with 3000 records each having 1500 postive and negative sentiment the data set was alread presented into a balanced manner. This is important for model building as data sets with imbalance lead models to favor the class with more frequency when information is limited.

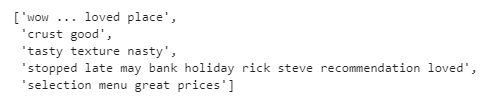
### Preprocessing

The first step in the preprocessing was to lowercase the text. This is important as later when creating vectors of Tokens those words that happen at the beginning of the sentence and would get captilized will now converge with those in the the middle of the sentence. Next, a column called sentence was created and the text was Tokenized by sentence. This was done to check how long reviews in the data set were. For just the text in the reviews it was anverage of 1.02 sentences per review. Which is quite short. Maybe because these are older reviews. When adding the twitter data it rose to 1.65 sentences but does help to know tokenization can happen without burying the sentiment of the review. Had these been longer use of bigrams maybe needed to keep negative sentiment intact as english often uses helper words to drive sentiment.

Last step at this first part was the removal of stop words from the main text. These are the function words mentioned earlier. Words like the, he, and of which have little sentiment to impart, but removing them will help reduce the size of the vectors created.



View of Sentence with Stop Words



View of first five sentences after stop word removal

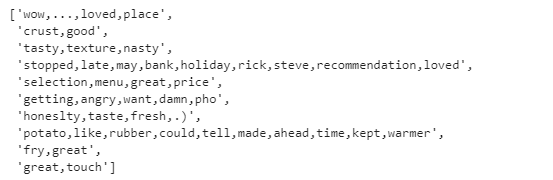
Stop words can present problems. As seen in the second sentence above the sentence went from crust is not good to crust good because both is and not are recognized as stop words. This example will likely be miss classified in the model as the negative sentiment has been removed.

In fact many of these type words can be seen the in the NLTK natural stop list towards the end.

Negative Terms on the standard stop list

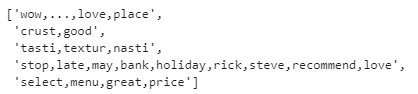
Negative Terms on the standard stop list

Next step was to look at what happened when applying a lemmentizer. As English has numerous words that gain various endings to show certain changes in things like number where plural words get an ‘s’ added to the end. Or changes in verb endings to denote tense as examples. Lemmentatization is the process the of returning these modified words to their original structure. Applying this to the list of sentences where the stop words have been removed, almost no detectable changes can be identified. This is one change when expanding the view from tastes to taste however; the lemmentizer skipping some words like loved instead of changing to love was a bit surprising.



Lemmentizer output

While the lemmentizer did not appear provide much change, stemming was also considered. Stemming similar to Lemmentization removes added endings, but often times leaves nonsensical words its wake by just chopping a portion of the ending off and leaving the butchered word behind. This is observed when applying it to the Tokenized Text.



Stemmed Text

Notice the the text document went from Tasty, texture, nasty to tasti, textur, nasti. All the words can be read yet with the porter stemmer but other may remove them from reconcilability.

The main reason for doing lemmatization or stemming is to reduce the number of text tokens that remain. Searching on stack overflow a technique was found to look at each of the list Tokenized Text, Lemma Text, and Stemmed Text that has been generated thus far and return the unique number of tokens.

Number of Tokens for each technique

Number of Tokens for each technique

The drastic number of tokens reduced for both Lemma and Stemmed results did call into question the reliability of the method. However; given time constraints, for vectorization and model building, it was decided that moving onto the task at hand and testing vectorization would be more fruitful.

# Vectorization and Model Buidling

Moving on the vectorization models for building the structured data set from the text documents current collected. Sklearn was used to build the vectors using straight word count for one model and term frequency inverse document frequency for the other.

Sklearn’s vectorization will actually accept text and tokenize it based on white space and remove numbers and punctuation by default. It also has a built in stop word list that can be used or a separate list can be supplied.

In the prior assignment these measure were just utilized and the vector produced was run through the model. For this assignment, since these factors can be defined for the model the following adjustments were made.

First lowercase words were set to true. In theory since this was done as the first step in the corpus data frame it would not need to be run again. Second the analyzer method was set to ‘word’ meaning the tokens generated are the basis for the intersection. Instead of letters or alpha numerics which while they have a place in other models are likely not going to be helpful for sentiment. Gender of a name is more likely useful place for this kind of analysis.

The next option added was n-grams. The vectorizers allows the selection of how many units should be in a token. An entry of (1,1) would return only one word tokens where (1,3) would return unigrams, bigrams, and trigrams. Nltk has options with the collocations section of the package to tokenize n-grams as well and with methods that allow for the removal of stop words in the process but stop words cannot be removed previous or it will create incorrect grams. In a sentence like “A man walked into the bar” the removal of stop words through pre-processing would bring “into bar” together is a single bi-gram even though there was no connection between these two words till stop words were removed. Both the Vectorizers similar to nltk’s collocations has the ability to recognize where the stop word was and not create these erroneous n-grams.

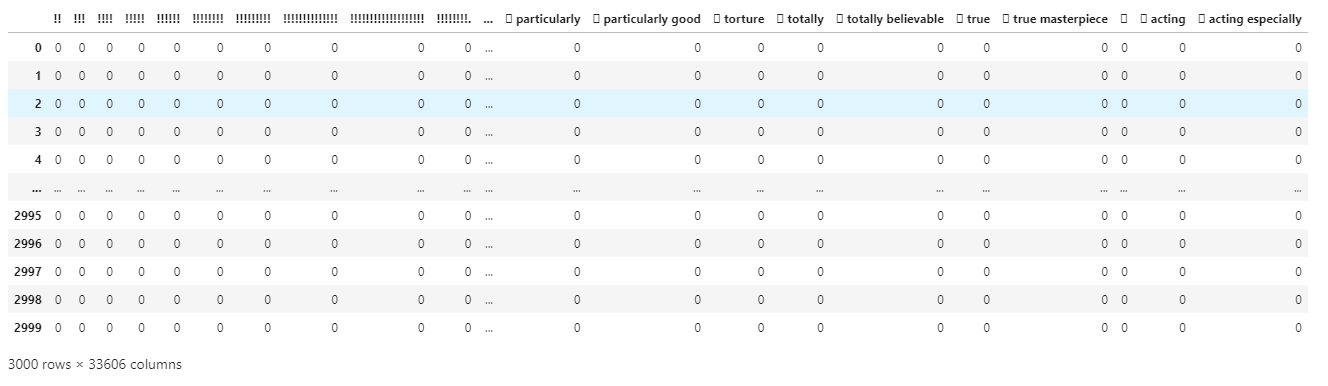
The decision was made to add in the bi-grams and trigrams to deal with the possibility of negation words noted earlier in the model.

Lastly into the vectorizer parameters stop\_word list defined at the start of the code, using mostly nltk’s standard stop list and tokenizer used the pre-processing step was defined to be used for tokenization. The decsion to use the word\_punct tokenizer was done so that way punctuation would be sperated out and could be added to the stopword list so these tokens would not be used. Since punctuation mostly does not indicate sentiment and creates a larger number of n-grams thus driving up the computation time for the models.

## Count Vectorizer and MultiNomial Model

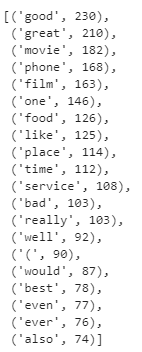
As the name suggest the count vectorizer goes through an creates a count of each word in each text of the corpora and then returns a sparse matrix of words and their counts where each row represents a corpus and each column represents a word.

Thankfully, Sklearn’s count vectorizer controls sparse matrix with dictionaries under the hood though the sparse matrix can be seen. As noted in the image below the Tokens are visible across the top but most have a count of zero.



Sprase matrix

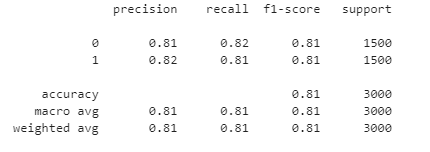
As an end goal is to apply a MutliNomial Naive Bays model to the data for predicting the outcome. After the count vectorizer often times the features with the highest counts are the ones most impact the model.



Top 20 features

With the features created, the final step is build the model and test it’s ability to predict sentiment. The tags from the data set were put into a variable called y, and 10 fold cross validation assigned. This enables the ability to use the entire training set and build the optimal model using the inputs.

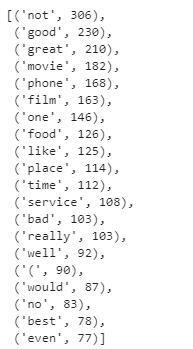
On the run with the initial controls from pre-processing including the stop list from nltk produced pretty nice results.



Initial NB Model with Count Vectorizer

Overall the model produced an 81% accuracy with similar recall and precision scores.

However; as noted earlier some of the text corpus were able to seen with dropped negation words such as not that would change the sentiment an normal reader would assess. To deal with this, the stop word list was modified to remove the negation words contained with in the the word count vectorizer rerun. The top 20 features show not at the top of the list thus proving the change.



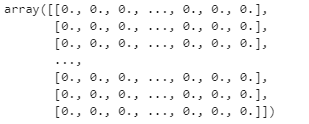
Top 20 Features from modified stop word list

The rest was held constant from before and the multinomial model was rerun this time an output of 83% and recall and precision scores slightly higher providing a model that is an improvement over the original, albeit slightly.

## Term Frequency Inverse Document Frequency Vectorizer and Model

This vectorizer takes in all the words and looks at them within their document and then penalizes them by how frequently they show up in other documents. This vectorizer can help negate stop words without the need to remove them, but for the sake of testing the model, the same parameters utilized in the Count Verctorizer. Namely stop word removal, word punct tokenization from nltk, n-grams between 1 to 3 words in length.

Similar to the word count matrix, tfidf will create a sparse matrix, except that instead of the values representing a count of the word it will represent a float between 0 and 1.

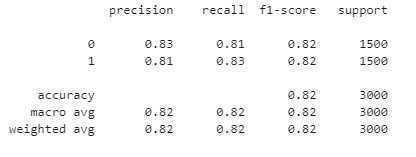


TFIDF Array

As can be see in the displace following the 0 is a “.” when the word represented at the intersection is present a decimal value will be available.

The next step step is to apply the naive bays multinomial model to the predict sentiment. This time the tfidf values are supplied as the independent variable and the same y from the count vectorizer model represents the human tagged sentiment.

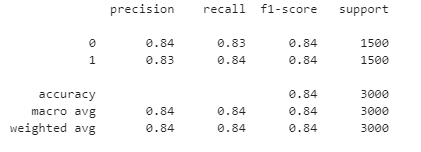
Keeping the same 10 fold cross validation in place to train for the optimal model across the data. The prediction matrix is returned as below and shows slightly better than the first run of the count vectorizer version of the model at 82% though the recall and precision are slightly slower the model itself is quite strong



Inital TFIDF Model

Following the testing pattern similar to what was completed in with Count Vectorizer, the next step was to apply the updated stop word list, with the negation terms such as not removed to allow them to remain as tokens.

The model was rebuilt, holding all other parameters the same and this time the model produced a nice 84% accuracy. with similar recall the precision to the model. This is a nice model to see as a normal target for any model is 85% accuracy with the strong recall the precision.



Update TFIDF Model

This would be the best model and vectorizer component recommended to the client.

# Conclusion

The final tfidf model moves the performance for the sentiment analysis from the initial review of free tools for sentiment analysis ranging from 64% to 77% accuracy to 2 models with variation ranging in the low to mid 80%. The additional techniques have helped move the model to place where the recommendation to client for utilizing these tools and techniques will produce likely produce a quality result.

The models are built on reviews and other sources readily utilized by the public so length of corpus and general complexity of words will be similar when searching for data around AI.