Capstone 2 - Goodreads Reviews Sentiment Analysis Report

Introduction:

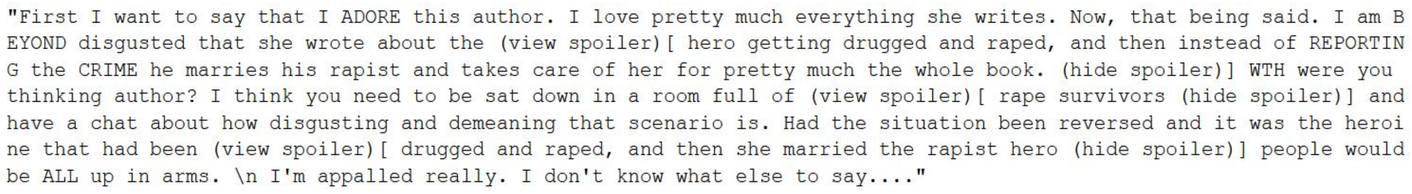
Goodreads is the largest site for readers and book recommendations. The primary feature of the site is to recommend users books based on other books they like. However, it also functions as a social media platform where users can see what books their friends are reading and recommending.

Goodreads has a large corpora of scored (one through five stars) reviews which can be used for sentiment analysis. This corpora has the potential to be applied to other review texts to predict the sentiment or make other predictions about the text.

Data Wrangling:

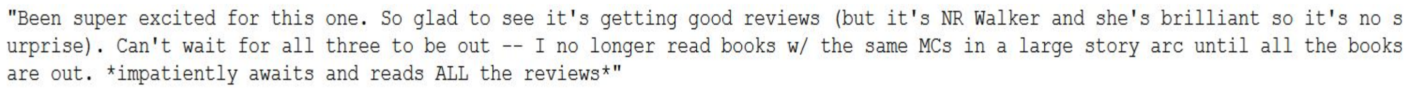
The dataset was taken from this University of California - San Diego research website. I downloaded the subset of reviews that contained a spoiler tag due to the overall size of the review corpus (over 15 million reviews) being too large for the scope of this project. The data was in JSON format with several features (review text, rating, user ID, book ID, and review ID).

The review text did come raw with some newline tags, “/n,” and spoiler markings, “(view spoiler).” I was able to create a binary spoiler variable upon import checking for the spoiler tag, as I am also interested how well I could predict a review containing a spoiler (without the tags) in the future. My first step in text preprocessing was to remove the newline tags and the spoiler tags. See a sample of a raw text review below:



```
"First I want to say that I ADORE this author. I love pretty much everything she writes. Now, that being said. I am B
EYOND disgusted that she wrote about the (view spoiler)[ hero getting drugged and raped, and then instead of REPORTIN
G the CRIME he marries his rapist and takes care of her for pretty much the whole book. (hide spoiler)] WTH were you
thinking author? I think you need to be sat down in a room full of (view spoiler)[ rape survivors (hide spoiler)] and
have a chat about how disgusting and demeaning that scenario is. Had the situation been reversed and it was the heroi
ne that had been (view spoiler)[ drugged and raped, and then she married the rapist hero (hide spoiler)] people would
be ALL up in arms. \n I'm appalled really. I don't know what else to say...."
```

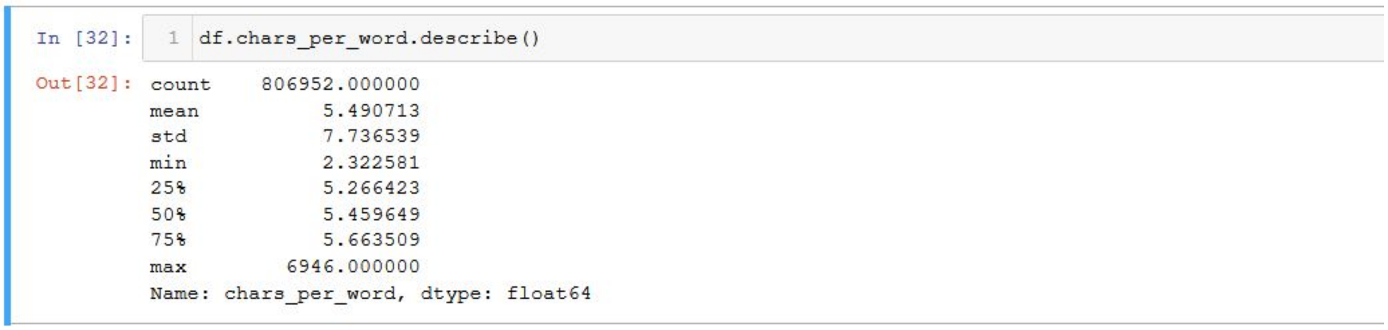
I noticed that there were a number of zero ratings in which a star rating was not given to the review. After sampling a few of the reviews, I noticed many were negative while others indicated that the reader had not finished the book, and thus was not a final review. It appeared that these were non-scored reviews and should be thrown out. Starred ratings were only available as integers from one to five. See a sample of a zero rating below:



```
"Been super excited for this one. So glad to see it's getting good reviews (but it's NR Walker and she's brilliant so it's no s
urprise). Can't wait for all three to be out -- I no longer read books w/ the same MCs in a large story arc until all the books
are out. *impatiently awaits and reads ALL the reviews*"
```

I also removed reviews under 100 characters as the text is unlikely to contain as many informative words as a longer review. This was also part of an effort to reduce the size of my corpus due to memory and time constraints.

I created a characters per word feature from the text, and noticed a few extreme outliers with the greatest value being over 6000. These few outliers were often a single broken link to an image, and I decided to remove them from the dataset.



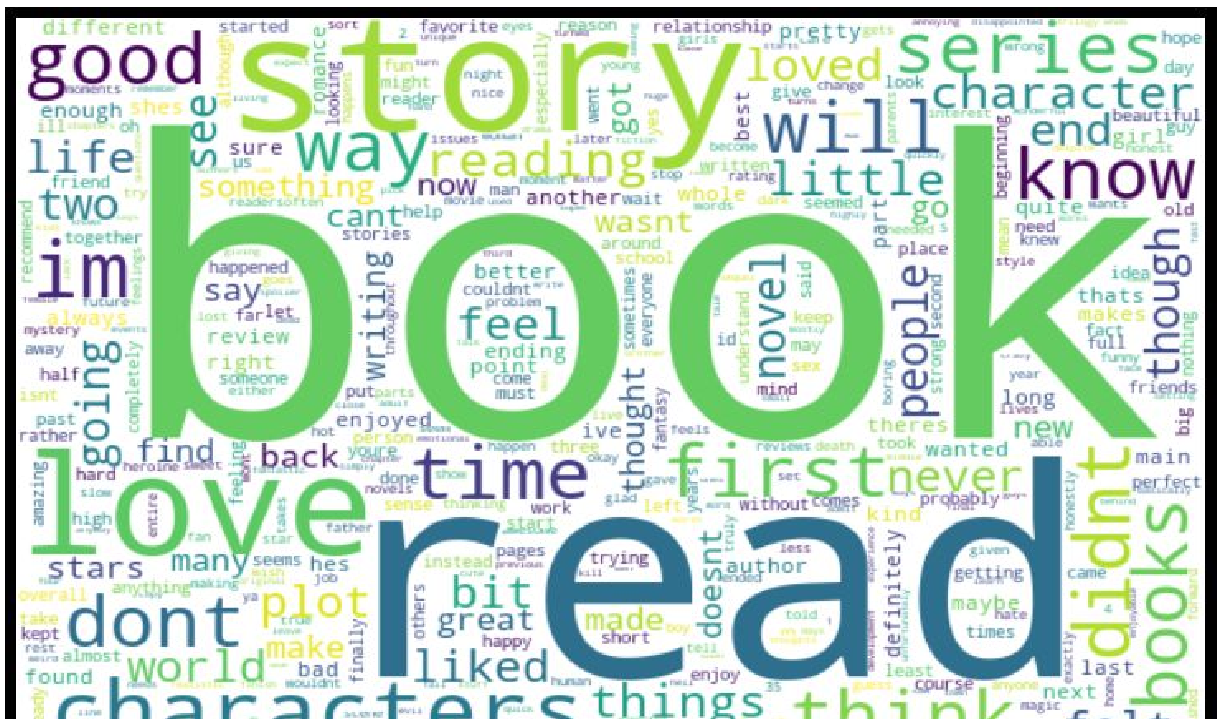
```
In [32]:  df.chars_per_word.describe()

Out[32]: count    806952.000000
         mean          5.490713
         std           7.736539
         min           2.322581
         25%           5.266423
         50%           5.459649
         75%           5.663509
         max        6946.000000
         Name: chars_per_word, dtype: float64
```

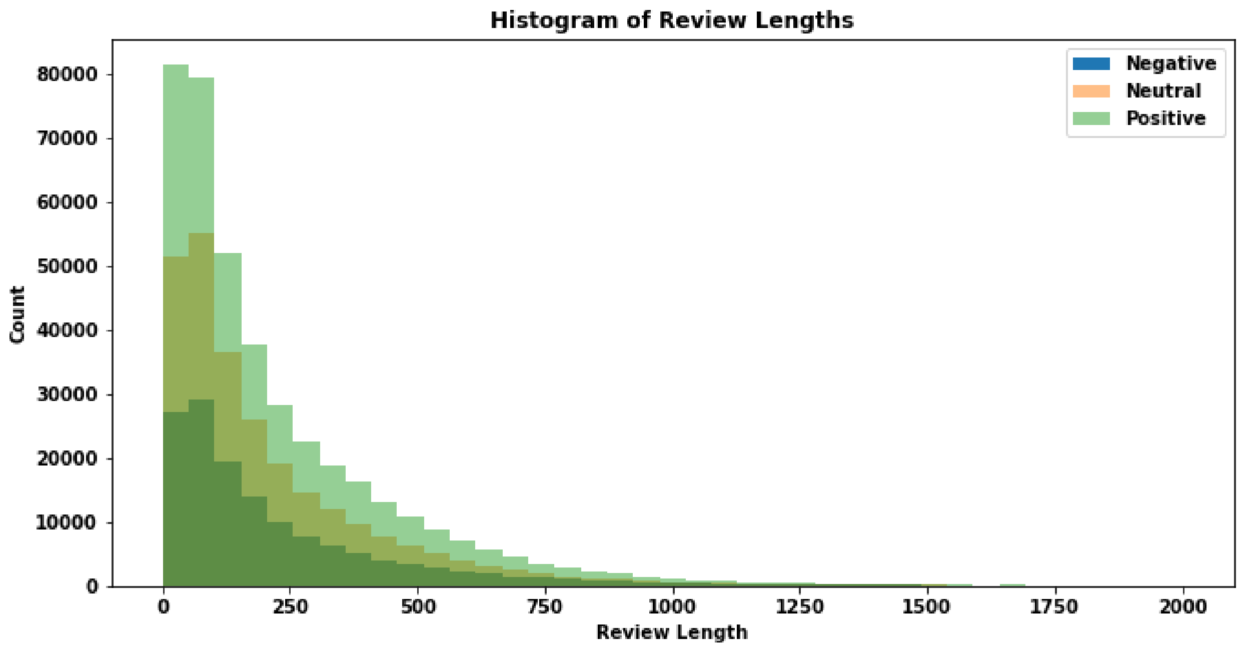
I then mapped the ratings to from 1 to 2 as -1 or negative, 3 as 0 or neutral, and 4 to 5 as 1 or positive. The vast majority of reviews were positive, so I took a sample of about 400,000 positive reviews to retain in my corpus to reduce the overall size of the corpus and to bring down the number of positive reviews to be equal to the total number of negative and neutral reviews.

Data Exploration and Story:

I made a word cloud of a sample of book reviews. As most of the reviews were positive and books are associated with pleasant feelings, most of the words are positive.



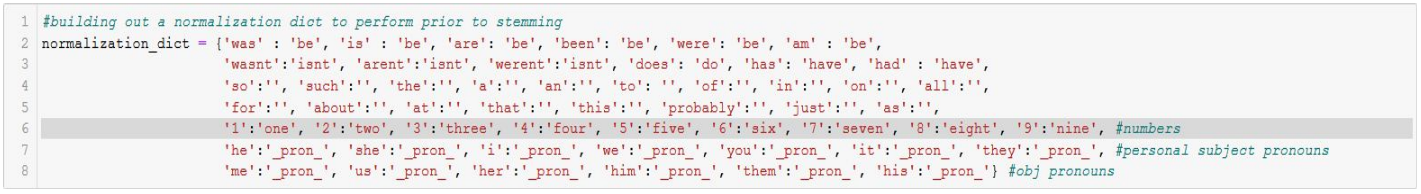
The review lengths for all types of reviews had a similar distribution with most being under 250 words.



Preprocessing and Text Features:

Preprocessing:

For the CountVectorizer and TFIDFVectorizer, I preprocessed the text by removing everything that was not a letter or number and converting to lowercase. I then created a normalization dictionary that converted common variants of common verbs to a single verb and converted common pronouns to a single "_pron_" that denoted a pronoun. Finally, I applied a Porter Stemmer to the text. All of this was in an effort to reduce the number of features and also have words with similar meaning or function to be captured as the same word. See the normalization dictionary below:



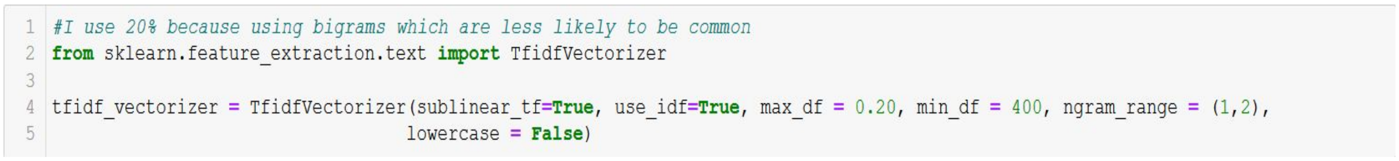
```
#building out a normalization dict to perform prior to stemming
normalization_dict = {'was' : 'be', 'is' : 'be', 'are': 'be', 'been': 'be', 'were': 'be', 'am' : 'be',
                      'wasnt':'isnt', 'arent':'isnt', 'werent':'isnt', 'does': 'do', 'has': 'have', 'had' : 'have',
                      'so':'', 'such':'', 'the':'', 'a':'', 'an':'', 'to': '', 'of':'', 'in':'', 'on':'', 'all':'',
                      'for':'', 'about':'', 'at':'', 'that':'', 'this':'', 'probably':'', 'just':'', 'as':'',
                      '1':'one', '2':'two', '3':'three', '4':'four', '5':'five', '6':'six', '7':'seven', '8':'eight', '9':'nine', #numbers
                      'he':'_pron_', 'she':'_pron_', 'i':'_pron_', 'we':'_pron_', 'you':'_pron_', 'it':'_pron_', 'they':'_pron_', #personal subject pronouns
                      'me':'_pron_', 'us':'_pron_', 'her':'_pron_', 'him':'_pron_', 'them':'_pron_', 'his':'_pron_'} #obj pronouns
```

I then used test-train-split to randomly split out the data prior to feature creation.

CountVectorizer:

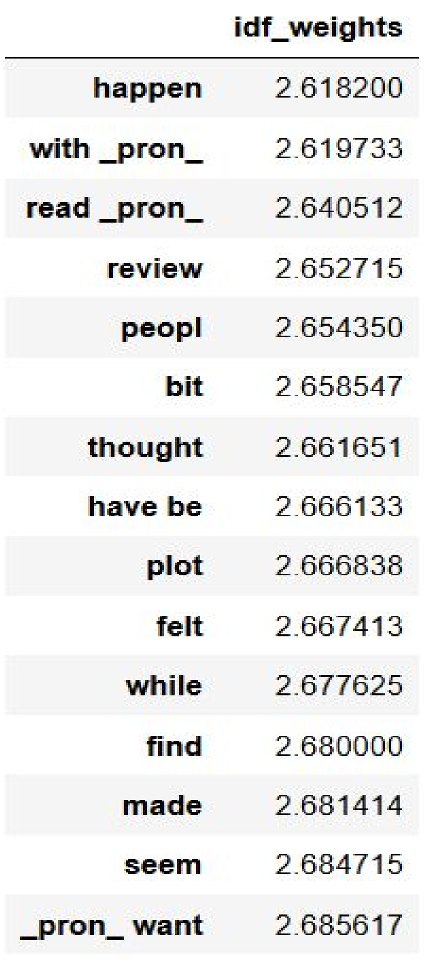
After splitting out the data, I ran a count vectorizer on the text with the parameters requiring that the phrase (or word) not appear in more than 20% of the text and appear at least 400 times. I chose to use only single words and bigrams as trigrams required far too much memory.

TFIDFVectorizer:
For the TFIDFVectorizer, I used the same constraints as my CountVectorizer, producing a vocabulary of slightly over 34,000. See the code below:



```
#I use 20% because using bigrams which are less likely to be common
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(sublinear_tf=True, use_idf=True, max_df = 0.20, min_df = 400, ngram_range = (1,2),
                                   lowercase = False)
```



	idf_weights
happen	2.618200
with _pron_	2.619733
read _pron_	2.640512
review	2.652715
peopl	2.654350
bit	2.658547
thought	2.661651
have be	2.666133
plot	2.666838
felt	2.667413
while	2.677625
find	2.680000
made	2.681414
seem	2.684715
pron want	2.685617

I sorted by IDF-weights to see the most common words and phrases after the preprocessing. These words tended to be generic and commonly used, although not common enough to be filtered out by the vectorizer. Most were not unique to book reviews except for the words “review,” “read,” and “plot.”

TruncatedSVD:
I also reduced the dimensionality of the TFIDF vector using TruncatedSVD with 300 components and 10 iterations. However, there was a lot of information lost reducing 34,000 words to 300 features and the conversion of the sparse TFIDF vector to a non-sparse SVD matrix hurt computation times, so I decided to have my final models from the CountVectorizer and TFIDFVectorizer vectors only take in sparse matrices.

Doc2Vec:
I used gensim’s simple_preprocess function to process the text prior to tagging them for creating the Doc2Vec corpus for training the model. I then used both the distributed bag of words (DBOW) and a distributed memory (DM) model to transform the documents into vectors. All vectors had 200 dimensions.

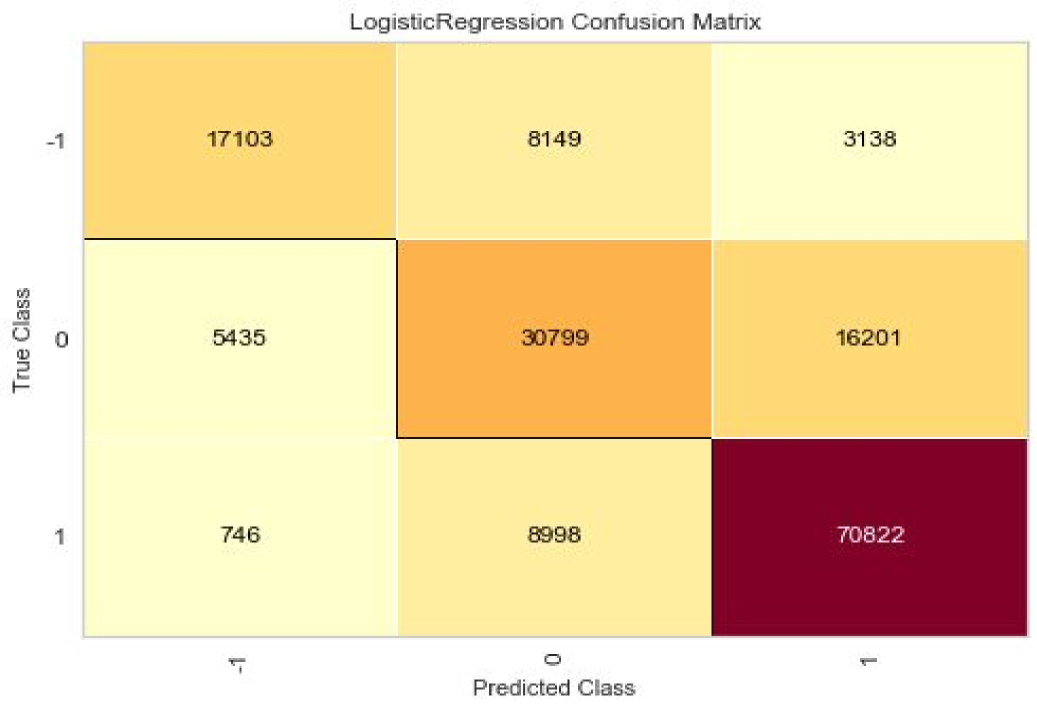
LSTM:
For the LSTM model, I used the standard keras preprocessing. I turned each comment into a list of word indices of equal length of 400. This max length was to capture the whole of a vast majority of reviews (75th percentile of review lengths was slightly over 300). I used the 300 dimension GloVe word vectors to create the embedding matrix with random initialization for words that aren’t included in GloVe word vectors.

Machine Learning:

CountVectorizer:

I used logistic regression to model my CountVectorizer vector as it is commonly used for sentiment analysis and it is able to accept sparse matrices. I chose accuracy as the primary metric as correct classification is most important for sentiment analysis problems.

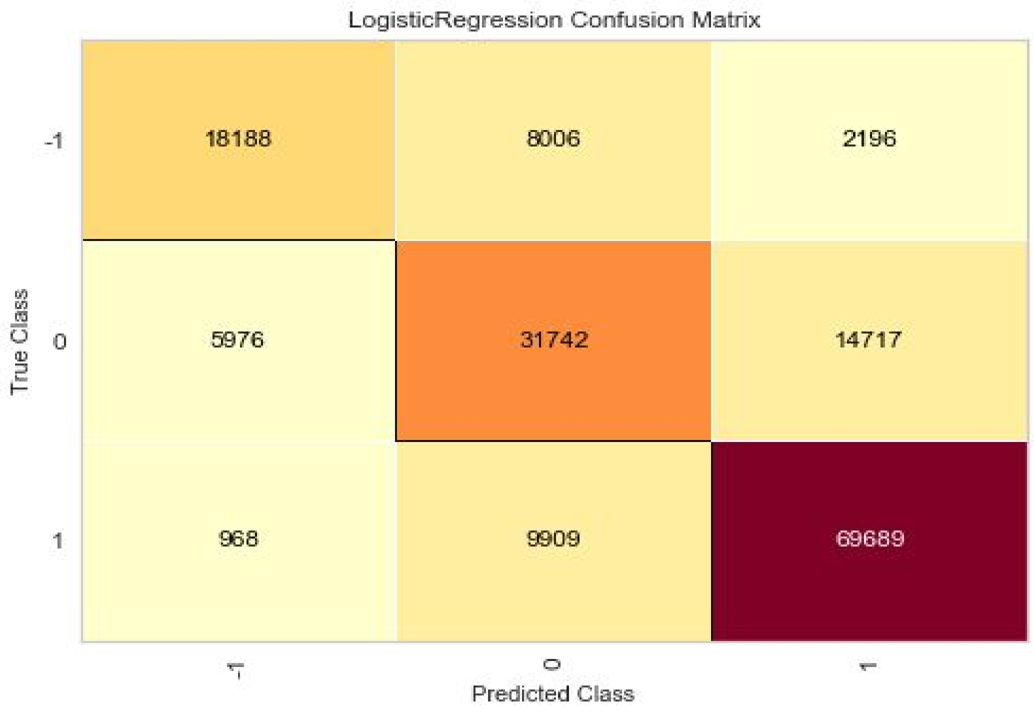
The model using all 34,000 features was able to produce an accuracy score of just under 73.6%. The model struggled with the neutral class especially.



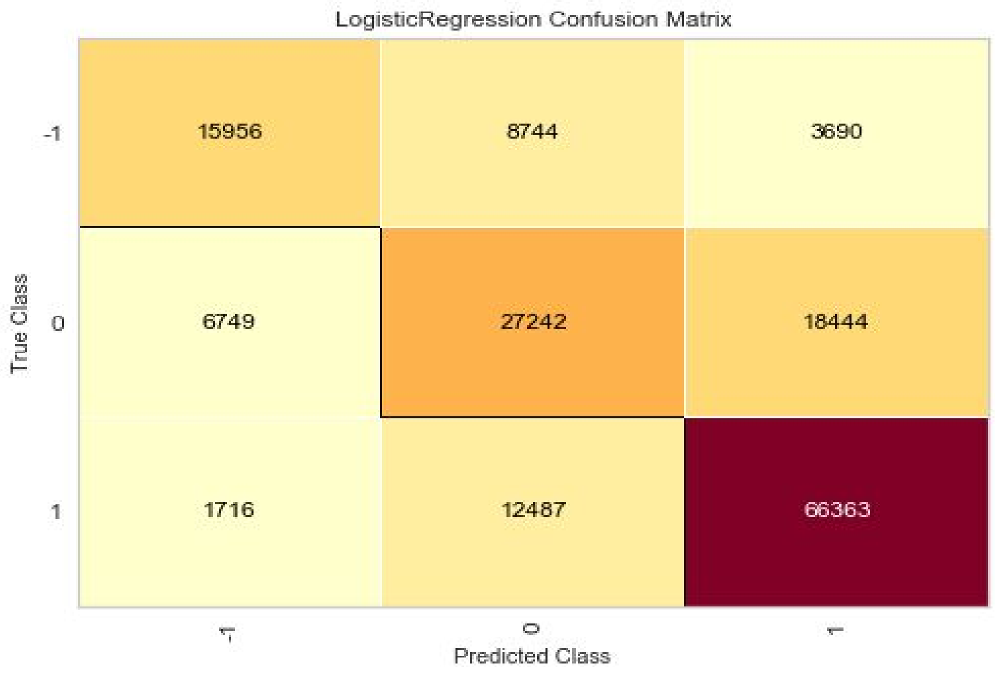
TFIDFVectorizer:

I used logistic regression to model my TFIDFVectorizer similar to the CounterVectorizer above. The model using all 34,000 features was able to produce an accuracy score of 74.1% vs 73.6% for the CountVectorizer

While the accuracy scores were not majorly different, the TFIDF model was less inclined to predict the dominant class (positive). It also had a significantly lower log loss score (0.589 vs 0.625). However, similar to the CountVectorizer, despite performing better, the model struggled mightily predicting the neutral class.

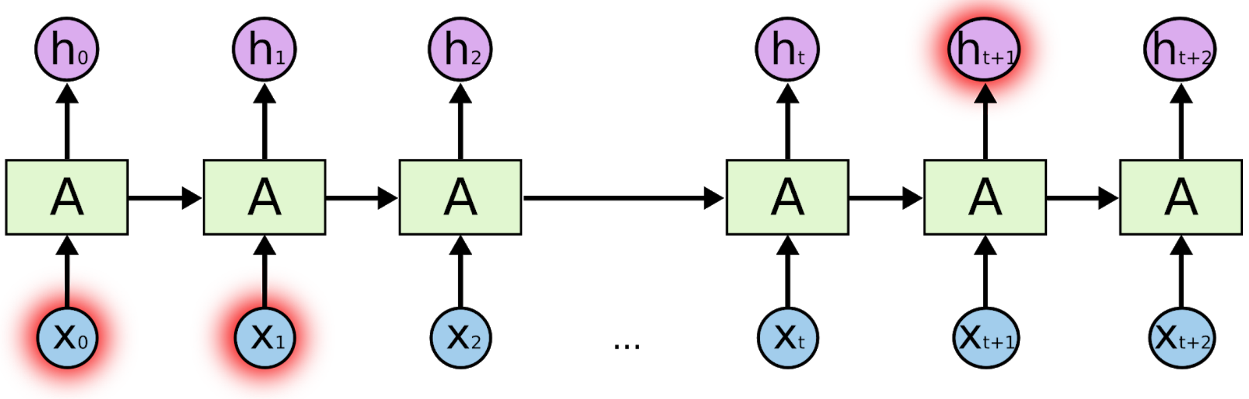


Doc2Vec:
I used logistic regression for the Doc2Vec vectors due to the speed and the fact that tree based models would not make sense with document vectors where every feature is important. The unsupervised Doc2Vec vectors did not perform well in classifying the sentiment. The best model (concatenation of DBOW and DM) produced an accuracy score of below 68%. Once again, the model struggled with the neutral class, barely reporting over 50% accuracy.



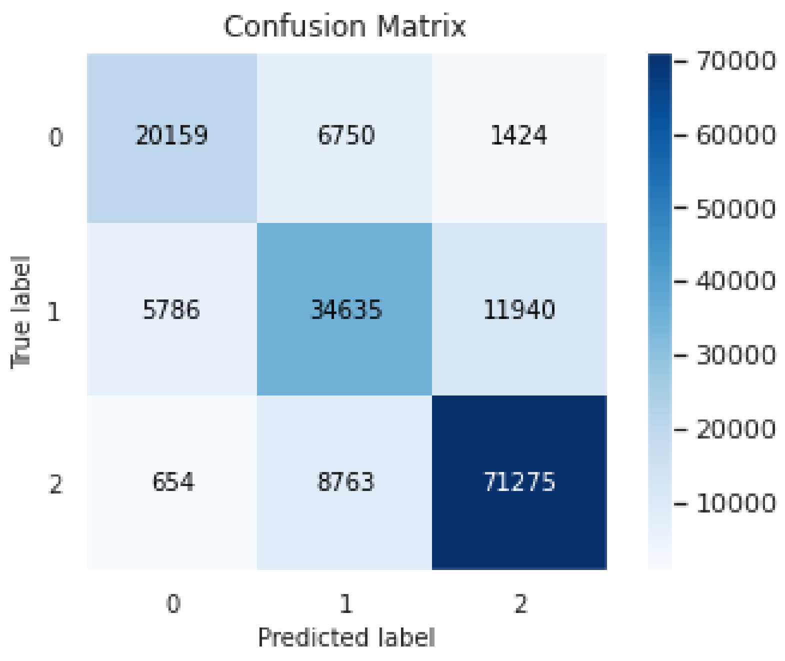
LSTM:

Long short term memory networks (LSTMs) are a special type of recurrent neural network which excel at handling "long-term dependencies." They are able to remember information for long periods of time which can be valuable for text data as relevant information may be separated by quite a few words or tokens.

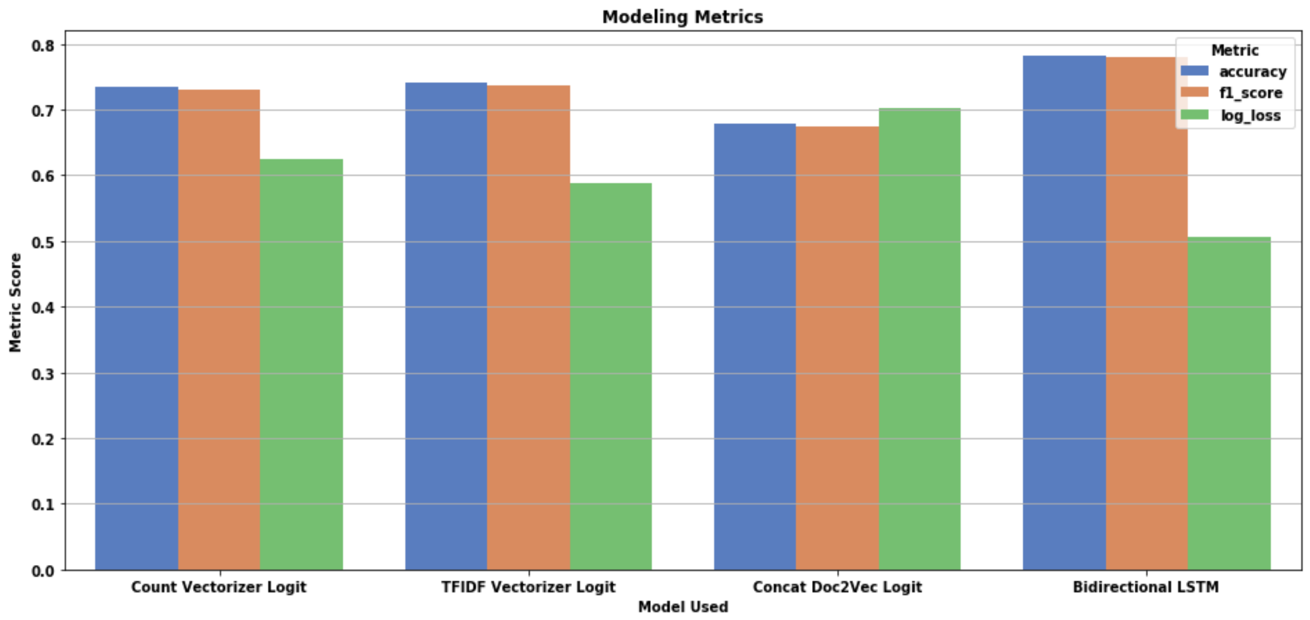


I padded or truncated each review to 400 tokens as that should capture all of the vast majority of words, and used the 300 dimensional GloVe word vectors to create the embedding matrix.

I experimented with different levels of dropout and batch sizes, but the models tended to overfit after the second epoch for most of the models I tried.



The LSTM model was the first model that had decent results with the neutral class. This is probably due to it being a neural network learning the connections between words rather than just a form of counting of words. My final LSTM model was able to predict with over 78% accuracy with significantly less log-loss than any other model indicating it was better at picking up patterns and connections between words rather than just counting words.



Conclusion:

The Goodreads reviews offered a large corpus with which to test many different NLP techniques. While I was not particularly surprised with the accuracy of any of the models. I was relatively surprised at how well the TFIDF logistic regression model compared with the deep-learning LSTM model despite the vast difference in sophistication of the models. However, the LSTM model was clearly the most useful as it tended to be far less wrong (less loss) than any other model indicating a deeper level of “learning.”

I was also surprised at how poorly the Doc2Vec vectors performed in sentiment analysis compared with simpler techniques even after concatenating the two different models.

There are many possibilities for additional work and experimentation including different forms of text preprocessing. To address the neutral class performance, I could isolate the neutral class to help all models learn what distinguishes neutral from positive/negative. I could have also tried more parameters in the LSTM model, but computation time was definitely a limitation for that type of model. Finally, I could try different deep-learning models that are known to perform well with sentiment analysis including convolutional neural networks (CNNs).