Capstone 2 – Milestone Report 2

Conducting NLP on Amazon Store Review Data

Zach Palamara

Project Aim and Background

Problem Statement

The goal of this project was to perform sentiment analysis on product reviews in the Amazon store. The target audience for this project are sellers on the Amazon store. They will be able to use this model to analyze consumer behavior and their response to products. This information will be useful in deciding on what new products to bring to the market and how to improve existing products. Additionally, this data can be used to help recommend products to consumers based on their behavior tendencies.

Dataset and Source

I am using a data set from the following website: [Amazon Review Data (2018)]. The data is in a .json format and can be easily read into a Pandas df for EDA and pre-processing. Each review is a single record in the dataset containing the following features:

- reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
- asin - ID of the product, e.g. 0000013714
- reviewerName - name of the reviewer
- vote - helpful votes of the review
- style - a dictionary of the product metadata, e.g., "Format" is "Hardcover"
- reviewText - text of the review
- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (unix time)
- reviewTime - time of the review (raw)
- image - images that users post after they have received the product

Project Goals

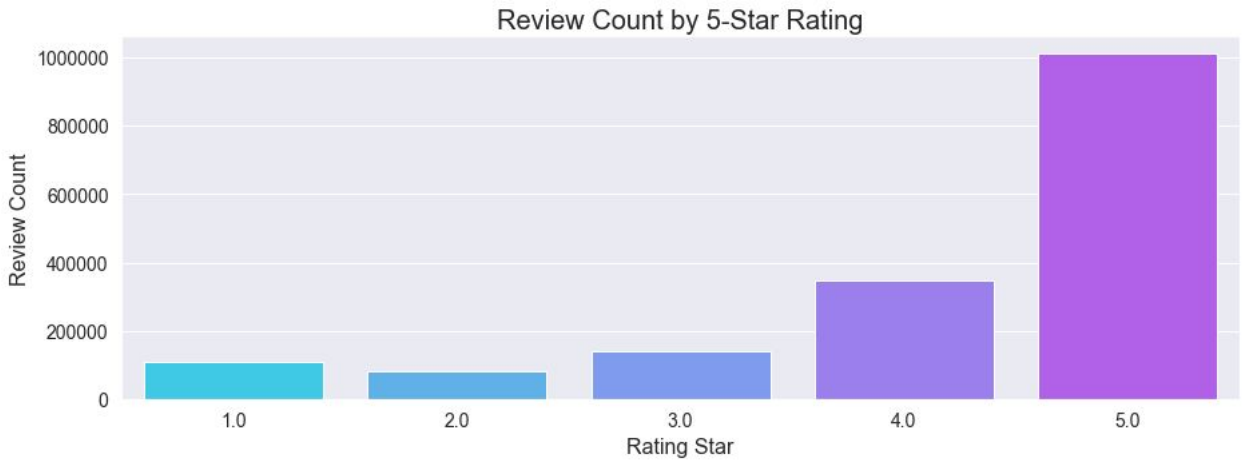
I used Natural Language Processing (NLP) to solve this problem. More specifically, I used Fine-Grained Sentiment Analysis which is a common method for determining sentiment in 5-Star reviews. After performing EDA I tested out a variety of methods for feature engineering. Some of the methods I used include text vectorization using a bag-of-words or bag-of-ngrams. After the features were properly cleaned and engineered, I used three different classifiers to construct my machine learning models for text and numerical features.

Table of Contents

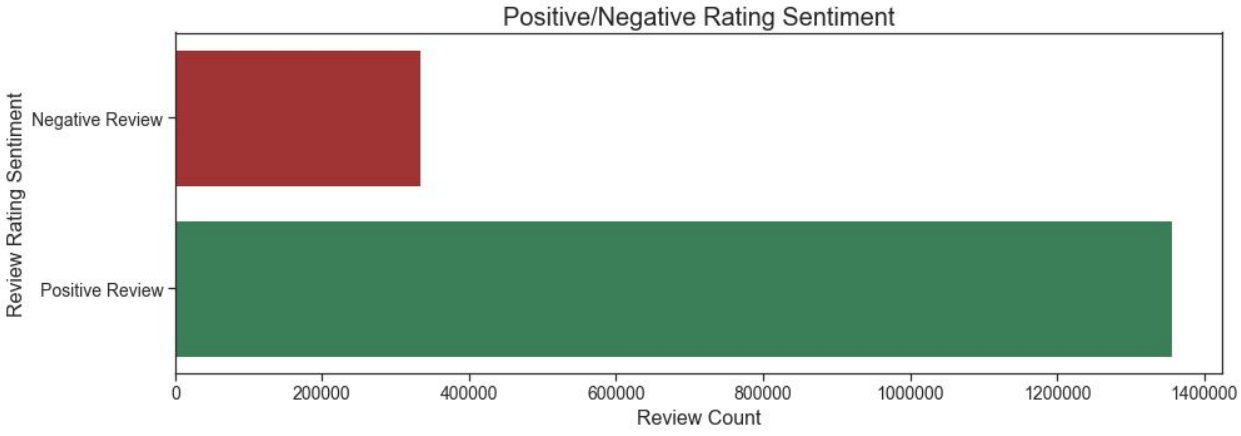
Data Cleaning and EDA

Initial Findings

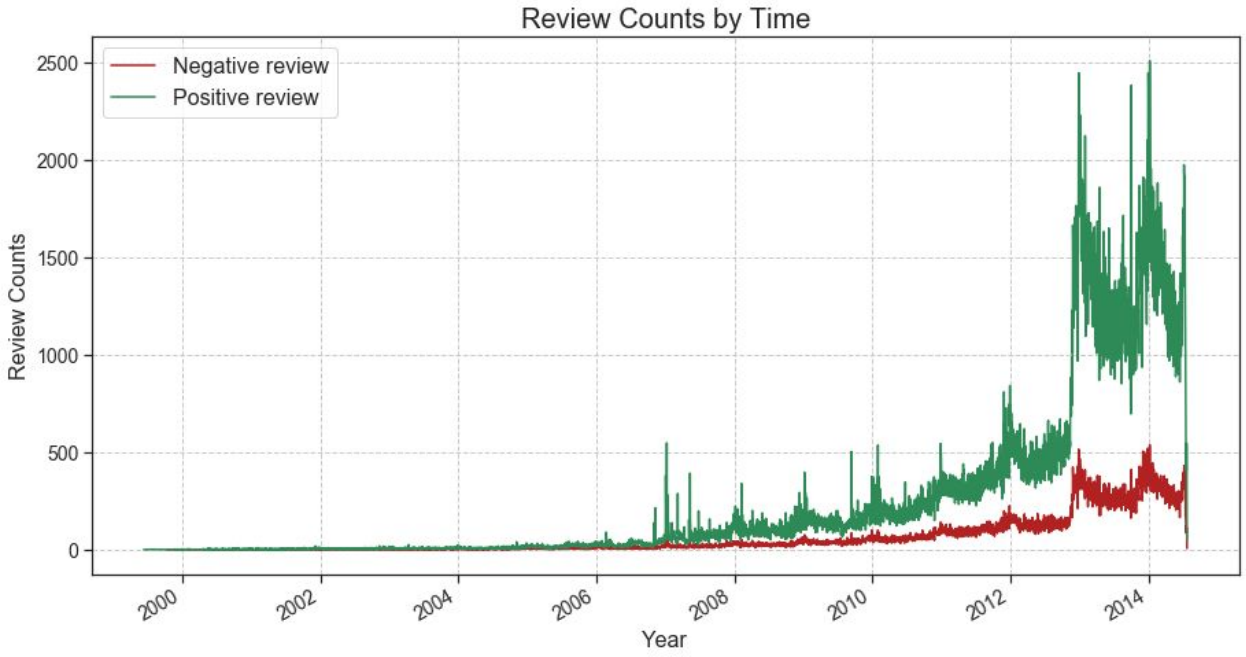
The first step I took in examining the data was to look at a simple count plot of reviews broken out by their respective 5-star rating.



This figures shows that most of the reviews are positive and overwhelmingly 5-stars. Additionally, I created a figure that depicts the count of negative and positive reviews. For this, I created a new binary target feature, which represents 1,2 and 3 star reviews as 0 or “Negative” and 4 and 5 star reviews as 1 or “Positive”.



When viewing positive and negative review counts in time-series, there’s a clear jump in the volume of reviews around the start of 2013. Also, it appears that positive sentiment seems to diverge from negative sentiment with time.



Text Feature Engineering

Removing Stop Words

One of the most important steps in any NLP problem is to remove “stop words” within a corpus. Stop words are words such as “the”, “a” and “are”, which are common words that do not really provide any significant meaning to our features. It is important to remove these words, because they take up unnecessary space in memory and processing power. I utilized the Natural Language Toolkit (NLTK) framework to remove stop words and punctuation in this dataset.

Tokenization

Once the stop words were removed, each review for each data record needed to be tokenized. Tokenization is a way of separating a large text document into tokens, which in this case would be singular words. I utilized the word_tokenize() method provided by the NLTK framework.

Lemmatization

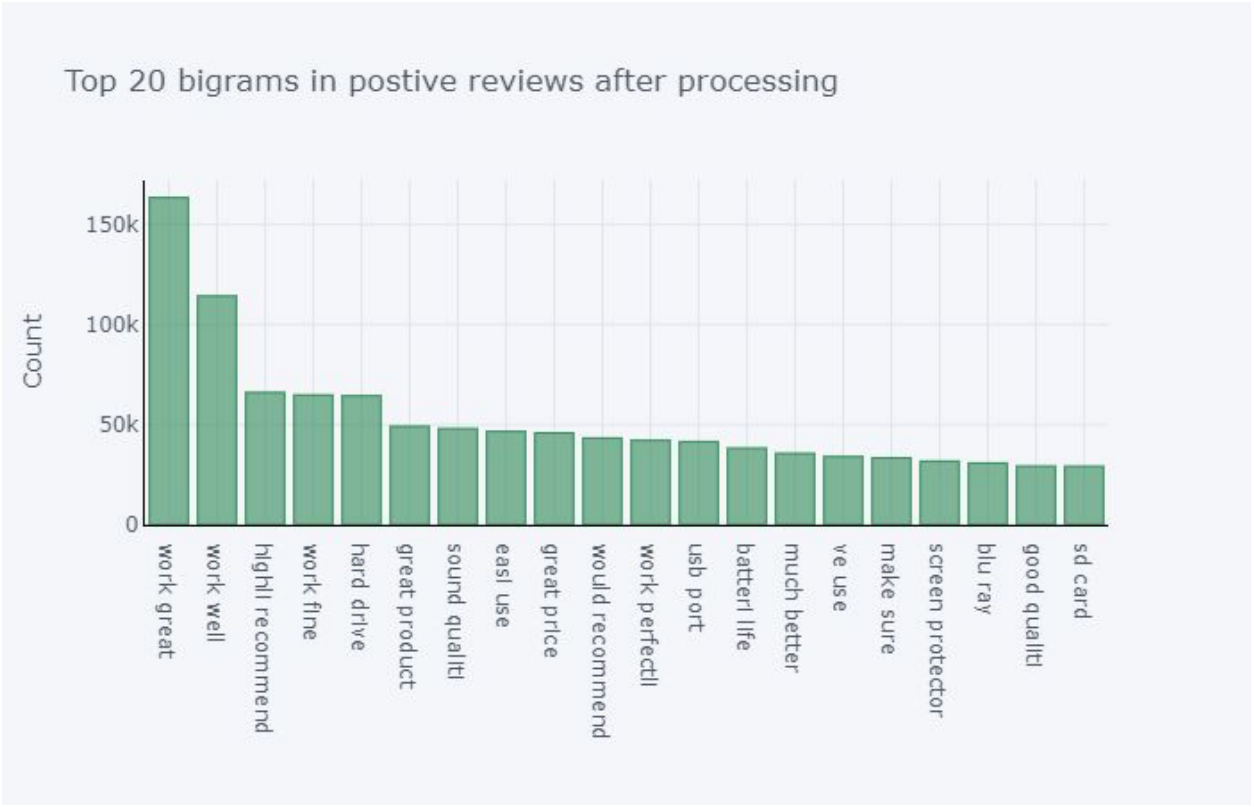
The next step in engineering the text features involved lemmatization. In its simplest form, lemmatization is a process in which different forms of the same word are mapped to a singular form of that word. This is also very important in trying to reduce the size and memory space that your data set takes up. For example, the words “running”, “ran” and “run” all get mapped to a singular form of “run”. This was executed using the PorterStemmer class from the NLTK framework.

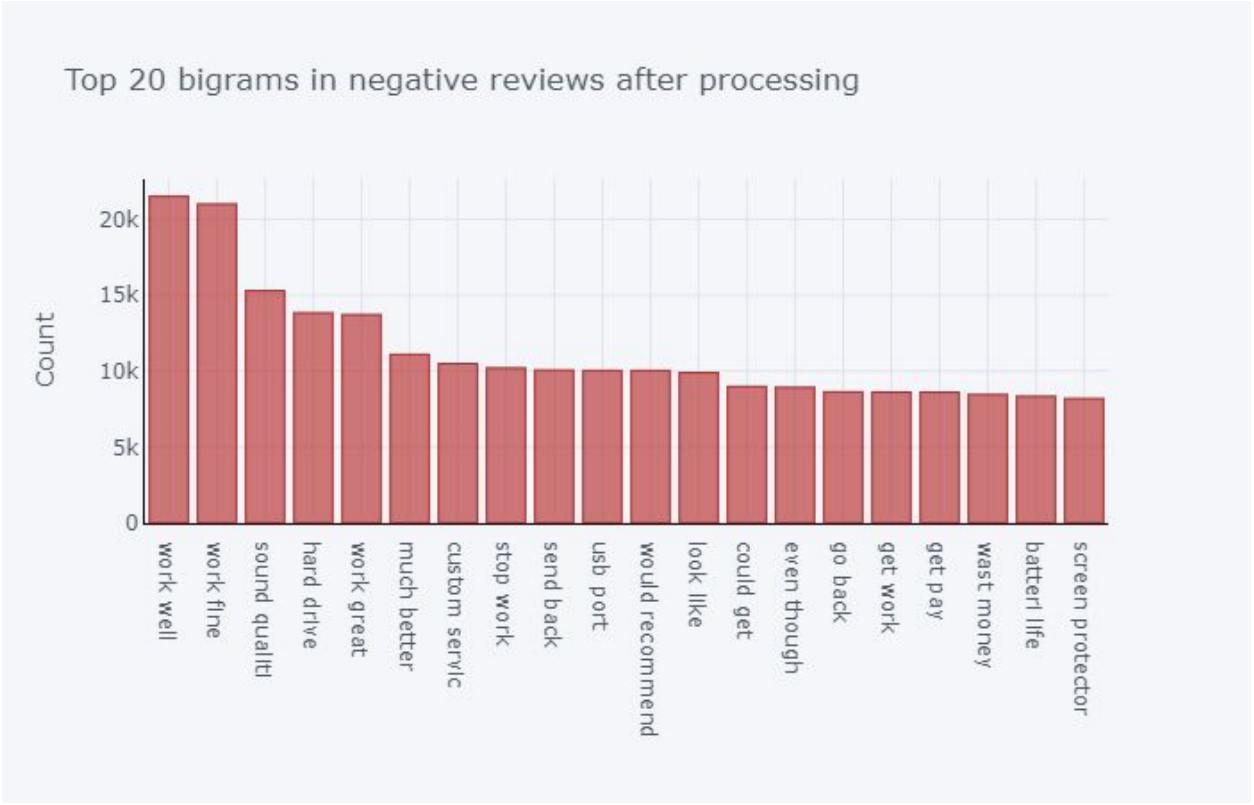
Stemming

The final step I took in engineering the review data involved stemming. Stemming simply removes prefixes and suffixes from tokenized words. Similar to Lemmatization, this is also done in an effort to reduce the size and complexity of the data. This was also executed using the WordNetLemmatizer class from the NLTK framework.

Extracting N-Gram Features

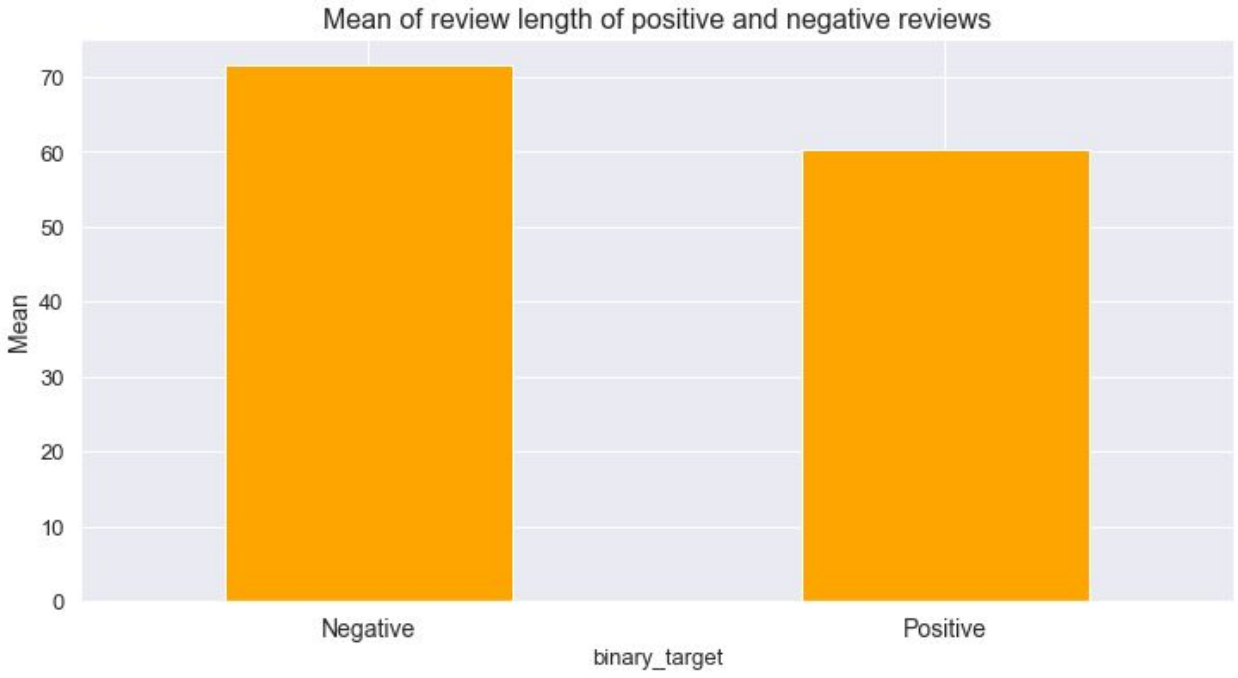
In the simplest of terms, N-grams are a statistical representation of the likelihood that a series of words appear in a text document. The letter “N” in N-grams represents the number of words in the series. For example, I created lists of the top 20 Bigrams and Trigrams of both positive and negative reviews. The results can be seen in the figures below.

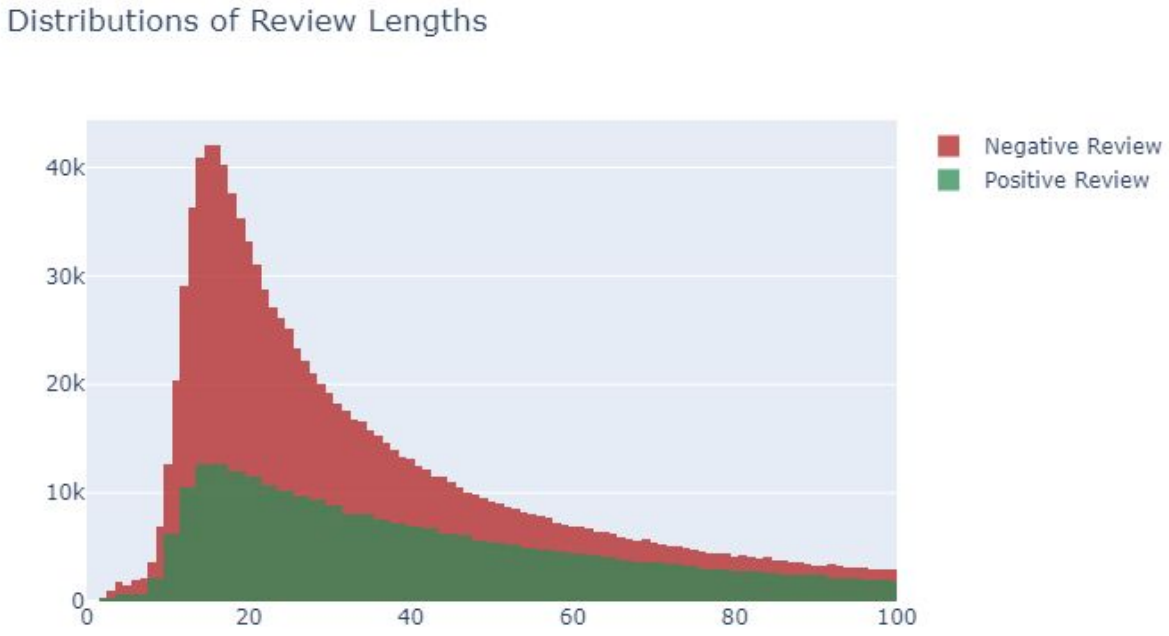

Top 20 bigrams in postive reviews after processing
Count
150k
100k
50k
0
work great
work well
highl recommend
work fine
hard drive
great product
sound qualitl
easi use
great price
would recommend
work perfectl
usb port
batteri life
much better
ve use
make sure
screen protector
blu ray
good qualitl
sd card


Top 20 bigrams in negative reviews after processing
Count
20k
15k
10k
5k
0
work well
work fine
sound qualitl
hard drive
work great
much better
custom servic
stop work
send back
usb port
would recommend
look like
could get
even though
go back
get work
get pay
wast money
batteri life
screen protector

Review Length

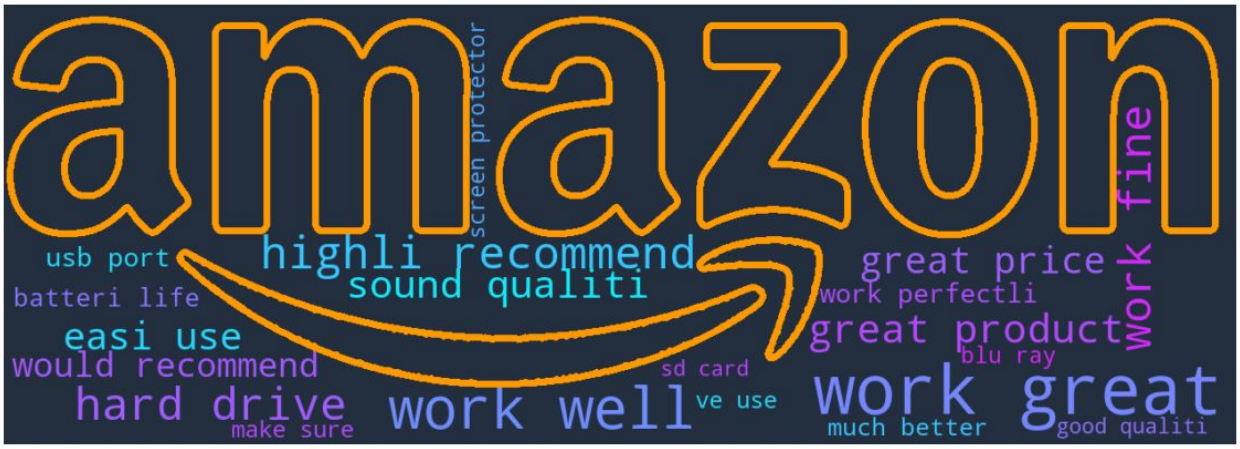
On average, negative reviews seem to be longer than positive reviews. Intuitively, it makes sense that people tend to be more vocal expressive about something that displeases them. Additionally, both positive and negative reviews seem to have right tail distributions.





Word Cloud

One of the most popular ways to visualize common words in an NLP project is through a word cloud. Pictured below is a word cloud containing the 20 most common bigrams found in positive reviews.



Model Preprocessing

Bootstrapping Samples

"Bootstrap sampling is a resampling method by independently sampling with replacement from an existing sample data with same sample size n, and performing inference among these resampled data." - Lorna Yen, "An Introduction to the Bootstrap Method"

Knowing that my classes of positive and negative reviews were unbalanced, I decided to use a bootstrap sampling method on the negative reviews to get a balanced number of positive and negative samples. I used the resample function from the sklearn framework to accomplish this. The code implementation can be seen (below).

```
# Step 1: Counting positive and negative reviews
count_dict = df.binary_target.value_counts().to_dict()
>>> {0: 333121, 1: 1356067} # 0:Negative, 1:Positive
```

```
# Step 2: Bootstrapping negative reviews
neg = df[df['binary_target'] == 0]
neg_bs = resample(neg,
                  replace=True, # Sample with replacement
                  n_samples=count_dict[1], # Number of positive reviews
                  random_state=123)

# Step 3: Combining positive reviews and bootstrapped negative reviews
pos = df[df['binary_target'] == 1]
bs_df = pd.concat([pos, neg_bs])

# Step 4: Getting new counts of positive and negative reviews
bs_count_dict = bs_df.binary_target.value_counts().to_dict()
>>> {0: 1356067, 1: 1356067} # 0:Negative, 1:Positive
```

Now the positive and negative review samples are balanced. This is especially a very important step for implementing Deep Learning models, because if your dataset is unbalanced, the model will learn to always pick the class that is overbalanced, which is not the goal.

Shuffling Dataset

After balancing the dataset, the next step I took in the preprocessing phase was to shuffle each sample. This is always a good idea, because if your features are ordered in a particular way, you can unknowingly introduce bias while feeding them into sequential models. I executed this step using the shuffle() function from the sklearn framework.

Text Modeling

The first series of models that I developed were trained using only the review text to predict whether or not the review was positive or negative. To do this, I had to vectorize each word using the CountVectorizer() class from sklearn. *"This implementation produces a sparse representation of the counts using"* - SKLearn (CountVectorizer)

After vectorizing the text features I split the data into training and testing sets. I used a test size of 20% and set the random_state equal to 123 for reproducibility.

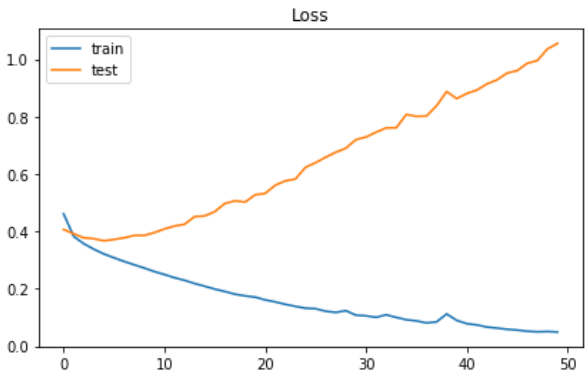
```
# Creating training and test sets
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=123)
```

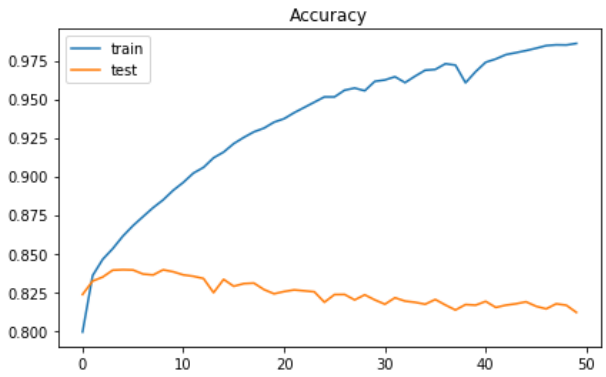
Naive Bayes Classifier

The first model that I decided to implement utilized a Multinomial Naive Bayes classifier. Naive Bayes is a commonly used algorithm that typically performs very well in NLP classification. *"Naive Bayes classifiers are based on Bayes theorem, a probability is calculated for each category and the category with the highest probability will be the predicted category."* - Naive Bayes for text classification in Python

Deep Neural Network

The second model that I developed was a deep neural network (DNN). In simple terms, neural networks operate by accepting input data at the input nodes on one side of the network. The input data is then passed through multiple hidden layers of nodes that assign weights to each value with the ultimate goal of minimizing loss. Finally, once the data is passed through each layer, it encounters an activation function, which attempts to predict the final result. More information on Deep Neural Networks and the type of model that I implemented can be found through this link. Practical Text Classification With Python and Keras Also, I only trained this DNN on a ~3% sample of my data set. This was due to computation constraints.





Logistic Regression

The third classifier that I used on the text data involved Logistic Regression. Logistic Regression is a basic, but effective classifier that is used to predict either binomial or multinomial classes. Ulike deep neural networks, it is relatively fast and uncomplicated. More information on Logistic Regression and how I applied it in my model can be found through this link. [Logistic Regression in Python]

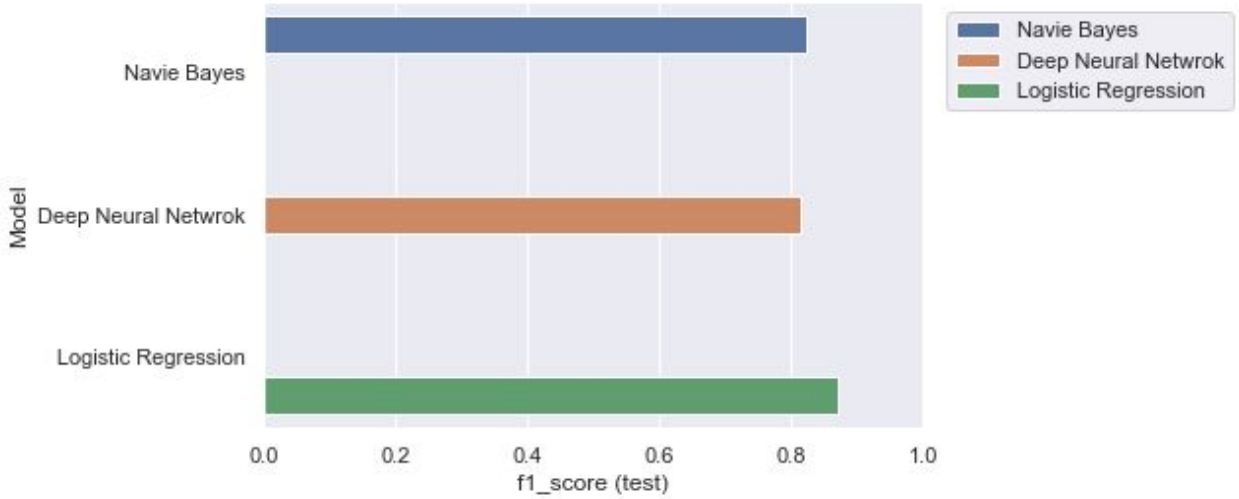
Results for Text Models

The f1 scores and computation time can be seen in the figure below. I chose the f1 score metric as the metric to compare these models, because it takes into account both precision and recall.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Model	F1 - Score	Wall Time
Naive Bayes	0.8225	1min 26s
Deep Neural Network	0.8134	~ 24 hrs (for full data set)
Logistic Regression	0.8721	14min 46s

Conclusions



Based on the results, it seems like Logistic Regression may be the best classifier for this particular type of review data. Not only did it have a much better F1-Score than both the Naive Bayes model and DNN, it took a reasonable time to fit the model. However, the results for the DNN are based on only a 3% sample of the entire dataset. Moving forward I would like to be able to train the DNN on all the review data, but since the dataset is so large I will likely need a GPU to accelerate the training process. It's quite possible that the DNN may perform better after training on all of the data.

Sources

[A Complete Exploratory Data Analysis and Visualization for Text Data | by Susan Li | Towards Data Science]

[Language Models: N-Gram. A step into statistical language… | by Shashank Kapadia | Towards Data Science]

[nltk.stem.porter — NLTK 3.5 documentation]

[Naive Bayes for text classification in Python | A Name Not Yet Taken AB]

[Practical Text Classification With Python and Keras – Real Python]

[Logistic Regression in Python – Real Python]