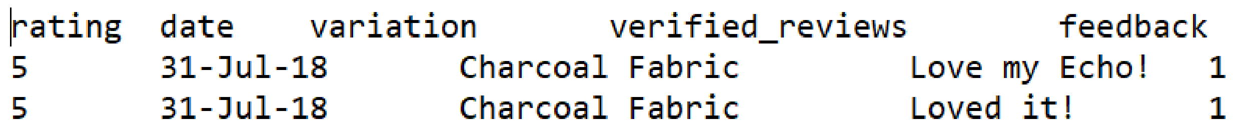
Becca Dura
CS167 Project D

Problem

The dataset I chose to perform text analysis on has 3,149 Amazon customer reviews on various Amazon Alexa products, such as the Alexa Echo and Alexa Firestick (see example below). It includes each customer's verified review, star rating (out of 5), date of the review submission, product variant, and feedback. This dataset is from Kaggle(https://www.kaggle.com/sid321axn/amazon-alexa-reviews) but was extracted directly from Amazon's website. I used this dataset for sentiment analysis purposes. In other words, I used the customer reviews to create a model that predicts the rating the customer gave the product. This will help Amazon better understand the correlation between a customer's review and product rating. A lot of times, customers will not give a product a rating, but will just write a review. This model will help Amazon be able to better predict what rating the customer would have given the product based off the review's sentiment, wording, and tone. Therefore, I used a dataset containing customer reviews on Amazon's Alexa products to create a model that predicts the rating of a product based on the written customer review.



```
rating  date      variation        verified_reviews        feedback
5       31-Jul-18       Charcoal Fabric         Love my Echo!   1
5       31-Jul-18       Charcoal Fabric         Loved it!       1
```

Data Prep

The dataset first had to be read in as a tab-delimited file instead of a comma-separated value like most files we have worked with this year. Since the reviews did not have any html markups, I did not need to use the Beautiful Soup package to clean up the data like we did with past text files. However, I did remove all characters that were not letters (i.e. commas and other punctuation) in each Amazon review by substituting the non-letter characters with a space. Then, I used split to tokenize each customer review. I created a new array, called filtered, to hold the words that had been tokenized that were not stop words. I removed the stop words (words in the English language that do not have much meaning according to the nltk stop words list), so I was only using words with meaning and converted the remaining words into one string for each customer review.

After cleaning up the customer reviews and preprocessing them, I split the dataset into a train and test set, with the test set containing 20% of the data. I used the verified customer reviews as the predictor and the product rating (out of 5) as the target variable. Then, I created a bag of the 5000 most common words by fitting a vectorizer to the train set. To do that, I used Count Vectorizer to turn each string, or customer review, into rows of word counts to find the 5000 most common words, not including stop words, in the train set. Once I fit the vectorizer to the train data and had the 5000 most common words, I transformed my data. Finally, I transformed each review string, in both the train and test set, into rows of word counts and converted it to a numpy array, which will allow it to be fed into a machine learning algorithm.

Experiment & Results

After preprocessing the data, prepping it to be suitable for a machine learning algorithm, splitting it up into a train and test set, and fitting a vectorizer to the train set that I used to transform the train and test set, I tried multiple variants of the machine learning algorithm. My main goal through experimenting with the variants was to increase the accuracy when predicting the reviews' ratings as much as possible. I tried feeding the dataset into various Naïve Bayes algorithms as well as a Support Vector Machine. I also tried feeding it into the same algorithms after performing Principal Component Analysis (PCA).

Out of all the variants I tried without Principal Component Analysis, the Multinomial Naïve Bayes variant produced the best performing model. Its accuracy when predicting the product rating was 75.40%, while the Bernoulli Naïve Bayes and Support Vector Machine variant produced accuracies of 71.59%. It makes sense that Multinomial Naïve Bayes worked better than Bernoulli Naïve Bayes because, although they are both frequently used for text classification, Bernoulli Naïve Bayes is better for shorter documents or smaller amounts of data. It also makes sense that a Support Vector Machine worked okay, but was not the best, because it finds the linear separator with the largest margin (greatest space between classes). Since there are 5 classes, it is probably harder to find linear separators for every class (all 5 ratings), meaning the Support Vector Machine would not be as accurate as if there were fewer classes to separate. The Gaussian Naïve Bayes variant produced the worst performing model with an accuracy of 40.32%, which made sense since it is not used for text classification as often. Since the Multinomial Naïve Bayes variant produced the most accurate predictions for the product ratings, I decided to mess around with the parameters and see if I could make the model even better. I discovered that changing alpha, the smoothing parameter which ranges from 0-1, increased the accuracy. The default is 1, but when I lowered alpha to decrease the amount of smoothing, the accuracy increased. The best accuracy I got was 77.78% with an alpha of 0.3.

Although that accuracy was good, I decided to try some other variants. I tried the same variants, but with Principal Component Analysis. Support Vector Machines with PCA seemed to work well, with an accuracy of 75.71%, but the model did not perform as well as the Multinomial Naïve Bayes variant without PCA. PCA with Naïve Bayes did not work well at all. Multinomial Naïve Bayes cannot be mixed with PCA here because the numbers must all be positive for Multinomial Naïve Bayes. As for Bernoulli Naïve Bayes with PCA, the accuracy was 66.51%, which is not great, and the accuracy for Gaussian Naïve Bayes with PCA was even lower (60.95%). In other words, adding Principal Component Analysis to each of the variants does not really help my model perform better. This makes sense since Principal Component Analysis is used to reduce the dimensionality of the data and reduce noise, but this dataset does not have a lot of features used to predict the product rating.

Finally, I decided to try predicting the product rating using a random forest variant. The accuracy of my model with the defaults for the Random Forest Classifier was 80.32%, which is very good compared to the rest of the variants. This is most likely because random forests build numerous decision trees and use them to vote on the prediction. Since there are only 5 possible prediction classes, using a random forest worked well and ensured that noise (unmeaningful words) did not lead to an incorrect prediction. The random forest variant also worked well because there were not a lot of irrelevant features in the dataset. I decided to try messing around with the number of decision trees (n_estimators) in the random forest to find the optimal value. The highest accuracy I got was 80.95% with 100 decision trees in the random forest. To conclude, the random forest variant of my model with 100 decision trees was the best at predicting the product rating (out of 5) based off the customer reviews of Amazon Alexa products.