Natural Language Processing – Amazon Alexa Reviews

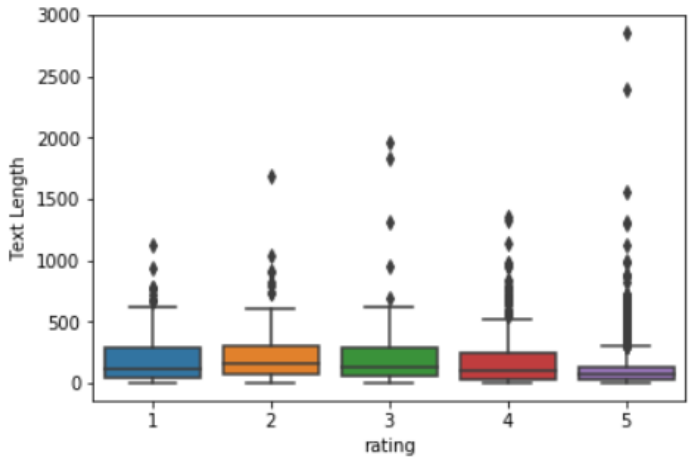
Summary
We have built a text classifier – a sentiment analysis model to predict whether the customer reviews are positive or negative. For the products/services/brands with large customer bases, it is impossible to manually glean negative reviews (or positive ones) from huge datasets.

Using Natural Language Processing (NLP), businesses can automate the process of highlighting negative reviews and thus can effectively uncover the major gaps in their products/services; however, it is important to note that frequency of positive and negative reviews plays an important role determining the performance of the model. In our data set, positive reviews outnumbered negative ones comprehensively (Positive: 2286, Negative: 161); hence, the accuracy while predicting positive reviews was significantly superior.

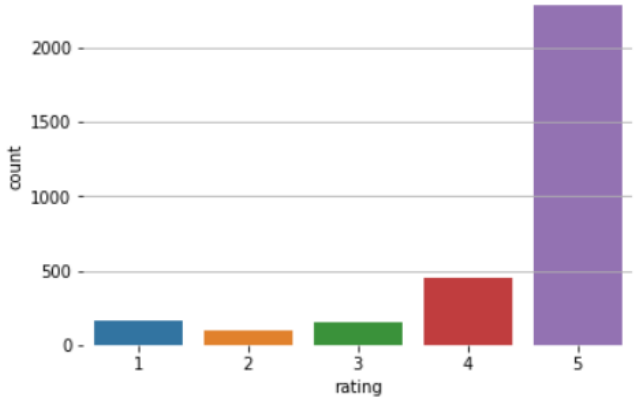
Data Source: https://www.kaggle.com/sid321axn/amazon-alexa-reviews

Analysis

1. We created a column to store the text length and created boxplots of text lengths for different ratings
    a. We observed that the text length for rating 5 is smaller (excluding outliers) than those of lower ratings. This could be because people are more descriptive/vocal in negative feedbacks, but not so much in positive ones



2. We then also created a bar chart to know the count of the different ratings. The count of 5 star ratings was much higher than those for other ratings. Hence, our model was better at predicting the ratings for positive reviews.



3. To build our model, we have used only the 1 star and 5 star ratings

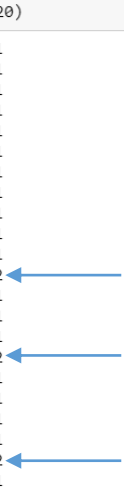
4. We then created a function (word_list) that would remove all the punctuations and stopwords, and return a list of important words. Following is an example of how the function works.

```
In [56]: # Following is an example of how the function works

example = "Hello! We need to remove the punctuations and additional characters - such as \,*"
print(word_list(example))

['Hello', 'need', 'remove', 'punctuations', 'additional', 'characters']
```

5. We have used Count Vectorizer object to get the weights of the tokens – the number of times a token appear in a document
    a. Following is an example of the number of unique words and their frequencies shown by the vectorizer

```
In [58]: # To see how vectorizer shows the number of unique words and their frequency


# Following is the review
review_10 = X[10]
review_10

Out[58]: "Great product and I would give 5 stars - but you can't scroll face cards without having the stupid &#34;try and ask Alexa&#
34; suggestions pop up.  Yes you can have it scroll once, and just stay on the clock, but I like having other cards as well.
God its the worst and SO irritating.  I got it super cheap, so i just face the screen toward the wall and treat it like a Do
t instead of a Spot.  What a dumb move on Amazons? part."

In [59]: transform_20 = text_matrix.transform([review_10])
transform_20

Out[59]: <1x3986 sparse matrix of type '<class 'numpy.int64'>'
        with 39 stored elements in Compressed Sparse Row format>
```

```
In [60]: print(transform_20)

  (0, 70)       1
  (0, 88)       1
  (0, 142)      1
  (0, 162)      1
  (0, 311)      1
  (0, 418)      1
  (0, 429)      1
  (0, 754)      1
  (0, 903)      1
  (0, 1080)     1
  (0, 1294)     1
  (0, 1303)     2
  (0, 1349)     1
  (0, 1377)     1
  (0, 1724)     1
  (0, 1877)     2
  (0, 2026)     1
  (0, 2048)     1
  (0, 2258)     1
  (0, 2304)     1
  (0, 2430)     2
  (0, 2595)     1
  (0, 2796)     1
  (0, 2892)     1
  (0, 2946)     1
  (0, 3215)     1
  (0, 3221)     2
  (0, 3449)     1
  (0, 3460)     1
  (0, 3490)     1
  (0, 3508)     1
  (0, 3510)     1
  (0, 3660)     1
  (0, 3677)     1
  (0, 3841)     1
  (0, 3879)     1
  (0, 3910)     1
  (0, 3929)     1
  (0, 3931)     1
```



This review had 39 unique words and 4 of them were repeated twice (shown by the blue pointers)

Following are the words

```
In [61]:
# The review has 39 unique words
# 4 of the unique words qppear twice
# Let's check what those word are

print(text_matrix.get_feature_names()[1303])
print(text_matrix.get_feature_names()[1877])
print(text_matrix.get_feature_names()[2430])
print(text_matrix.get_feature_names()[3221])
```

```
cards
face
like
scroll
```

6. The shape of our sparse matrix is 2447 * 3986, and it has 25728 non-zero values

```
In [63]:
# Shape of the matrix and the number of non-zero values

print('Sparse Matrix Shape: ', X.shape)
print('Non-Zero occurrences = ', X.nnz)
```

```
Sparse Matrix Shape:  (2447, 3986)
Non-Zero occurrences =  25728
```

7. Prediction Model – Multinomila Naive Bayes

Confusion Matrix and Classification Report

```
Confusion Matrix
[[ 19  26]
 [  6 684]]
```

Classification Report

	precision	recall	f1-score	support
1	0.76	0.42	0.54	45
5	0.96	0.99	0.98	690
micro avg	0.96	0.96	0.96	735
macro avg	0.86	0.71	0.76	735
weighted avg	0.95	0.96	0.95	735

In the confusion matrix, it can be seen that the recall rate for 5 star ratings is 99%, but it is just 42% for 1 star ratings – with the overall accuracy of 96%. The performance aligns with our expectation that few instances of 1 star rating will negatively impact recall for 1 star ratings.