Predicting Stars based on review

Parth Desai

People love to voice their opinions on the internet whether it be on through tweeting about the weather, posting your love for yarn on a knitting forum or leaving a scathing review about a product that broke on the first day. In doing this, people are littering free data all over the internet and it never seems to stop coming. This data can be gathered and sorted through and that is exactly what will be accomplished in this project. The aim of this project is to build a classifier that is able to predict the number of stars a review will receive based solely on the text of the review. This could be used to monitor reviews and look for mistakes by users when clicking the number of stars. This could be very helpful when certain products have very low number of reviews and every rating has a significant impact on the average a user sees when shopping.

I. Information Retrieval

*A. Choosing a Source*

When choosing a source for the reviews, many things were taken into account including accessibility of data, number of reviews, and reputability of the company and products. I started off by picking some reliable retailers that had significant traffic in hopes of them having products with a large number of reviews. Home Depot[1], Amazon[2], Best Buy[3], Target[4] and Walmart[5]. When looking at Home Depot’s and Target’s websites, you have to manually click the load more button and it only does 8-10 at a time, I was not sure how to automate this, so I moved on from these websites. Looking at amazon, each review had a data hook of review-bod and there were a set number of reviews per page and the URL incremented by 1 giving an easy loop. Unfortunately, I kept getting an error that I did not have authorization to access. Best Buy had similar formatting in the webpage, however it also gave an unable to access error. That left Walmart’s website that had the perfect formatting and gave me no errors when trying to access. Next, I picked a few products to pull reviews from. The products were chosen for having a good number of reviews that would be from a wide variety of people. I chose four products, the Apple Airpods Gen 1[6], a Revlon Hair Dryer Brush[7], an onn 720p Roku Smart TV[8], and the Shark autonomous Vacuum[9].

*B. Data Gathering*

The structure of Walmart’s website is simple in terms of its organization of reviews. Once page 2 is reached of the reviews for each product, the next page button increments the page variable up by 1 for a GET method. Then a number of pages to fit all of the reviews with 20 on each page. There is a div named review-body that contained the text review and an item prop named ratingValue that gave me the number of stars a review received. Starting with the Airpods, I then looped through the first 100 pages 2 times, first grabbing the texts a saving them to an Airpod review array, and then grabbing the rating and saving it to an Airpod ratings array. I had a mismatch in my arrays of 161 because I was taking in reviews that had no text. Walmart orders all of the reviews with no content and just at the start and leaves the ones with no reviews and just a rating for last. I then search the website and manually found that the last page with written reviews was 92. I ran the loops again and still had a mismatch in my array sizes, but by only 1 this time. This required a careful combing of the data to find the exact point of the review that for some reason contained no text yet gave a rating. The oddity was a review that was a picture of the air pods as the body of the review with no words. So, then I ran my retrieval again with the range 2-92 excluding 45, yielding 1760 written reviews and 1760 star ratings. I then double checked to see that the first and last 10 reviews and stars of my arrays matched the website. I then repeated the process for the three other products. The hair dryer brush and vacuum had had equal arrays from pages 2-62 and 2-27 respectively. The TV had 3 missing values from the reviews, which were again just pictures, and those were on pages 4,5 and 41, at the time of recording this data. From here, not much cleaning was needed as the text came with no extra containers or symbols, and the rating value was a number 1-5 instead of text staying ‘one out of five stars’ like amazon. I then ran all of the web scrapes for each product in the ranges of pages that did not give missing values and saved all of the reviews to an array and all of the rating to an array.

II. Data Analysis and Preprocessing

Now I had 4800 reviews and it was time to take a look at our data. A boxplot was created in *Figure 1* with the character counts in a review. In *Figure 3*, a bar graph was created suing the distribution of star ratings among the reviews.

*A. Characters per Review*

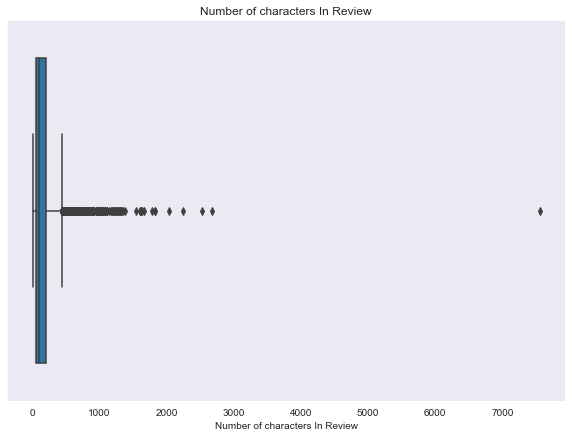


Figure 1

In *Figure 1*, we can see we have an outlier that brings down that creates a very smushed plot. From this I see I need to shave some of the reviews off the outliers and have more consistent data. I removed all reviews containing 450 or more character. I then began looking into the very short reviews and decided they were not helpful as they only contained 3 or four words out of the thousands in the document term matrix we will create later. I then shaved off any reviews that had less than 40 characters. The new dataset was called fixed. The resulting boxplot can be found in *Figure 3*.

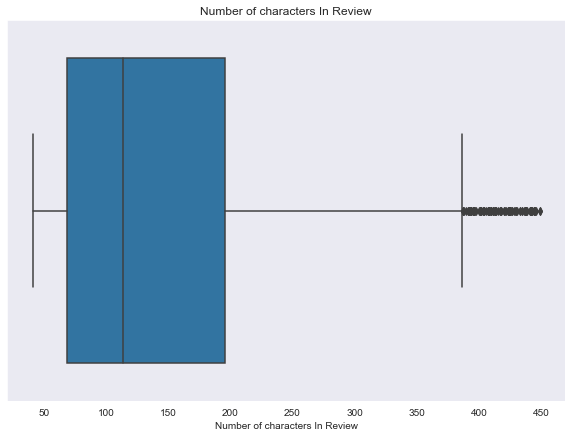


Figure 2

Here we can get a better look at the details and see the bulk of our reviews are between 70-210 characters.

*B. Reviews by Stars*

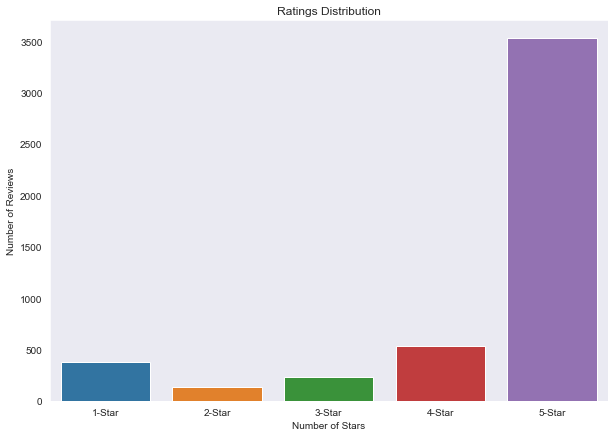


Figure 3

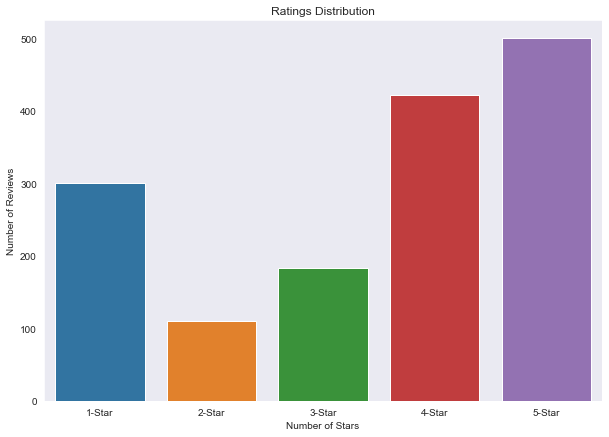
Next, looking at *Figure 3*, we can see that there are more 5-star ratings than all others combined. While this is in line with real life, this is concerning for a predictor because it can the model to guess exclusively 5-star and still get a fairly high accuracy rating but will have a horrible recall rate and will not be of use. Due to this fact, I created a new set called less with all of the reviews the same, expect I removed all but 500 5-star reviews to make the distribution a bit more even. The distribution of less can be seen in *Figure 4.* 

Figure 4

This is a much more even group of data, but will it help the predictor, or will it confuse the model as the real world has the left tail distribution. We can also see a very small number of 2 and 3-star reviews. Our model might have trouble differentiating between those classes and others due to this. We will look into both datasets and see how they compare.

The two datasets were then split into their own x and y test and train groups with the test size being 30%. The xtrain and xtest splits were then vectorized with Count Vectorizer form sklearn to get the document term matrices. The reason we split before we vectorize is so our model isn’t guaranteed to have seen all of the words like in the real-world.

III. Model Training and Evaluation

In this section we will try out many different models for our data including Naïve Bayes, Logistic Regression, Linear Discriminant Analysis, and K-Nearest Neighbors.

*A. Naïve Bayes*

I ran a multinomial and gaussian nb to see which did better. Using the fixed dataset training multinomial Naïve Bayes with fivefold validation, an accuracy score of 74.389% and the gaussian model got a mean accuracy of 45.473%. This makes sense as we are working with discrete word counts opposed to continuous variables

To see how the different datasets compared using a multinomial nb classifier, I trained and tested them, first with fixed, then with less.

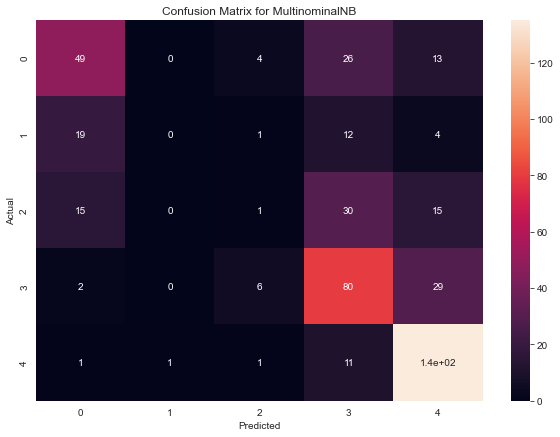
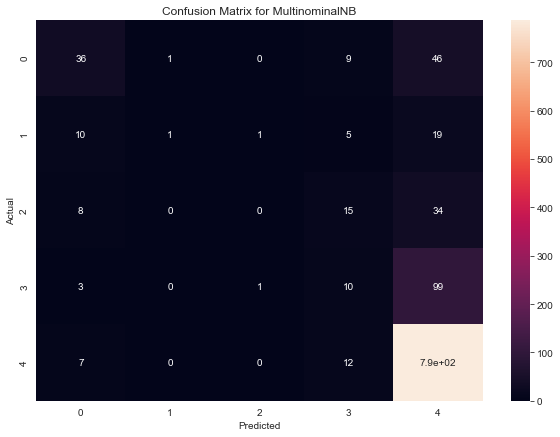


Figure 5 Figure 6

Taking a look at the confusion matrices, in figure 5, the fixed model was predicting a lot of 5s like we expected and the less model in figure 6 shows that again we are predicting a lot of 5s, but we are also seeing many more 4s were predicted. About the same number of one’s twos and threes were predicted. Looking at the F1 scores, fixed had a weighted average of .69 while less had a .52. The fixed has a higher one because the while the precision and recall increased for 4-star guesses with the less set, the precision and recall for 5-star guesses went down and since the 5-stars make up a larger portion of fixed, it was rewarded more in the average weighted F1.

*B. Logistic Regression*

To find the best model I ran a Grid Search Cross Validation on the Logistic Regressor. This created a model from every combination of parameters I give it and runs cross fold validation. For the solver used, I gave the options of newton-cg, lbfgs, liblinear, sag, and saga. The class weights could be balanced or have none and the multi-class was either ovr or auto. A few of these combinations do not work together fundamentally. Running this for both datasets, we see,

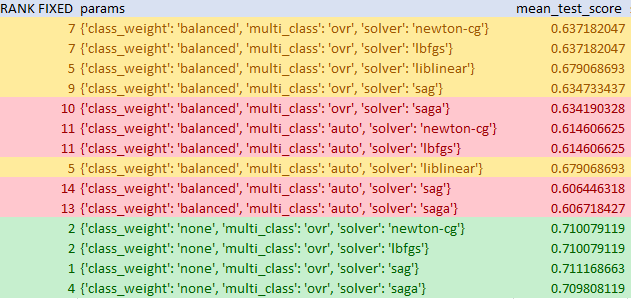


Table 1

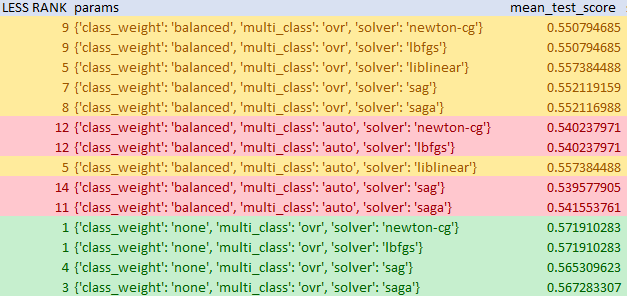


Table 2

Looking at the both tables, we can see that both times, an ovr multi class parameter and no class weights yielded the best accuracy with the sag solver performing best for fixed and the lbfgs solver best for less, but not by much in either case. The significant thing we can see from here is that the test score for fixed was around 14% higher. This still can possibly be due to the distribution. Training the models for fixed and less with their optimal parameters and printing a classification report, we see that the macro average is .34 for fixed and .46 for less. This shows us the distributions impact. The less set was again better at predicting non 5-stars, but since most reviews are for 5-stars, the fixed set created a model that appears to perform better.

*C. Linear Discriminate Analysis*

I trained my LDA model with the two sets of data with a svd solver and got less than promising results. The accuracy score of .38 for fixed and .37 for less. The difference on a per star basis was consistent as well across the two datasets including the fact that it was a poor estimator of stars. A bright spot for the classifier is that it did better at predicting 2- and 3-star reviews than any other.

*D. K-Nearest Neighbors*

I trained a KNN model with the two sets of data for an odd number of neighbors 1-37 and found that 11 was the best accuracy for fixed and 7 for less. Then training the models with the two sets at their respective neighbor number, we get *Figure 7* and *Figure 8*.

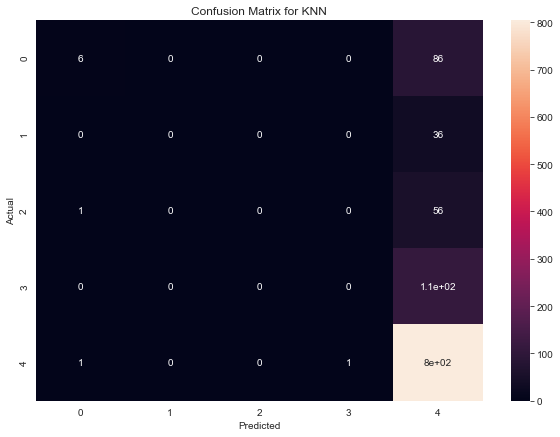
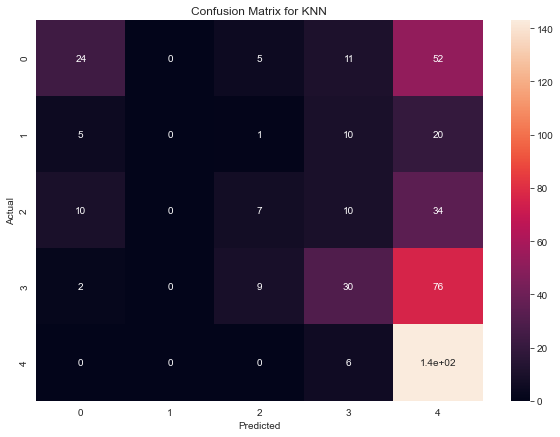
 

Figure 7 Figure 8

*Figure 7* shows the model trained with the fixed dataset and we can see that it predicteed 5-stars everytime except for 3 occasions. In figure 8, we can see the KNN model doing a better job of not picking 5-stars everytime.

IV. Ensemble Classifiers

Now that I have my base predictors and their parameters, I would like to create a few ensembles

*A. Voting Classifier*

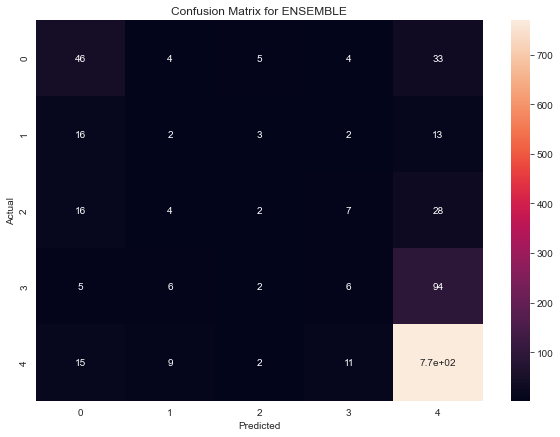
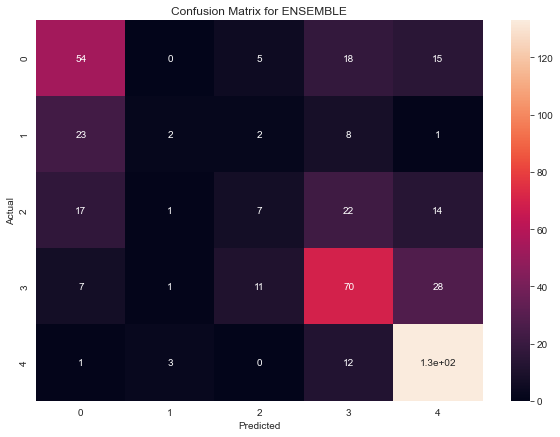
Starting off, we have a voting classifier with the MultinomialNB, LogReg, and LDA with the fixed, and then less datasets. Choosing the first two due to their optimal performance and the LDA to hopefully provide a voice to the middle values. I did not include all of the predictors too have an odd number  

Figure 9 Figure 10

Looking at these, we can see that the fixed in *Figure 9* looks very similar to our previous fixed and the less in *Figure 10* looks very similar to many previous. Fixed is good at guessing 1-star and great at 5-star while Less is less accurate but does a better job in 1-star and 4-star reviews. The lack of 2-star and 3-star reviews in the data hurts our ability to predict these, but since these aren’t seen very often. It’s a trade-off. We see our less dataset produces less false positives on the guessing 5-stars when it is 1-star and vie versa which we like to see. Guessing a 2-star on a 1-star or a 4-star on a 5-star is not nearly as much of an issue. The less dataset had an accuracy of 58% which is better than any one classifier with the less dataset.

*B. Bagging Classifier*

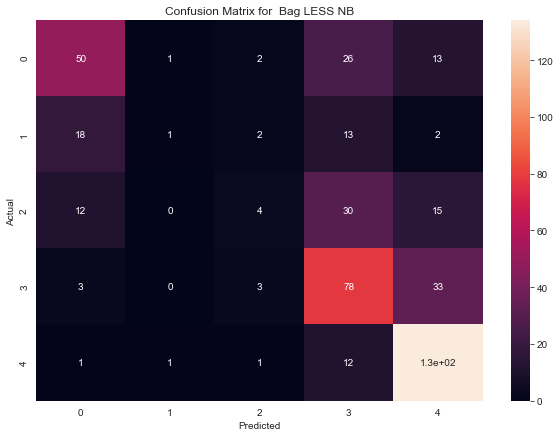
Training a bagging classifier with a base classifier of Naïve Bayes with both datasets gives us no increase to our fixed original Naïve Bayes model, however we do get a nice 5% increase to our accuracy using this ensemble. 

Figure 11

*C. Boosting Classifier*

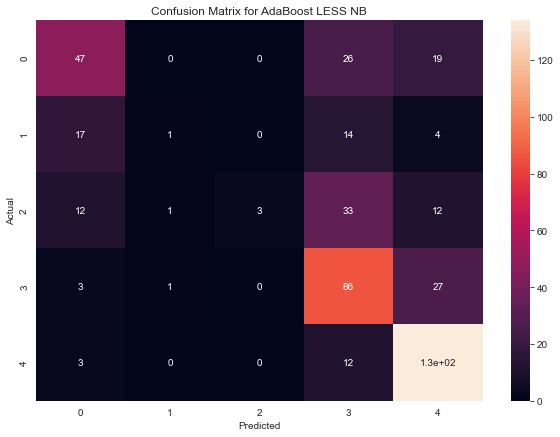
Using an Adaboost Classifier with a base estimator of Naïve Bayes and using the less dataset we get an accuracy of 60%, which is an increase of 6% from before. Again, the fixed dataset did not produce any nontrivial change. 

Figure 12

V. Conclusions

*A. Regarding the Goal*

After looking at all of our options, it appears that the best predictor for our goal is the voting classifier using LDA, Multinomial Naïve Bayes, and Logistic Regression that is trained with the less dataset with a more even distribution of stars than fixed, which was very skewed towards 5-stars. It has the 3rd highest accuracy out of the ensembles using this dataset, however it produced the least large mistakes, meaning that the guesses that were off by 3 or 4 stars were minimized in this model. This model also classified 1-star reviews correctly the most often. While the fixed dataset produced more accurate models, they were riddled in mistakes due to the bias towards 5-star labeling in many cases labeling more 1-star reviews as 5 than 1.

*B. Regarding the Experience*

In the future, I’d like to try a couple of new things with the data. For one, I’d like to see how this model would perform on reviews of products that aren’t in my training data. I tried to pick some basic household items in my scrape to get general reviews in hopes of getting a solid model. It would be interesting to see how my trained model would predict reviews for a different TV or maybe an article of clothing. I’d assume it would do better on the former rather than the latter. I also would like to explore making predictors for specific subset of products, like one for just headphones. It would be hard to cover all types of products, but it would be much more accurate for the intended items. In this project I had to deal with highly unbalanced data, but I was more prepared with steps I could take to mitigate those. I would have like to try a stack ensemble because it seems easier to differentiate between a positive and negative review. Maybe one learner could separate them into two groups and then use a secondary learner to further narrow down which specific number the positive or negative review is. Overall, it was a good learning experience on working with organized text as training data.

APPENDIX A

The Github Repo with my files and code can be found at <https://github.com/ParfDesai/CSI5810Project2>

REFERNCES

[1] <https://www.homedepot.com/>

[2] <https://www.amazon.com/>

[3] <https://www.bestbuy.com/>

[4] <https://www.target.com/>

[5] <https://www.walmart.com/>

[6] <https://www.walmart.com/reviews/product/604342441>

[7] <https://www.walmart.com/reviews/product/734692345>

[8] <https://www.walmart.com/reviews/product/314022535>

[9] <https://www.walmart.com/reviews/product/506507232>