Melissa Zhen
Github user: melzhen
11/07/2022

Final Report

I. Preliminary analysis / exploration:

I explored several features and checked if they had any correlation to the rating they were given using the training set data. The features I checked were ‘Helpfulness’, ‘num_reviews’, ‘popularity’, ‘Number of Words’, ‘Sentiment_Score’, and ‘Time’.

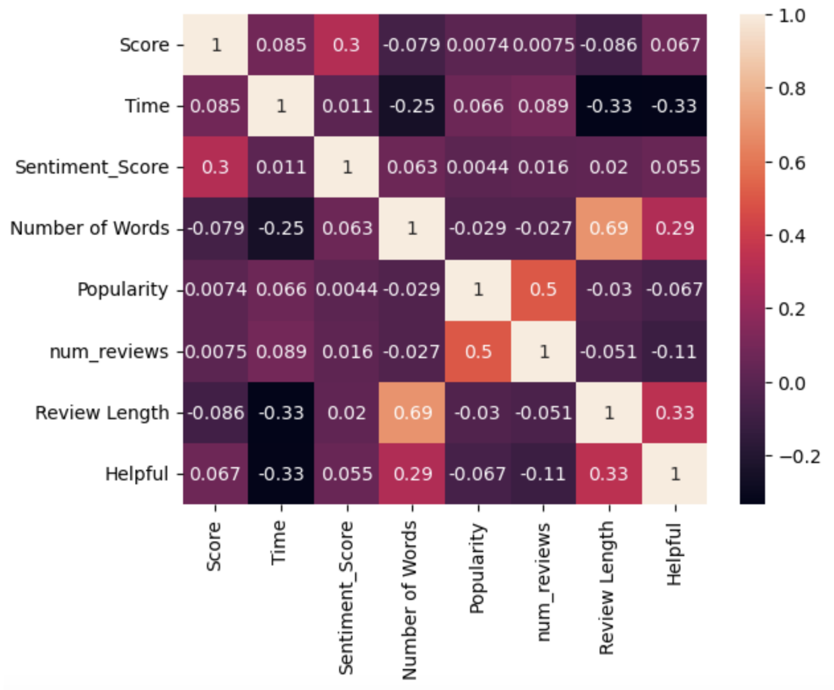
The reasons I decided to look at/create these features is as follows:

- Helpfulness: Taken from the professor’s starter code. This gives the number of likes / number of likes + dislikes.
- num_reviews/Popularity: I used value_counts() and sorted the products in descending order of the number of reviews they had. I then used Kmeans to split them into 5 clusters and ranked them by popularity on a scale from 1-5. I did this because I assumed that more popular products would have higher ratings.
- Number of words: I tried exploring if the length (number of words) of the column ‘Text’ had any correlation to the score rating. I assumed that the longer a review was, the higher the rating would be.
- Sentiment_Score: I used NLTK’s opinion lexicon dictionary to get an estimate of the ‘Sentiment_score’ by counting the ratio of positive words to negative words. This sentiment_score ranged from -1 to 1.

II. Feature extraction

In order to decide which features to use, I first tried plotting a correlation matrix to see which features had a positive correlation with the target ‘Score’. I decided to remove all the features that had negative correlation. The features that had a positive correlation were ‘Helpful’, ‘Review_Length’, ‘num_reviews’, ‘popularity’, ‘Sentiment_Score’, and ‘Time’. However, I ended up only using ‘Helpfulness’ and ‘Number of words’ out of these features because I ultimately decided to use CountVectorizer (bag of words) as a tool for my method and I did not want too many numbers involved. So, I chose two numerical features that seemed most relevant.

The correlation matrix I looked at:



	Score	Time	Sentiment_Score	Number of Words	Popularity	num_reviews	Review Length	Helpful
Score	1	0.085	0.3	-0.079	0.0074	0.0075	-0.086	0.067
Time	0.085	1	0.011	-0.25	0.066	0.089	-0.33	-0.33
Sentiment_Score	0.3	0.011	1	0.063	0.0044	0.016	0.02	0.055
Number of Words	-0.079	-0.25	0.063	1	-0.029	-0.027	0.69	0.29
Popularity	0.0074	0.066	0.0044	-0.029	1	0.5	-0.03	-0.067
num_reviews	0.0075	0.089	0.016	-0.027	0.5	1	-0.051	-0.11
Review Length	-0.086	-0.33	0.02	0.69	-0.03	-0.051	1	0.33
Helpful	0.067	-0.33	0.055	0.29	-0.067	-0.11	0.33	1

III. Workflow, decisions, and techniques tried

I decided to handle all the missing values by using the ‘forward fill’ method. I then did pre-processing on the ‘Summary’ and ‘Text’ columns to convert them into lowercase, to remove non alphanumeric words, to get rid of any punctuation, and to remove stopwords. I also tried stemming the words using Snowball Stemmer, but got a higher RMSE score as a result. Next, I combined all the features I chose into one column so that I could feed the data into CountVectorizer() to featurize it. I also tried using Tf-idf Vectorizer instead of CountVectorizer but got a higher RMSE as a result, which was not what I wanted. I experimented with training a Multinomial Naive Bayes classifier and got a RMSE of 1.03 (further analyzed later). To see if I could decrease the RMSE a bit more, I tried training a Logistic Regression model to compare their performances. In the end, the Logistic Regression classifier outperformed the MNB classifier for the training dataset.

IV. Model tuning/testing

For CountVectorizer, I initialized it so that it would remove all stopwords in English, strip all the accents, lowercase the letters (if not already), and have a max_df of 0.92 (explained later why I chose this). As stated above, I tried training a multinomial naive bayes model as well as a logistic regression model to use on my testing set. The results for each of the tests are below:

- Multinomial naive bayes
  - Accuracy on testing set = 0.6113903326998374
  - RMSE on testing set = 1.0342753860328338

- Logistic regression
    - Accuracy on testing set = 0.6475253589174089
    - RMSE on testing set = 0.9203054285591599

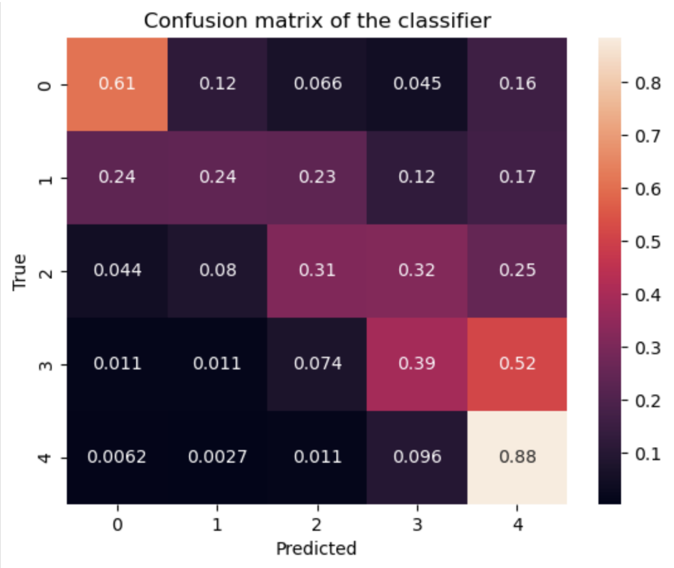
  I received a ConvergenceWarning so I fixed some parameters according to some solutions I found online at StackOverflow.
- Logistic regression w/ fixed hyperparameters and no warning
    - Accuracy on testing set = 0.6328853067112403
    - RMSE on testing set = 1.0260639485104266

I found that the logistic regression model without the fixed parameters performed better in overall accuracy and RMSE.

V. Ideas & Challenges

I used NLTK's opinion lexicon to get an estimate of the 'Sentiment_score' but it ended up giving me a lower accuracy and higher RMSE score. Therefore, I ended up not using it for my final submission. I also suspect that this approach would not give me an accurate prediction because NLTK's opinion lexicon dictionary is limited and does not contain every 'positive' or 'negative' word. I also decided to pre-process the data more than once because I realized that even though CountVectorizer would already do some of the pre-processing, I got a lower RMSE by manually pre-processing the data beforehand as well. I tried running a test to find the most optimal max_df to use for CountVectorizer. I tried values ranging from 0.9 to 0.99 but they all gave me the same accuracy. I assume the reason for this is because there are not enough words that appear most frequently since I removed all stopwords during pre-processing. Hence, I chose a random max_df value of 0.92 to use.

Based on the confusion matrix shown below, I found that my model is most confused when predicting the target scores 2, 3, and 4.



True \ Predicted	0	1	2	3	4
0	0.61	0.12	0.066	0.045	0.16
1	0.24	0.24	0.23	0.12	0.17
2	0.044	0.08	0.31	0.32	0.25
3	0.011	0.011	0.074	0.39	0.52
4	0.0062	0.0027	0.011	0.096	0.88