Abdelalim Yakoub
Ashit Patel
Dr. Mandy Korpusik
Natural Language Processing
December 15, 2020

Amazing Reviews Rating

Introduction

For our final project in the Natural Language Processing class, we decided to make a program that predicts the rating of a customer's review. In many commercial/ merchandise websites, customers usually leave feedback reviews to product that those customers have purchased, and this is popular in websites such as Amazon, eBay, and many other websites. In most websites, the customer can also leave a rating, usually from 1 to 5, along with the text review to rate the product. However, some websites only allow a text review without a numerical rating value, which makes it hard to actually process this data for further analysis. Therefore, to actually process such data, we need to first make a program that gives us a numerical value for such reviews. To make such a program, we would have to first it train it on set of reviews that already have rating in order to be able to predict ratings for reviews that do not have ratings.

Dataset

The dataset used in this project is an amazon product data provided by a professor in UCSD, Dr. Julian McAuley. It contains millions of reviews spanning from 1996 to 2014; however, we only have access to subsets of this data. The subsets are classified into product genres, and the genre we used is the Clothing, Shoes, and Jewelry products. It contains around 300,000 reviews. and each review contains multiple data points, such as the reviewer-ID, reviewer-Name, review-Text, review-Date, overall, …etc, but we will only be using 2 of these data points for each review: the review-Text and overall (rating), where the review-Text is the X-data and overall (rating) is the Y-data.

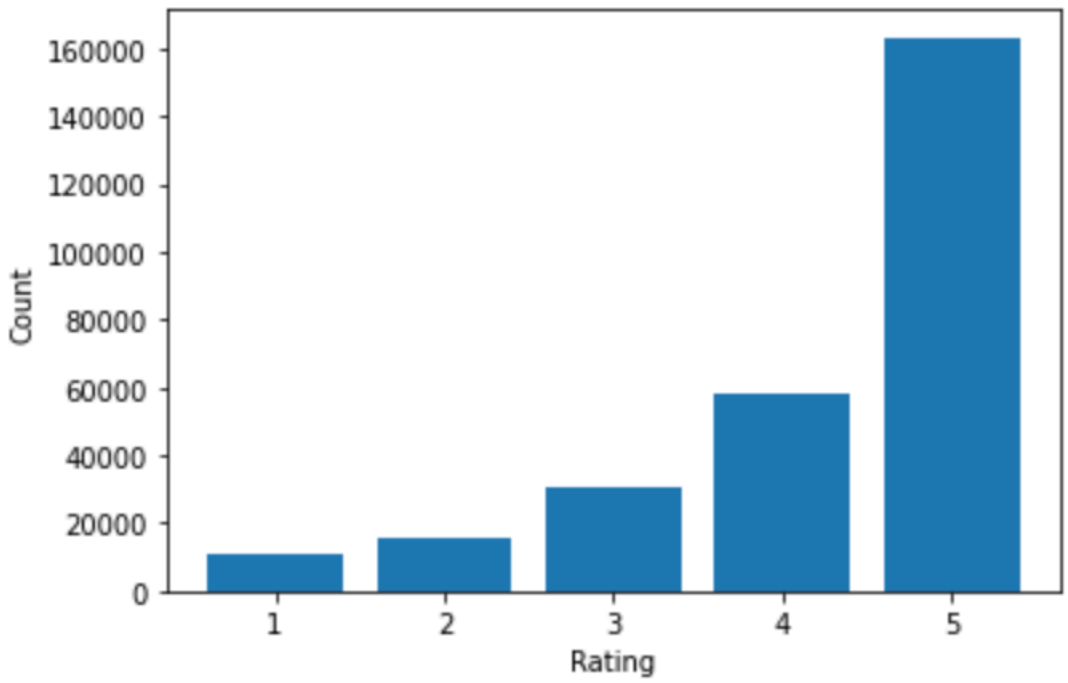

Fig. 1 Ratings Bar Graph

This figure above shows the distribution of the ratings in our dataset, and we can clearly see that most labels are 5-STAR ratings. Consequently, we might end up with a bias in the results.

Before we start using the dataset, we split it into 3 subsets: a training set, a validation set, and a testing set. The training set is the the set used to train the model, the validation set is the set used to evaluate the model's behavior after every epoch during training, and the test set will be used to test the model after training is done. The number of reviews in the training set, validation set, and testing set are around 243,000; 27,000; and 30,000 respectively.

Dataset Preprocessing

Before processing our dataset and actually using it to train our model, we had to first do a few changes to it. The main change to the dataset was to tokenize the data because the model would not be able to actually process the words. Instead, each unique word in the dataset would be assigned a token (a number), and since the words are now replaced with numbers, then the model would be able to process and train on

such data. To do the tokenization process, we used the BERT tokenizer, which is a function available on an open source Library, and it also adds two extra tokens to each sentence: a token at the beginning of every sentence, and a token at the end of every sentence.

The model also requires all token (sentences) sequences be of the same length, and it also accepts a maximum length of 512 tokens; therefore, we had to truncate or pad certain sequences according to whether they were more than or less than 512 tokens. This also shows that we might have lost a few words that might have only appeared on the truncated parts of the sentences.

The next change we made to the dataset was done to the labels, and it is a very simple change. When declaring the BERT model, we specify the number of classes, and for this project, there are 5 classes (1, 2, 3, 4, 5). However, because we specify 5 classes for the BERT model, its output will be [0, 1, 2, 3, 4]. Therefore, all we had to do was to subtract 1 from all the labels in our dataset in order to match the output of the model.

The final step was to add attention masks to the token sequences in order to identify whether the token represents an actual word or is just a padded value.

Model

Our initial model was a simple logistic regression with all default values set. We did not even validate during training; instead we trained it on most of the data and simple ran the model on the test.

As mentioned earlier, the main model used for this project is the BERT (Bidirectional Encoder Representations) neural-network. To load the model on Python, we used the BERT for Sequence Classification function, an open source function, which is pertained BERT model with 1 classification layer on top. We used the ‘bert-base-uncased’ model which is a 12 layer model with uncased vocabulary. The model has 5 output classes, uses the Adam optimizer with 0.00002 learning rate, has a batch size of 16, and trains for 2 epochs.

Training & Validation

Logistic Regression Model

The average training accuracy for this model was 69%, and no validation was done on this model.

BERT Model

Finally, we started to train our BERT model, and as mentioned above, we only trained the model for 2 epochs, and thats because of the very long time it would take to train, which was around 4 hours per epoch.

In the first epoch, the average training loss was 0.69, the average training accuracy was 72%, and the validation accuracy was 74%. The total training time was around 3 hours and 30 mins.

In the second/last epoch, the average training loss was 0.55, the average trading accuracy was 78%, and the validation accuracy was 75%. The total training time was also around 3 hours and 30 mins.

Testing

Logistic Regression Model

After training this logistic regression model, we ran the model on the test set and the testing accuracy was 69%, very close to its own training accuracy.

Rating	Precision	Recall	F1-Score
1	0.53	0.49	0.51
2	0.36	0.15	0.21
3	0.41	0.28	0.33
4	0.50	0.26	0.34
5	0.76	0.95	0.84

BERT Model

After training the BERT model, we ran the model on the test set and the testing accuracy was 75%.

Conclusion

The BERT model did better than the Logistic Regression model; it had 10% more in training accuracy and 6% in testing accuracy; however, it was very time consuming; which was our main difficulty in this project.

For future improvements, we can try using a different neural-network, a different BERT model, or a larger dataset but with a bigger GPU.