Model Training Report

Rowan Whelan

Abstract:

The workflow I followed for this project was first experimenting on a small section of the data to find a set of features that I thought was strongly encapsulating the data. Then after I selected my features I experimented with model choice until I found what I thought was the best model I could make. Finally after I chose my model I went back and tried updating my parameters to include some new features with the hope that these features would better fit my model.

Workflow:

Features:

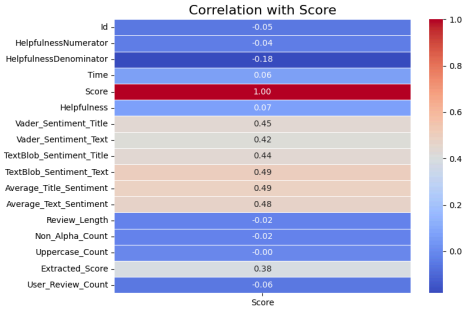
At first, I sampled the data into a very small chunk and naively ran a KNN model with the general idea that I would readily be able to reason about the features after watching a fast model's performance. After some testing I decided to use 'Helpfulness' ,'HelpfulnessDenominator', 'Average_Title_Sentiment' , 'Average_Text_Sentiment' , 'Review_Length', 'Vader_Sentiment_Title', 'Vader_Sentiment_Text', 'TextBlob_Sentiment_Title', 'TextBlob_Sentiment_Text', 'Non_Alpha_Count' as features.

The idea behind helpfulness and helpfulness denominator was that positive community engagement would likely correlate with higher scores (both values high), while a high denominator with no corresponding increase would be the result of an unhelpful review which I thought would likely be lower rated. Next, I used two text sentiment tools, the first was Vader which was mentioned in the lecture and the second was Textblob which I discovered through some research. Importantly, I started by just naively averaging the scores, but after I did some research later on in the project I discovered that Textblob is normally used on large text sets while Vader is much more suited to shorter text like social media posts. I decided that this difference, while hard to see on while looking at the entire dataset, might be novel enough on very long or very short posts that allowing the model to have access to both sets of numbers would be beneficial. Finally I used review length and non-alpha-numeric character counts because I thought that more dissatisfied customers would leave shorter less verbose reviews.

```
Accuracy per star value (including 1-star):
Score
1.0     6.568833
2.0    70.917407
3.0    56.005669
4.0    51.097687
5.0    81.908762
Name: Score_Match, dtype: float64
```

Finally at every iteration of the project I added a final variable which would act as a final processing step. I realized that some people would write out the score they were giving in the review. After the algorithm made its predictions I went back through them and updated them based on if it mentioned a score. I did this because it proved to be incrementally more accurate in every case except the case where the review mentioned a 1 star rating (See Figure above).

The first trick I used to more objectively look at my choice of features was graphing their correlation to score in a heatmap (See figure to the right). However this is the first place I feel that I made a mistake approaching the problem. I reasoned that the fact that I had 4-5 moderately correlated features to score was a reflection of the difficulty of the problem and that improving these features would be a waste of time (which was very limited at this point because of model running time and would only be solved later) I regret this because I feel like it bottlenecked my score until the last day I was able to work on the model.



Model:

Finally after I settled on what I thought at the time was the best I could do, I decided to move away from feature selection and begin training a model on the whole dataset which was more capable of capturing the data than my previous purposefully naive attempts. At first I tried to use KNN and just adapt it to the larger dataset, but with a little bit of testing my results were subpar so I decided to research models. The first model I decided to train in earnest was a gradient boosting model. These models create trees to approximate the data and iterate to correct errors the trees make. This has the advantage of focusing on harder to categorize points. So I began training with this model and used its predictions on my first official submission and ended up getting .57 accuracy on the testing data.

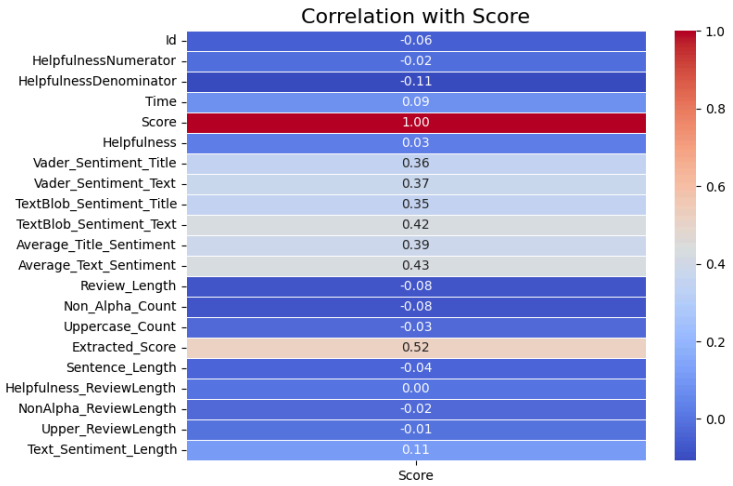
While this was a good score at the time I was unsatisfied with my progress trying to get this model to be more accurate. This was mainly because of how expensive training this model was (Approx 2 hours). So I did some research and selected three models which would dramatically improve the speed at which I could test models. I found a library which (lightgbm) implemented a version of gradient boosting called light gradient boosting. This is a much more optimized version of gradient boost which only builds trees based on the leaf it predicts to be the best. This library also allowed me to fiddle with the parameters of the input to best improve model performance. I ran a loop over these parameters and outputted them to the log. I then refined my search to only gbdt (gradient boosting) because rf (random forest) ended up being 3 percent less accurate and only changed its accuracy score very minutely. This approach ended up costing me most of my second to last day to work, but did produce the model training parameters used for my best submissions. I also experimented with inputting a class weights matrix to warn the model about the oversampling of 5 star reviews present in the training data. However, when I implemented this it decreased the accuracy of my model. This was likely because the model was predicting the less extreme (2,3,4) star counts more often and getting them wrong more often. One shortcoming of my final model is that it very accurately predicts extreme star values and gets a majority of the middling scores wrong as 5 star reviews. Finally

```
param_grid = {
    'boosting_type': ['gbdt', 'rf'],
    'n_estimators': [200, 250, 300, 350, 400, 450],
    'learning_rate': [0.05, 0.1, .15, 0.2],
    'max_depth': [5]}        You, 8 hours ago • mode
```

as part of my experimentation I discovered that lgbm allows users to input a bagging frequency and percent value. This had the model run 5 times (each with 80% of the data) and select the best of the generated models.

New Features:

Finally I spent my last day working trying to improve my features. I added features which were a combination of other features hoping to more effectively grasp what I had intuitively decided to be good variables to correlate. I added four new features: Helpfulness * Length, Non Alphanumeric Characters * Length, Uppercase Characters * Length, and Sentiment * Length. The idea behind each of these would be that allowing the model to see the interactions between them would help increase performance and also allow me to get rid of length (a not very correlated value) while keeping the information it contained. This did slightly work, Text Sentiment * Length was more correlated to score than length. However due to the limited time I had I wasn't able to effectively experiment with these combinations.



Conclusion:

In conclusion, my approach and understanding of the concept was sound enough to get me an average score, but the short nature of this exercise and my poor time management stopped me from reaching a higher score. Given time I would have wanted to experiment with a more robust feature set and reimplementing a class weight matrix which could help the model more accurately model the data. I am happy with the way I chose to approach the problem. I think starting with a smaller sampled set allowed me to get very comfortable with managing the data and the model.

Sources:
https://lightgbm.readthedocs.io/en/stable/