Kevin Smith
CS506
November 8, 2023

MIDTERM PROJECT REPORT:
MOVIE REVIEW PREDICTIONS

TASK: Given a table of Movie Reviews with attributes {*ProductId, UserId, HelpfulnessNumerator, HelpfulnessDenominator, Time, Summary, Text*, *Score*}, predict the *Score* attribute of an unseen table of Reviews.

EXPLORATORY ANALYSIS:

I began by generating some key statistics (mostly related to the other attributes in the dataset) and examining their relationship to the final score given. Those key statistics:

- *Count of each score*
- *Mean HelpfulnessNumerator per score*
- *Mean HelpfulnessDenominator per score*
- *Mean Helpfulness (calculated as HNumerator/HDenominator) per score*
- *Time distribution (in hour-long blocks) per score*

This preliminary analysis is contained in the **Exploration** section in the Jupyter Notebook. From the graphs shown, I came to the following conclusions:

- In general, scores are more likely to be high than low
- The mean Helpfulness is not a very useful statistic as all five scores tend to have a mean helpfulness score around 0.4, give or take 0.05. However, the mean Helpfulness Denominator is much higher for 1-star reviews than any other, and higher for 2-star reviews than 3-, 4-, and 5-star reviews. Therefore, I hoped to include this as an important factor in my model.
- Time distribution, while interesting, seems fairly constant across all reviews (with more reviews being made in the evening and late at night while less reviews are made in the morning and early afternoon). Therefore I won't use the timestamp as a feature in the model.

- The mean review length tends to be a bit longer for 2-, 3-, and 4-star reviews than for 1- and 5-star reviews. I tried to take this into account in my model but wasn't super successful in making that differentiation.

FEATURE SELECTION:

From there, I wanted a good way to analyze the actual text of the review, since I believe intuitively that the words used by the reviewer will eventually have the greatest impact on the final score over all these other numerical values. For that, I did some research and found the VADER Sentiment Analysis engine. It seems to supply a passable positivity/negativity rating system which I hoped to use to transform the review Text attributes into numerical scores. **I tried a few iterations of Vader Sentiment Analysis weighing the scores of the Summary and Text columns differently,** but the lowest RMSE I got from those trials was from just using the Text column's analysis, so in my final model, the VADER of the Summary column is not taken into account.

I also calculated the average review by each unique user and added that to the rows, as well as including the review length since 2-, 3-, and 4-star reviews tended to be longer than 1- and 5-star reviews. Finally, I added a column calculating the Coleman-Liau Complexity Index for each review, which takes into account a text's average word length as well as its average sentence length.

I also wanted to try a TF-IDF vectorization since it was recommended in office hours. I used this vectorization approach on the Text column of the training set without specifying a max number of features, which ended up causing problems when trying to create the final CSV submission (lots of errors) but I finally got it working after attending another office hours. I combined the vectorized columns with the other numerical columns that I decided to include in the model (HelpfulnessNumerator, HelpfulnessDenominator, ReviewLength, PositivityScore, AvgScoreByUser, and the Complexity Index) to get down to my lowest RMSE yet, 1.021.

By this time I was running out of time, but I tried one last thing: I researched the Surprise scikit which I thought might help me in conjunction with the PositivityScore from VADER, but I wasn't able to get the library working on my dataset before the submission deadline so my final notebook submission does not include that approach.

THE MODEL ITSELF:

I used the CatBoostClassifier, I found from multiple sources that regression algorithms tend to have better performance on classification than something like KNN. It took me a couple hours to even get the Classifier running, but once I did I tuned its parameters by testing various values — I ultimately found that 100 iterations was a good balance between performance and runtime, and MultiClass worked best for the loss function (intuitive). I also tried to implement scaling & normalization as recommended, but both the StandardScaler and MinMaxScaler from sklearn made my RMSE notably worse when included in the model.

PROJECT TAKEAWAYS:

I've been hearing the old adage "all models are wrong, but some are useful" since 11th-grade physics class, but I think this was my first and certainly most impactful hands-on experience with that idea. It's surprisingly frustrating to spend so much time implementing the nitty-gritty aspects of a model such as complicated feature extraction just for the model to perform even worse than before (which happened many, many times). What was especially interesting to me (and somewhat expected) was how the model generally was able to distinguish fairly easily between 1- and 5-star reviews, but tended to be fairly optimistic when classifying 2-, 3-, and 4-star reviews — it predicted many 4-star reviews correctly but almost no 2-star reviews correctly. I believe given more time I would try to implement sub-processing steps to identify certain aspects of 2-star language that differentiate those reviews from the qualifying features of a 4-star review as opposed to a 5-star review, as I imagine there is a lot of overlap there.

My algorithm- and OOP-centered experience with CS thus far kept pulling me towards a temptation to just write a long algorithm using high-dimensional matrices and dictionaries and the like to manually predict a score for each row, but I had to actively stop myself from trying to do so given the infeasibility of writing such an algorithm, much less one that would be able to predict take in data from 140,000 rows in any short amount of time. I think over everything this project just reminded me that if I want to successfully enter the CS workforce, especially in a data-focused role, I'd better spend a little less time on the foundational, theoretical aspects of CS and more on learning the most common libraries out there and how to use them to their full power, since they're more efficient and accurate when dealing with high quantities of data than any algorithm I could write from scratch (which tends to be more fun for me).