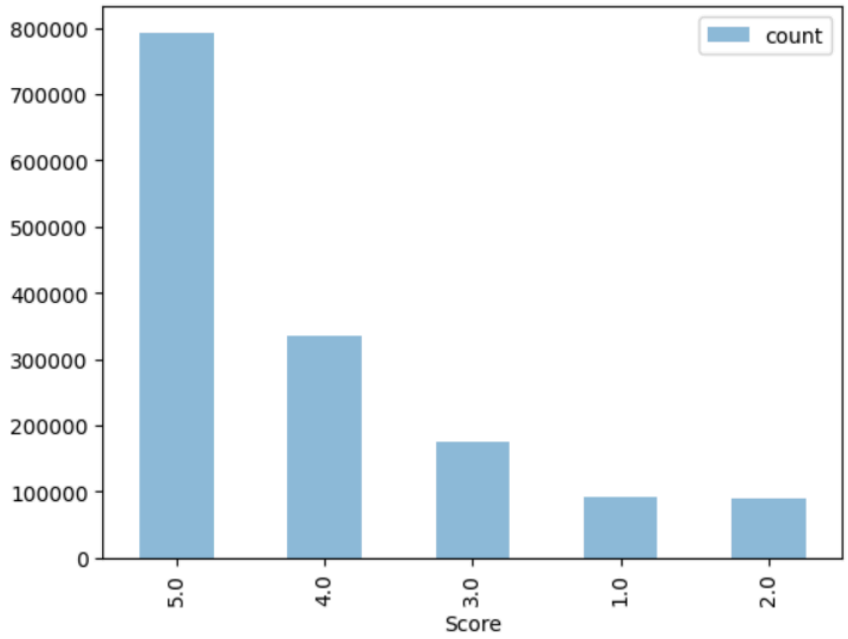
Jason Zhu
CS 506 Midterm Writeup

In this report, I outline my process for designing, engineering features, and optimizing an algorithm to predict Amazon movie review ratings. I aimed for a solution that balances interpretability with high classification performance. The majority of the project was focused on feature selection and tuning the model. The data used in this project came from Amazon's movie review dataset, and my goal was to predict review scores using several carefully derived features.

Data Preprocessing

I began by loading data to examine the dataset's structure and class distribution. A quick look at the score distributions revealed class imbalances, so I planned to address this with weighting or balancing techniques during model training. The distribution was highly skewed, with far more 5 star reviews than the other ratings. Filling missing values and extracting relevant text and temporal features were also essential first steps that I did next.



Feature Selection

When making an initial judgment on the data, some features that I assumed would be highly correlated with the score of the review were user mean score, product mean score, review word count, and sentiment analysis of the reviews. I believed that these would correlate with the score given on the review because user mean score and product mean score would give us an idea of the kind of ratings that the user usually gives, or an idea of the kind of ratings other users have given the product. I thought review word count would correlate with score because reviews

that are long may be extreme in the positive or the negative. Similarly, sentiment analysis of the reviews would probably show that positive polarity of the review correlates with a higher score and a negative review correlates with a lower score.

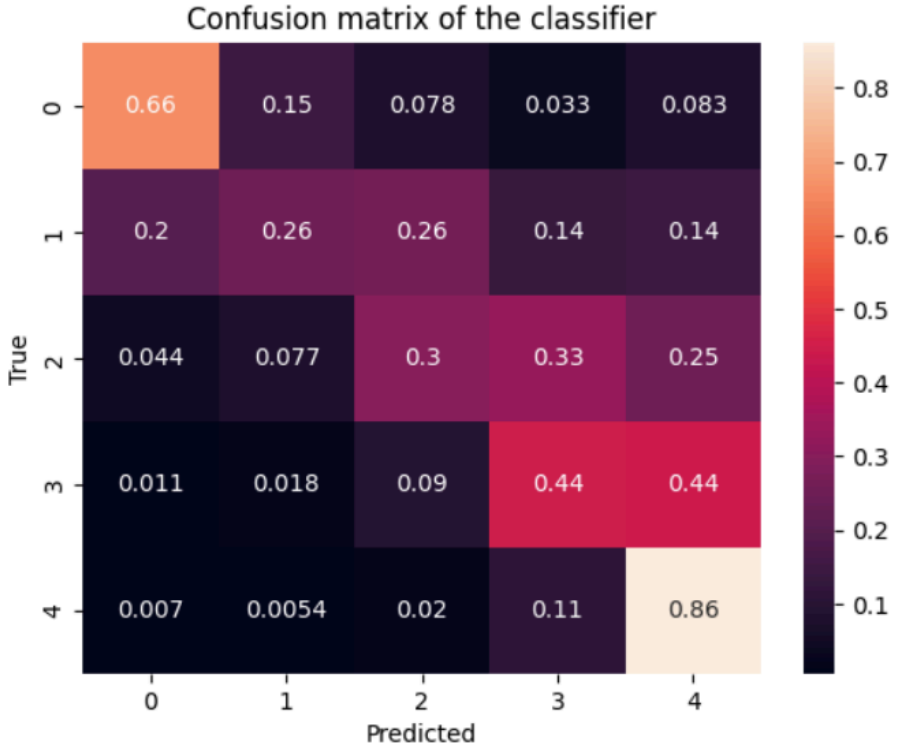
This was in fact the case as all these features were somewhat correlated with the score of the review. I found that these features were more accurate when predicting a review that had either a 1 star or 5 star review, the extremes. It was difficult to predict when a review was given a 2,3, or 4. Thus, I added many more features that, although were not all highly correlated with score, I figured it would help in distinguishing between reviews that were less extreme. I used the HelpfulnessNumerator and HelpfulnessDenominator, as well as IsHelpful (if the helpfulness numerator was greater than 0), HelpfulnessDifference (helpfulness numerator - helpfulness denominator), and WeightedHelpfulness. These additional features regarding helpfulness helped to indicate whether any users found the review helpful at all, captured reviews that might have received more unhelpful votes than helpful ones and distinguished newer or less popular reviews from heavily voted reviews, providing better insight into potential quality without being skewed by low-vote counts.

UserAvgScore and ProductAvgScore, as well as UserMedianScore and UserReviewTotal helped shed light on user and product patterns. There were missing values and I replaced them with global averages. There were not many missing values and replacing them with global averages avoided skews and did not impact the data much. Surprisingly, I found that time was slightly correlated with score, as there was a small positive correlation between time and score. Therefore, I converted the time to year, month, day of the week, and quarter and added them to my model. To analyze the content of the reviews, I extracted textual features such as the review word count and the count of unique words. These metrics implied that longer reviews with varied vocabulary could offer more nuanced opinions. After exploring various sentiment analysis tools, I selected Python's TextBlob library for assessing sentence subjectivity and polarity. I did not have any experience with natural language processing but I found TextBlob simple to learn and apply in handling natural language tasks. I calculated polarity scores that reflect the emotional tone of the reviews and added this feature to my model.

Modeling

Initially, I experimented with the KNN model to predict the scores, however, I ran into problems due to the size of the dataset. The model took way too long to complete each run, which made it difficult to test multiple iterations. Thus, I chose to use Gradient Boosting Classifier for its ability to handle complex datasets. Although it still took about an hour for each run, it was still a major improvement. In cases with imbalanced classes (like the overrepresentation of 5-star ratings), Gradient Boosting Classifier re-weights the samples after each iteration, giving more focus to samples misclassified by previous trees. This way, GBC can better distinguish minority classes and avoid bias toward the majority class without requiring explicit re-sampling or re-weighting adjustments. During each iteration of my modeling, I

adjusted the parameters to optimize the model. I initially started with 300 n estimators to maximize accuracy but it increased training times and may have caused overfitting. I decreased it to 150 as a balance between performance and efficiency. I also adjusted learning rates, initially starting with 0.1. A lower learning rate allowed for better convergence but I would need to increase the number of estimators again, so I settled for 0.2 learning rate. I found these parameters to be sufficient in creating this model.



This confusion matrix shows the performance of the classifier in predicting the star ratings. The classifier tends to struggle more with middle-range ratings (scores 2 to 4) and is more accurate with extreme ratings, especially 5. This is common in rating prediction models, as middle ratings often have overlapping features that make them harder to distinguish. The model performed decently as I got an accuracy of 0.65074 on the testing set and an accuracy of 0.60956. The combination of the selected features contributed to creating this model.

Challenges and Next Steps

A challenge that I faced was the long runtimes. Efficiency was crucial to this project as we needed to run multiple iterations in order to improve our model. Not to mention, adding more and more features caused the runtime to increase as well. Each iteration would take hours which greatly slowed down the process of testing and refining. Towards this challenge, I could leverage distributed systems or parallel processing (e.g., with Dask, Spark, or a cloud-based environment) could speed up computations, especially for hyperparameter tuning or running multiple experiments simultaneously.

References

GradientBoostingClassifier. (n.d.). Scikit-Learn. Retrieved October 28, 2024, from https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.

GradientBoostingClassif ier.html TextBlob: Simplified Text Processing — TextBlob 0.18.0.post0 documentation. (n.d.). Retrieved October 28, 2024, from https://textblob.readthedocs.io/en/dev/