Amazon Movie Review Gradient Boosting Model

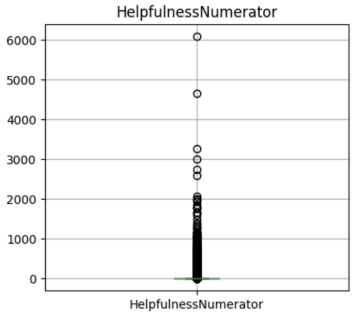
This report presents a predictive model designed to determine the star rating of Amazon movie reviews. Gradient Boosting emerged as the optimal choice due to its, efficiency, and ability to capture complex patterns. This document details the steps taken in data exploration, feature engineering, model selection, tuning, and evaluation, with insights on the impact of each decision on the model's performance.

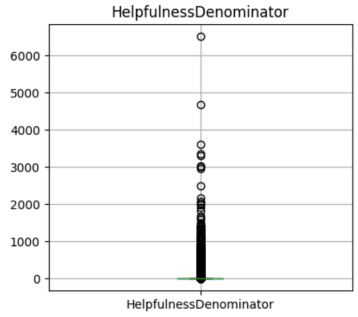
Exploratory Data Analysis

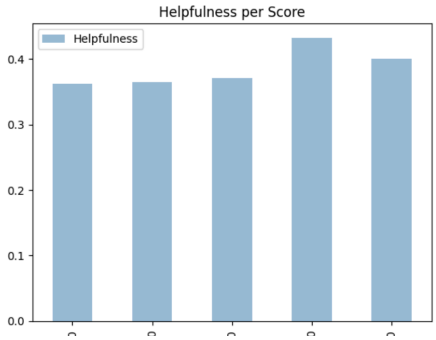
It was key to the choosing of a model to firstly analyze and explore the features. The features in the dataset include a unique identifier for the user and product, the number of users who found the review helpful, the number of users who indicated whether they found the review helpful, the timestamp for the review, the brief summary of the review and the text of the review.

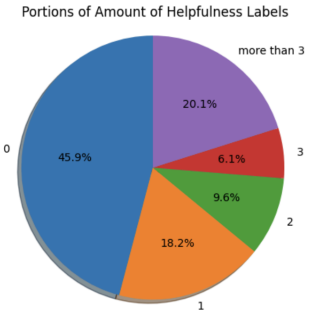
In thinking about the concept of movie reviews in my day to day two things came to mind. The better score I give a movie the more effort I put into the review, meaning there might be a correlation between length of review/helpfulness of review and score. Also, I tend to either love a movie or hate it which could be reflected here by class imbalances.

To reflect this, I chose to process helpfulness numerator and denominator first, by creating a box-plot to determine outliers, a pie chart and distribution chart to understand the range of the ratio between the two, and also a bar chart that correlated it to review scores.





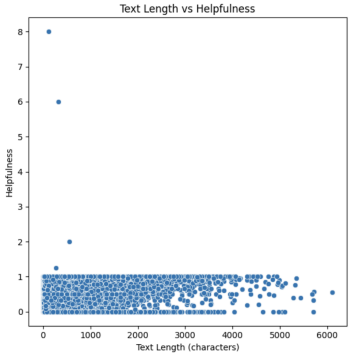


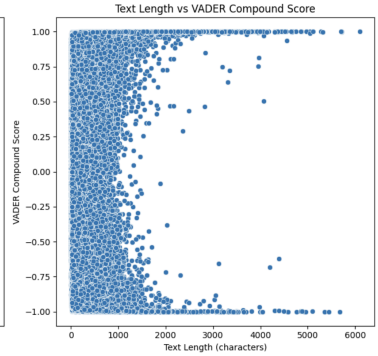


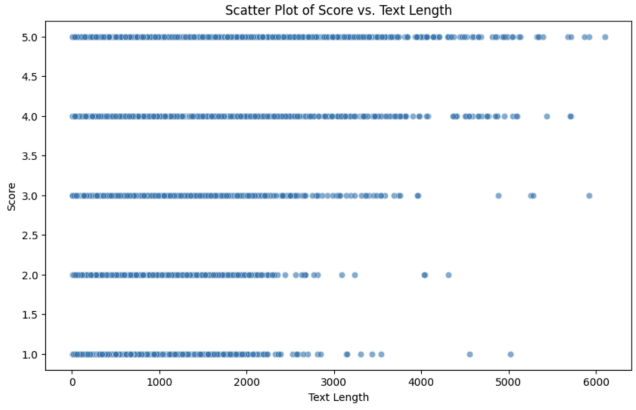
The analysis revealed minimal outliers and an even distribution of helpfulness across scores, though the average helpfulness score was relatively low at around 0.35. Helpfulness appeared to be a polarized variable, with reviews rated as either very helpful or not at all. This led me to create a new feature, Helpful, defined as the ratio between the numerator and denominator. Representing this variable as a value between 0 and 1 improved its predictive accuracy, suggesting that more helpful reviews were correlated with more accurately predicted scores.

Moreover, continuing on this thought I created a Review Length variable to represent the effort of writing a review and explore if it had any correlation with score, sentiment, and helpfulness. There was barely any correlation between Review Length and Text Sentiment (-0.009702) but there was a stronger positive correlation with Helpfulness (0.304723) indicating that indeed a longer review makes it more helpful.

```
Correlation Matrix:
                ReviewLength  Helpfulness  Text_Sentiment
ReviewLength        1.000000     0.304723       -0.009702
Helpfulness         0.304723     1.000000        0.036624
Text_Sentiment     -0.009702     0.036624        1.000000
```







The second thing I wanted to determine was if there was a class imbalance. My assumption was that there would have been more scores for 1 and 5 but to my surprise the dataset was imbalanced towards higher scores, 4 and 5. The lowest frequency was for 1 and this was 0.11 as many as the most frequent which was 5. Taking this into account, I decided to balance the dataset fully. Even though this can sometimes lead to a model that doesn't accurately reflect real-world data where classes are often naturally imbalanced, I decided to balance the model so it wouldn't become biased towards a class. I also looked if there were any missing values using a heatmap but there were only some scores missing and since the score column was dropped it should not impact the model.

Finally, I wanted to determine whether to use both Summary and Text or only one since in my head they seemed pretty similar which could lead to overfitting and redundancies. By looking at the dataset I found that many summaries were either gibberish or very brief descriptions of the reviews. In the word cloud above we see the words used themselves were not similar, which could have meant that summary also contributed to the model. However, when adding the summary features a lot of noise was created and the accuracy of the model went down significantly from 0.395 (at that point) to 0.219, hence why in the final model Summary was not a feature.

Feature Engineering and Selection

Regarding feature engineering I realized when they were movie reviews that there would have to be some sort of sentiment analysis since more positive sentiment would most likely align with higher ratings. After some research on non-NLP tools, I found TextBlob and VADER. In the end I chose to use VADER on the 'Text' field instead of TextBlob because of VADER's design towards online media and slang. I extracted only the compound sentiment score that reflects the polarity of each review, as it was redundant to have positive, neutral, and negative. This score best captures the review's emotional tone, providing additional predictive power by quantifying positivity or negativity. A limitation I ran into with VADER was that there were several bottlenecks when given such a big dataset. To address this, I implemented parallel processing, distributing the computation across multiple CPU cores. Using Python's joblib library, I applied VADER sentiment analysis concurrently, significantly reducing the processing time.

In the feature engineering process, I initially considered using TF-IDF vectorization on the text data to capture the importance of words across reviews, which had been mentioned in class. However, after implementing and testing it, I found that TF-IDF significantly increased computation time without improving the model's accuracy greatly. Consequently, I decided not to include it in the final model to ensure a more efficient solution, although without a time constraint I would have fine-tuned TF-IDFfor this dataset specifically.

In addition to sentiment analysis, I explored various other features to enhance model performance. I used the Id feature, which uniquely identifies each review, primarily for tracking and aligning records across different datasets and ensuring consistency without directly influencing the model's predictions.

Time-based features were another essential part of the feature engineering process. Converting the timestamp data into separate features—Year, Month, Day, and DayOfWeek—allowed the model to capture potential seasonal or day-specific variations in review ratings, which I assumed would be relevant in the context of user behavior. Reviews during holiday seasons could exhibit different patterns in rating

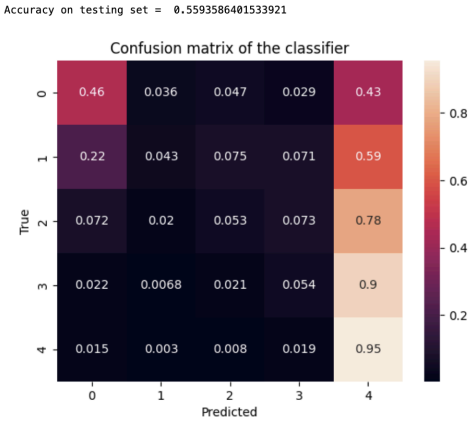
trends and I found that there were more reviews during the weekend, although the sentiment had the same proportions week day and end.

Model Creation and Evaluation

For this project, I experimented with several models to find the best balance between accuracy, interpretability, and efficiency. Initially, I explored K-Nearest Neighbors (KNN) and Ridge Classifier due to their simplicity, my knowledge on KNN from lectures and their well-established performance in classification tasks. However, both models had limitations for this dataset. KNN, being a lazy learner, requires high memory and is computationally expensive on large datasets, which makes it impractical given the time constraints. Ridge Classifier, did not perform as well in distinguishing between the various classes, particularly with highly imbalanced data. Ultimately, I chose Gradient Boosting Classifier, as it offers a more advanced ensemble approach that combines weak learners to improve overall predictive power. Its iterative learning process made it a natural choice, especially considering the complexity of predicting star ratings from text-based reviews.

To optimize the Gradient Boosting model, I conducted parameter tuning using RandomizedSearchCV. This process allowed me to explore a broad range of hyperparameters, such as n_estimators, learning_rate, max_depth, and subsample. However, due to the substantial size of the dataset, running a full grid search was infeasible. Instead, I used a subset of the training data, sampling between 5-20% of it to reduce computational load while maintaining parameter relevance to the full dataset. There was not much variation in the n_estimators and max_depth which were almost at every run 200, 3, but yes for the other three. The combination that gave the best accuracy was subsample: 1.0, n_estimators: 200, min_samples_split: 2, max_depth: 3, learning_rate: np.float64(0.2).

To further optimize processing time and avoid kernel crashes, I incorporated a "special trick" where I converted sparse matrices to dense format only when necessary and utilized parallel processing to distribute the workload across multiple CPU cores. This approach significantly minimized runtime and memory issues, enabling smoother execution during model training.



The confusion matrix provides insights into the model's classification tendencies. Although I balanced the dataset, the model still struggled with mid-range scores. The confusion matrix shows strong classification for ratings of 4 and 5, but lower accuracy for scores 2 and 3, suggesting ongoing difficulty in distinguishing moderate sentiments. Refining the class balancing or adding new features may help improve mid-range accuracy.

In conclusion, this project effectively used Gradient Boosting to predict Amazon movie ratings, reaching a 0.56 accuracy. Key takeaways include the importance of feature engineering, where adding VADER sentiment analysis, review length, and helpfulness metrics improved predictive power. Computational optimizations like subsampling, parallel processing, and sparse matrix handling were essential for managing runtime and memory constraints. Although the model faced challenges with feature selection and class imbalances, more advanced balancing and NLP techniques like embeddings could enhance future performance. Overall, this approach highlights the impact of thoughtful feature selection and efficient processing on model success, providing a strong basis for further improvements.