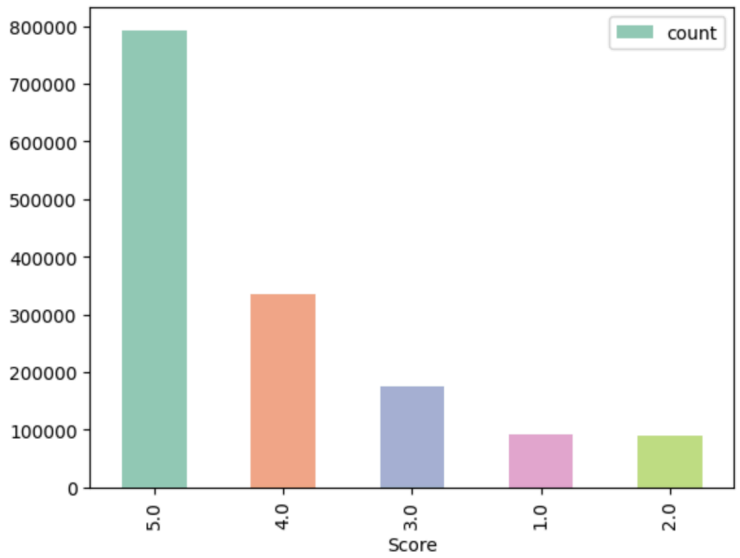
Amazon Movie Rating Prediction

By Katelyn (Kae) Chi

Section 1: Data Analysis

The dataset consisted of around 1.7 million rows, which included fields for identifying products and users, capturing review helpfulness, rating scores, timestamps, and the review content (summary and full text).



My initial observations suggested potential biases within the dataset. For instance, there is a noticeable skew toward higher ratings, as illustrated in the following figure. This led me to explore patterns that might contribute to high ratings and informed my approach to grouping features based on their influence on positive feedback. Additionally, I assumed that certain review characteristics (like helpfulness and length) might correlate with higher user satisfaction, and these assumptions guided my feature engineering process.

Section 2: Algorithm

The purpose of adding features is to shape the model's understanding of various factors influencing user-generated movie ratings. By analyzing components such as review helpfulness, timing, user history, and text features, each feature captures distinct insights relevant to predictive accuracy. I decided to group the features based on the purpose they served, as shown below:

1. **Helpfulness**- Variables like HelpfulnessNumerator, HelpfulnessRatio, and WeightedHelpfulness quantify the impact of reviews based on audience engagement, providing a measure of social validation. Missing values are filled as zero to standardize the data and ensure sparse engagements are accounted for in the model. I did this as the observation that helpful reviews may tend to correlate with higher ratings, potentially influencing the predictive weight of this feature category.

2. **Timing** - Time-related features such as Month, IsWeekend, and DayOfYear reveal if specific periods influence rating trends, capturing seasonal or trend-based variations. These attributes, extracted through date time conversion, help the model identify patterns, such as holiday boosts or weekend leniency in ratings. My hypothesis here was that certain times of year might yield more favorable reviews, a trend seen in similar datasets.

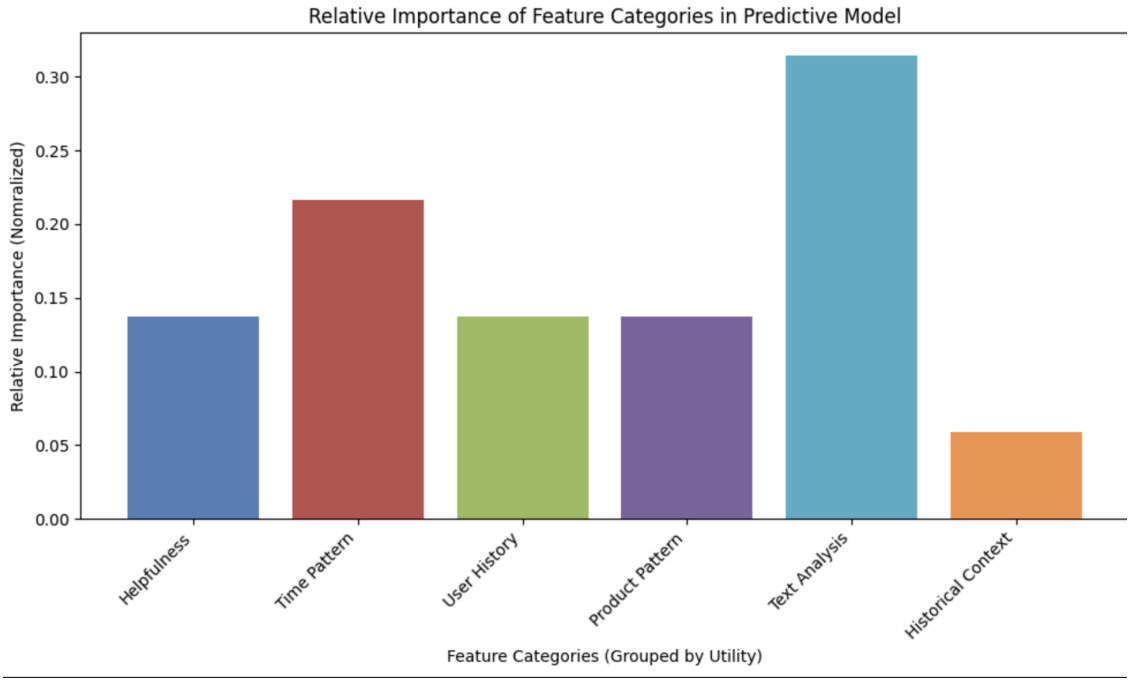
3. **User History -** Metrics like UserMeanScore, UserStdScore, and UserReviewCount capture users' rating tendencies, including leniency and frequency. User statistics, derived through grouped aggregations, allow the model to account for individual biases in rating predictions. This feature

group was motivated by the observation that some users tend to rate more generously or critically on average, and identifying these tendencies could help the model differentiate among user patterns.

4. **Product Patterns -** Features including ProductMeanScore, ProductStdScore, and ProductReviewCount provide insights into a product's popularity and reputation. Patterns here suggested that products with higher mean scores might receive additional high ratings, creating a feedback loop that could bias predictions toward popular items.

5. **Text Analysis -** Text-based features, such as ReviewLength, TextSentiment, and CapitalWordCount, quantify the sentiment, verbosity, and emotional emphasis in reviews. Using lambdas to calculate text characteristics and TextBlob to capture sentiment and subjectivity, the model can interpret emotional cues that often align with ratings.

6. **Historical Context -** Variables such as DaysSinceFirstReview and ReviewTimePosition position reviews along a movie's timeline, indicating potential novelty or recency effects. Mapping the first and last review timestamps allowed the model to detect shifts in sentiment over time, identifying trends like initial excitement or later dissatisfaction.



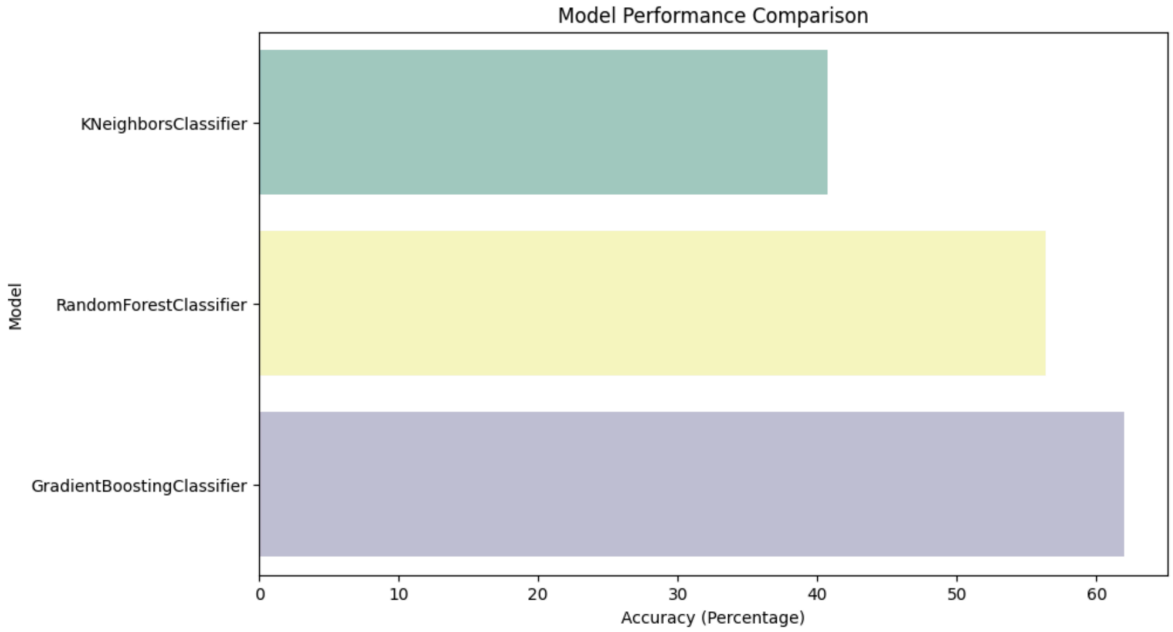
Weighting of Features

The figure provides the weighting of each category of features. I calculated this using the formula: $category\ weight\ = \frac{\#\ of\ features\ in\ category}{total\ \#\ of\ features}$. My model puts the most emphasis on semantics and a user's summary when determining a prediction.

Section 3: Training, Modeling, and Validating Data

Training
The training process involved splitting the dataset into training and testing sets to ensure model evaluation on unseen data. Seventy-five percent of the data was allocated for training and 25% for testing, with a fixed random state to ensure reproducibility.

Models



I utilized three different models when determining how best to make predictions from my data.

- **KNeighborsClassified-** This algorithm was initially selected for its simplicity and ease of implementation. However, it achieved only 40.736% accuracy, indicating limited effectiveness in handling the complex patterns within the data, particularly with such a large dataset.

- **RandomForestClassifier-** Utilizing multiple decision trees, this approach improved prediction accuracy, reaching 56.357%. Although it captured some broader relationships in the data, it still struggled to fully represent complex interactions due to the averaging process across trees.

- **GradientBoostingClassifier-**Gradient boosting provided the highest accuracy at 62.004%, leveraging iterative boosting to refine predictions and adjust for errors. This algorithm's ability to focus on misclassified instances helped it perform better, especially in the skewed dataset with a concentration of high ratings.

Section 4 Model Limitations and Conclusions

This project utilized engineered features across dimensions like user history, product popularity, and review sentiment, alongside tools like TextBlob for NLP methods and lambdas for data organization, to predict Amazon movie ratings. Among the tested algorithms, Gradient Boosting achieved the best accuracy by capturing complex relationships in the data.

Section 5 Citations

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

2. Parrish, A. (n.d.). TextBlob. Retrieved October 28, 2024, from https://aparrish.neocities.org/textblob

3. Scikit-learn. (n.d.). KNeighborsClassifier. Retrieved October 28, 2024, from https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

4. Scikit-learn. (n.d.). RandomForestClassifier. Retrieved October 28, 2024, from https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html

5. W3Schools. (n.d.). Python Lambda. Retrieved October 28, 2024, from https://www.w3schools.com/python/python_lambda.asp

I also consulted ChatGPT for additional information to improve my algorithm and model.