Amazon Movie Reviews Analysis Report

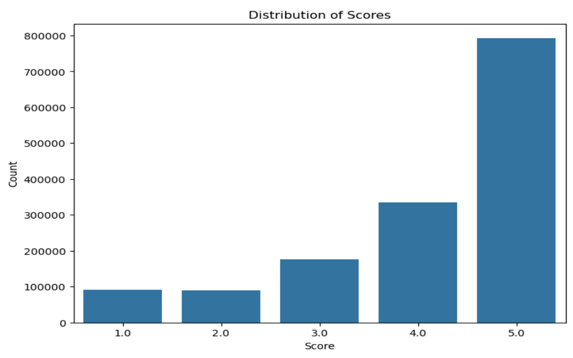
Alan Lin (alanl193@bu.edu) | Kaggle: alanl193

1. Preliminary Analysis

The dataset comprises approximately 1.7 million entries with columns such as `Summary`, `Text`, `HelpfulnessNumerator`, `HelpfulnessDenominator`, and `Score`. The initial step involved thoroughly exploring these features to understand their potential impact on star ratings.

1.1 Data Structure & Distribution

- **Textual Features**: Both Summary and Text are rich in content, with millions of unique entries. This diversity captures a wide range of opinions and sentiments that could influence ratings.
- **Numerical Features:** The `HelpfulnessNumerator` and `HelpfulnessDenominator` provide a quantitative measure of how useful users found each review, which could correlate with the reliability or quality of the review content.



1.2 Data Quality

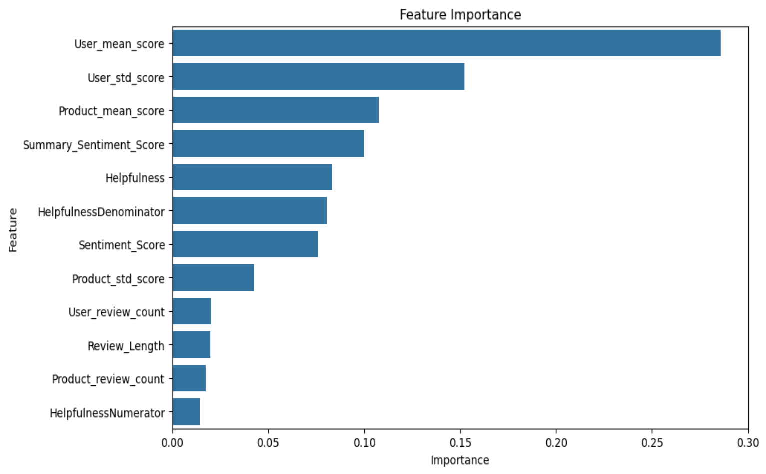
The Score feature is skewed towards higher ratings, reflecting a tendency for positive reviews. This distribution informed my approach, ensuring that I design a model capable of accurately predicting the majority class while addressing class imbalance.

2. Feature Extraction

Choosing and extracting features was a vital step in enhancing the model's predictive capabilities. Here's a detailed look at the process and insights:

- **Sentiment Analysis:** I used VADER sentiment analysis to extract sentiment scores from both `Text` and `Summary`. This provided a quantitative measure of sentiment, capturing nuances that correlate with review scores. The compound score was useful, as it summarizes the sentiment in a single value.
- **User-Based Features:** By aggregating the scores for each user, I captured individual reviewing patterns. Features such as `User_mean_score`, `User_Std_score`, and `User_review_count` helped account for user biases. For instance, some users might consistently rate products higher or lower than the average, and understanding this tendency was crucial for accurate predictions.

- **Product-Based Features:** Similar to user features, product-level statistics like `Product_mean_score` and `Product_review_count` provided insights into general product reception. This helped the model understand the baseline expectation for product ratings, allowing it to adjust predictions accordingly.
- **Textual Features:** Features like `Review_Length` were derived from the text data. Longer reviews might indicate more detailed feedback, potentially correlating with extreme scores. This feature helped capture the depth of the review content.
- **Helpfulness Metrics:** The `Helpfulness` ratio, calculated as `HelpfulnessNumerator` divided by `HelpfulnessDenominator`, offered insights into the perceived utility of a review. Reviews deemed more helpful by users might carry more weight in determining the overall score.



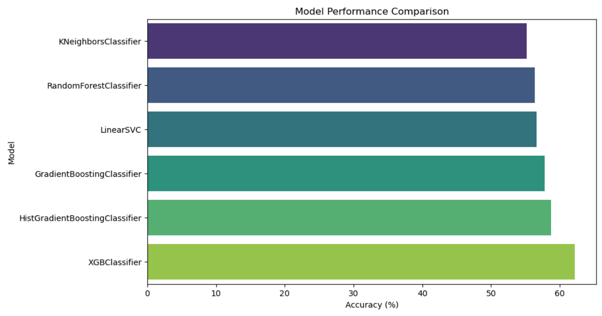
3. Used Techniques & Experiments

During the development process, I experimented with various classifiers to optimize model performance:

3.1 Techniques - Model Selection & Iteration

- **KNeighborsClassifier:** Initially I tried the K-Nearest Neighbors algorithm due to its straightforward approach. However, it struggled with scalability, especially given the large dataset size, leading to less effective handling of complex data patterns. (Kaggle Score: 55.212%)
- **RandomForestClassifier:** This approach leveraged multiple decision trees to enhance prediction accuracy. While it offered improvements over KNN, it faced challenges in capturing intricate data patterns due to its reliance on averaging decision trees outputs. (Kaggle Score: 56.357%)
- **LinearSVC (SVM):** I explored Linear Support Vector Classification for its computation efficiency with linearly separable data. However, it did not adequately capture non-linear relationships present in the dataset, limiting its effectiveness. (Kaggle Score: 56.607%)
- **GradientBoostingClassifier:** This approach improved accuracy but was computationally intensive, making it less feasible for very large datasets. (Kaggle Score: 57.778%)

- **HistGradientBoostingClassifier:** This variant of gradient boosting offered faster computation times and better handling of large datasets, thanks to its histogram-based approach. (Kaggle Score: 58.758%)
- **XGBClassifier:** Ultimately, I selected XGBoost for its ability to handle non-linear relationships effectively and its robust regularization capabilities. XGBoost provided a balance of accuracy and computational efficiency, making it the most suitable choice for the dataset's complexity and size. (Kaggle Score: 62.198%)



3.2 Experiments

- **Hyperparameter Tuning:** For each model, I conducted grid searches to tune parameters such as `n_estimators`, `learning_rate`, and `max_depth`, optimizing the tradeoff between bias and variance.
- **Feature Engineering:** As I iterated through models, I refined feature sets based on feedback from feature importance metrics and model performance, ensuring the most relevant features were used.

4. Model Validation

The model was validated through systematic approach:

- **Train-Test Split:** The data was split into training and testing sets (75/25) to evaluate model generalization.
- **Performance Metrics:** Accuracy and confusion matrix were primary metrics. I also considered precision, recall, and F1-score to ensure balanced performance across classes.
- **Cross-Validation:** Employed to assess the model's stability and mitigate overfitting, providing a more reliable estimate of performance.

5. Conclusion

To enhance the model performance, several approaches were employed:

- **Aggregated Features:** User and product statistics were crucial in capturing contextual information beyond individual reviews.
- **Sentiment Ratios:** Derived features such as sentiment-to-length ratios provided nuanced insights into review intensity relative to length.
- **Iterative Refinement:** Frequent iteration on the model and feature set ensured continuous improvement, adapting strategies based on observed patterns and model feedback.

Citations

GeeksforGeeks. (2021, October 7). *Python | Sentiment Analysis using VADER*. GeeksforGeeks. https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/

GradientBoostingClassifier. (n.d.). Scikit-learn. https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

HistGradientBoostingClassifier. (n.d.). Scikit-learn. https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html

KNeighborsClassifier. (n.d.). Scikit-learn. https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

LinearSVC. (n.d.). Scikit-learn. https://scikit-learn.org/dev/modules/generated/sklearn.svm.LinearSVC.html

RandomForestClassifier. (n.d.). Scikit-learn. https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html

XGBoost Documentation — xgboost 2.1.1 documentation. (n.d.). https://xgboost.readthedocs.io/en/stable/