Oscar Mo Project Writeup: Amazon Movie Reviews Rating Prediction

Introduction

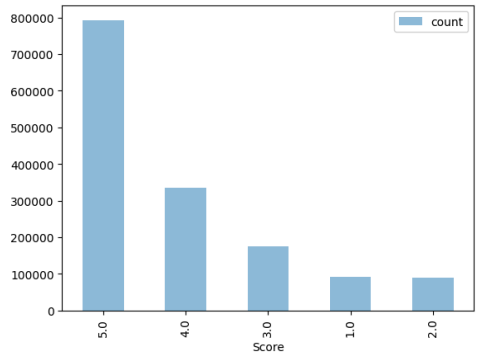
The task was to predict the score associated with Amazon movie reviews based on various features derived from the review text, user history, and product information. The goal was to develop a robust prediction model through feature engineering and data modeling techniques. This project involved creating features that captured key aspects of user behavior, product characteristics, and textual content to better understand what drives different star ratings. To approach this task of creating a viable model, it is important to analyze the data.

Data Analysis

The dataset consists of user reviews for Amazon movie products. The primary columns include:

ProductId (unique identifier for the product), UserId (unique identifier for the user), HelpfulnessNumerator (number of users who found the review helpful), HelpfulnessDenominator (number of users who indicated whether they found the review helpful), Score (rating between 1 and 5),Time (Timestamp for the review), Summary (brief summary of the review), Text (text of the review), Id (a unique identifier associated with a review)

The columns of data are rich in data which would lead to more additional columns in the feature engineering process. Additionally, it would be important to look at the data distribution regarding the scores of the amazon movie reviews. The bar chart is generated below:



The distribution of ratings is highly skewed toward 5 stars with a majority of the reviews being 5 stars which shows that user ratings are more positive when rating the movies which could affect the data modeling process. This imbalance can lead to a biased model that predicts the majority class (5-star ratings) more frequently, potentially neglecting other classes. With the observations of the graph, this would be helpful when considering feature engineering with columns of the data.

Feature Engineering

Feature engineering was a key focus in improving the model's prediction accuracy. The feature set included a combination of numerical, categorical, and text-derived features. Here are the key features generated and their significance:

Helpfulness Metrics:

a. **Helpfulness:** The ratio of HelpfulnessNumerator to HelpfulnessDenominator to capture the proportion of users who found a review helpful.
b. **IsHelpful:** A binary indicator of whether the review was rated as helpful by at least one user.
c. **HelpfulnessDifference**: The difference between the numerator and denominator, capturing the net helpful votes.
d. **WeightedHelpfulness:** A variation that normalizes the numerator by adding 1 to the denominator.

Time-Based Features:

a. Year, Month, DayOfWeek, Quarter: Extracted from the review timestamp to capture temporal patterns.

User and Product History Features:

a. **UserMeanScore:** The average score given by a user, representing their general rating tendency.
b. **ProductMeanScore:** The average score received by a product, representing its overall popularity.
c. **UserMedianScore**: The median score given by the user to capture rating tendencies while mitigating the effect of outliers.
d. **UserReviewCount:** The total number of reviews made by the user, indicating their reviewing frequency.
e. **UserProductInteraction:** The product of UserMeanScore and ProductMeanScore, indicating the interaction between a user's general rating behavior and the product's general reception.

Text-Based Features:

a. **ReviewLength:** The number of words in the review.
b. **UniqueWords:** The number of unique words used, indicating review diversity.
c. **AverageWordLength:** The average length of words, capturing linguistic patterns.
d. **Sentiment:** The polarity score calculated using TextBlob, indicating the positive/negative sentiment of the review text.

User-Based Polarity:

a. **UserPolarity:** The average sentiment polarity across all reviews made by a user. This captures the general sentiment bias of the user, which could indicate their rating tendencies.

Adding features like **Text-Based Features** and **UserPolarity** significantly boosted accuracy. **Text-based metrics** such as **ReviewLength**, **UniqueWords**, and **AverageWordLength** provided insights into review content, distinguishing detailed, higher-rated reviews from shorter, critical ones. The **Sentiment** feature captured each review's tone, enhancing the model's ability to connect review content with star ratings. Meanwhile, the **UserPolarity** feature improved accuracy by reflecting each user's average sentiment, helping the model account for consistent rating tendencies and better predict user behavior.

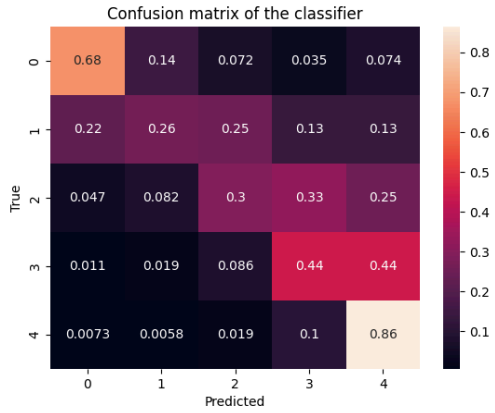
Note: this was implemented after playing around with the different models

Model Development

After trying various models such as K-Neighbors Classifier, Gradient Boosting, and Rainforest Classifier, the best results were obtained using the HistGradientBoostingClassifier. This model was chosen due to its performance and efficiency in handling large datasets.

KNeighborsClassifier: This was the first approach tried due to its simplicity and ease of implementation. With minimal feature engineering, it served as a baseline model to understand the general structure of the data.
Score: 0.5006
RandomForestClassifier: Random Forest was introduced to leverage ensemble learning through multiple decision trees.
Score: 0.53503
GradientBoostingClassifier: To address the limitations of the simpler models, this is where I discovered GBM as it is capable of capturing complex patterns through boosting and was chosen as a step up from the previous models. However, its shortcomings is that it takes a long time to run (20-45 mins), especially when trying out different parameters. The optimal parameters are n_estimators = 150, learning rate = 0.3, and random_state=42.
Score: 0.60719
HistGradientBoostingClassifier: This model is similar to GradientBoostingClassifier but is more efficient in runtime which allows me to better find the most optimal parameters for the highest score. The optimal parameters are max_iter = 100, learning_rate=0.25, max_leaf_nodes=31, random_state=42
Score: 0.60980

The model was trained on 75% of the training data and evaluated on the remaining 25% to validate performance giving a classification accuracy of 0.60980. A confusion matrix was plotted to analyze the model's performance in predicting different rating classes which is shown below:



Conclusion

The final model successfully predicted star ratings based on Amazon movie reviews with an accuracy of around 60.980%. The project highlights the importance of feature engineering and combining these features with an appropriate model, the performance exceeded initial benchmarks. However, there are aspects that I can improve such as implementing TF-IDF vectorization and finding the most optimal hyperparameter tuning with grid search CV but this was restricted due to time. Additionally, I ran into issues with overfitting when applying the additional features onto the original data which lead to a big disparity between my local and kaggle scores.

Work Cited

https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/

https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html

https://scikit-learn.org/1.5/modules/ensemble.html

https://aparrish.neocities.org/textblob