Rashfiqur Rahman
Professor Galetti
CS 506: Tools for Data Science
Kaggle Username: rashfiqur
28 October 2024

Predicting Amazon Star Ratings Based on Reviews

Introduction

This project involves predicting Amazon review star ratings based on review text and other metadata. With the dataset provided from Amazon's movie reviews, the goal was to classify each review into one of five star ratings using features such as review text, helpfulness scores, and metadata about each review. The dataset is substantial, with over 1.7 million unique reviews in the training set and over 200,000 in the test set. Each review includes fields like a brief summary, the full review text, and indicators of helpfulness. My approach focused on using text-based features derived from Summary and Text, alongside metadata, and balancing the model's sensitivity across classes.

Feature Engineering and Preprocessing

Handling Missing Values: For the *HelpfulnessDenominator* field, which records the total number of users who rated the helpfulness of a review, I replaced any zero values with NaN. This transformation allowed me to compute a ratio that wouldn't lead to divide-by-zero issues. I also created a *Helpfulness* feature, calculated as *HelpfulnessNumerator / HelpfulnessDenominator*. This ratio helps interpret how helpful users found a review in proportion to the total number of responses. Missing values were then filled with 0 to handle cases where no helpfulness data was available.

Text-Based Features: The *Summary* and *Text* fields were combined into a single feature called *Combined_Text* to capture both the condensed and detailed aspects of each review in a unified format. This consolidated text field formed the basis for TF-IDF vectorization. Additional features included *Summary_length* and *Text_length*, representing the character count of each text field. These features were useful as they could correlate with user engagement; for example, longer reviews might indicate stronger opinions or experiences. I also got rid of the time features such as review *Year, Month, Day* because I found out that those features were less useful in my model, and removing them from the feature list resulted in a 2% more accurate model. Preprocessing steps included converting text to lowercase, removing non-alphabetic characters, and eliminating common English stopwords (like "the" and "is") using NLTK's stopword list. These steps help retain only meaningful words, reducing noise in the model. I used the TF-IDF (Term Frequency-Inverse Document Frequency) method to vectorize the *Combined_Text* field. TF-IDF assigns higher weights to unique words, capturing important keywords across different reviews. I limited the vocabulary to the top 5000 words with a maximum n-gram length of 2, optimizing for performance and interpretability while capturing bigrams that might carry meaning (e.g., "not good"). Given the dataset's size, processing the entire training set was computationally expensive. To overcome this, I only sampled 10% of the dataset for initial feature engineering, model tuning, and evaluation. This sampling allowed for faster iterations and testing without compromising the representativeness of the training data.

Model Selection and Tuning

After completing feature engineering, I opted to use Logistic Regression for classification. I had tried running KNN, and Random Forest as well, but Logistic Regression was the classification method that gave me the best results. Logistic regression allowed me to directly interpret feature importance, particularly helpful for understanding which words or phrases strongly correlate with specific ratings. Logistic regression can efficiently handle large, sparse datasets, especially when combined with TF-IDF features.

In initial analyses, I observed imbalanced class distribution across star ratings, with certain ratings (e.g., 5 stars) appearing more frequently than others with 53%. I looked at the distribution and it was definitely skewed. To address this, I set the *class_weight* parameter to *'balanced'* in the logistic regression model. This approach adjusts the weights inversely to class frequencies, helping the model treat all classes more equitably.

Hyperparameter Tuning

I used GridSearchCV to optimize hyperparameters, including *C*: inverse regularization strength, with smaller values imposing stronger regularization to avoid overfitting ; *solver*: I selected *'liblinear'* as it performs well with smaller datasets and supports L2 regularization, which was a good match for the feature structure ; *penalty*: L2 regularization was chosen as it is less prone to overfitting in high-dimensional TF-IDF feature spaces. After tuning and trying a group of parameters, I found that *C=1, solver='liblinear', and penalty='l2'* yielded the best results. The model achieved a validation accuracy of approximately 63.4%, indicating a reasonable ability to distinguish between ratings based on the engineered features. I tried other *solvers* like *newton-cg, lbfgs* and penalties such as *elasticity, and l1*, and a range of values for *C.*

Key Insights and Patterns

During the model training, I noticed that certain words and phrases strongly correlated with specific star ratings, as shown in the feature importance plots. 1-star ratings frequently included words like "worst," "terrible," and "avoid." 5-star ratings were often characterized by words like "excellent," "favorite," and "amazing." This alignment of words with ratings highlighted the model's reliance on sentiment-laden terms, validating the effectiveness of TF-IDF features. Longer reviews tended to correspond with more extreme ratings (1 or 5 stars), suggesting that users were more likely to write detailed feedback when they felt strongly about the product. Including *Text_length* and *Summary_length* as features helped capture this trend.

Evaluation and Visualizations

The top TF-IDF features for each rating provided insight into the language typically associated with different star ratings. For example, 1-star reviews often used highly negative terms like "worst" and "horrible." 3-star reviews included more neutral words, reflecting moderate satisfaction. 5-star reviews leaned heavily on positive descriptors like "amazing" and "excellent." The classification report provides a detailed breakdown of precision, recall, and F1-score for each class. The model performed best on extreme ratings (1 and 5 stars) but struggled with middle ratings (2, 3, and 4 stars), where sentiment is more nuanced. The distribution plot revealed that the model correctly captured the general distribution of ratings but occasionally misclassified similar ratings (e.g., 4 as 5, or 2 as 3). Improving model performance on these middle ratings could be a target for future development. After tuning and evaluating the model on a sample, I trained the final logistic regression model on the complete (filtered) dataset, removing rows with missing target values (*Score)*. This comprehensive model was used to generate predictions for the test set.

Improvements

Throughout this project, I tried a bunch of techniques. Initially I took the starter code, and tried KNN with n_neighbors set to 30 which gave me an accuracy of around 53%. I also tried Random forest and logistic regression classifiers and that gave me around 53-55% accuracy. I figured out that possibly my model was constantly predicting ‘5’ as the score or overfitting which is why I was getting around 53% accuracy. Then to combat this, I tried introducing TFIDF on the ‘Text’ column, and treated the problem as a binary classification problem where my model only predicted 1 or 5 (bad or good). The sole purpose of this was to examine whether my model was able to separate the extremely good and bad reviews correctly. This accuracy was around 58-59% which made sense as the ‘1’ and ‘5’ ratings made up around that much of the data altogether. I then introduced Gridsearch, which improves accuracy by exhaustively searching for the best hyperparameters, allowing the model to fit the data more effectively without overfitting or underfitting. By using cross-validation, it ensures that the selected parameters generalize well across different data splits, resulting in a more accurate and robust model. Through iterative experimentation with KNN, Random Forest, and logistic regression, I identified issues with overfitting and constant predictions, leading me to implement TF-IDF on the ‘Text’ column and simplify the problem to binary classification initially. Introducing GridSearchCV further improved accuracy by tuning hyperparameters with cross-validation, and my final submission achieved 63% accuracy by leveraging optimized logistic regression with TF-IDF features across all star ratings.

Conclusions and Future Improvements

The logistic regression model provided a solid baseline, achieving reasonable accuracy in predicting star ratings. While the model handled extreme ratings well, there’s room for improvement with middle ratings. Potential future enhancements could include: models like Gradient Boosting might capture nonlinear relationships better, particularly for reviews with mixed sentiments. Also, fine-tuning the TF-IDF parameters, such as n-gram range or vocabulary size, may further improve feature representation. For the sake of time and computing power, I wasn’t able to perform a lot of these tactics, but these are some suggestions to improve upon my implementation in the future.