Afzal Khan

Professor Galletti

CS506

28 October 2024

Algorithm Implementation Report for Amazon Movie Reviews Rating Prediction

Introduction:

The goal of this project was to create a reliable algorithm that could accurately predict movie review ratings on Amazon. With such a large volume of user reviews, an effective model needed to learn from various types of information found in the reviews—such as user sentiments, length of the review, helpfulness ratings, and specific word choices—to predict the rating accurately. I started this project by experimenting with a K-Nearest Neighbors (KNN) model, a straightforward and easy-to-implement machine learning algorithm. However, I quickly realized that KNN was not suitable for this complex task due to its limited ability to handle the kind of high-dimensional, text-heavy data I had. KNN's performance was modest, achieving an accuracy of only around 0.40, which highlighted the need for more sophisticated techniques to improve accuracy.

After experimenting with K-Nearest Neighbors (KNN) and achieving limited accuracy, I shifted to Logistic Regression as my next approach. Logistic Regression is often a good choice for classification tasks, especially when dealing with text because it can model probabilities and handle high-dimensional data reasonably well when combined with text vectorization techniques like TF-IDF. Despite these strengths, the model only achieved an accuracy of around 0.53. This indicated that while Logistic Regression might be suitable for the task, it would require further optimization and richer features to improve its accuracy.

To achieve better results, I focused on feature engineering, data sampling, and model selection as ways to refine the approach. Feature engineering was especially critical, as it allowed me to extract meaningful patterns from the data. By creating and adding custom features—like text length, helpfulness ratios, sentiment indicators, and word counts—I could capture insights that were not immediately available from the raw data. Through a series of enhancements, I managed to incrementally improve the model's performance, moving from the initial 0.53 accuracy to 0.54 because of the use of RandomForest Classifier, and ultimately reaching 0.63. This report will detail the final algorithm I implemented, the specific tweaks and optimizations that helped improve its performance.

Data Sampling And Feature Engineering

Working with the full 1.7 million rows in the train.csv dataset was computationally intensive, making iterative model development challenging. To address this, I used a random sample of 10% of the training data for feature engineering and model tuning, allowing for faster processing while preserving a representative subset of the data. This approach enabled quick testing of different feature combinations

and text preprocessing steps, making model tuning feasible without compromising quality. Feature engineering played a crucial role in enhancing the model's predictive performance. In addition to text processing, I introduced numeric features that offered structural insights into each review. For example, the Helpfulness Ratio—calculated as the ratio of HelpfulnessNumerator to HelpfulnessDenominator was designed to capture the perceived helpfulness of each review, with zeros applied to avoid division errors. The idea was that reviews endorsed by other users might reflect stronger opinions or thoughtful feedback. Additionally, I calculated both character and word counts for the Text and Summary fields, assuming that longer reviews likely contained more detailed opinions, potentially corresponding to higher or lower ratings. To consolidate the text data, I combined the Summary and Text fields into a single Combined_Text feature, creating a unified view of each review's content and helping the model capture both sentiment and detail without distinguishing between the summary and main text

Text Preprocessing and Vectorization

Text preprocessing involved several simple steps to clean up and focus the text data, making it easier for the model to understand. First, I processed the Combined_Text feature by removing any non-letter characters to keep only meaningful words. Then, I converted everything to lowercase to avoid treating words with different cases as separate entries. I also removed common words, like “and” or “the,” which don’t add much value for prediction. After cleaning, I transformed the text into numbers using TF-IDF Vectorization. This method gives more weight to words that are common within a review but less common across other reviews, helping the model focus on unique language patterns. I also used bigrams (1-2 word pairs) to capture important phrases, like “highly recommend” or “not worth,” that often reflect sentiment. Limiting TF-IDF to 5000 features kept the model efficient. This transformation gave the model a structured view of the text, allowing it to spot patterns and associations useful for predicting review ratings.

Combining Numeric and Text Features

After vectorizing the text data, I combined it with the other numeric features (such as Helpfulness Ratio and Text Length) using a sparse matrix. This combination allowed the model to consider both the semantic content of the reviews and the quantitative indicators of review quality. Using a sparse matrix format preserved memory efficiency and made it easier to handle the high-dimensional TF-IDF output alongside the simpler numeric features. By capturing both content and structure, this approach helped improve predictive performance by providing a comprehensive view of each review.

Logistic Regression with GridSearchCV

For the final model, I chose Logistic Regression because it performs well in classification tasks, especially with high-dimensional text data. To optimize the model, I used GridSearchCV for hyperparameter tuning, focusing on key parameters. This included adjusting the regularization strength (C) to balance model complexity and accuracy, preventing overfitting, and selecting the best solver and penalty (liblinear, L2) for efficient convergence on large datasets. The optimal settings were found to be C=10, solver=liblinear, and penalty=l2, with balanced class weights to address class imbalances. This tuning improved the model’s ability to generalize, achieving a validation accuracy of 0.63.

Model Evaluation

Once tuned, the model was tested on a validation set, achieving an accuracy score of 0.63. This score represented a significant improvement over the initial models, validating the benefit of my enhanced feature engineering and parameter tuning. To ensure robustness, I also evaluated the model on the full training data, refining the predictions with the most optimized Logistic Regression settings. The confusion matrix in the code file shows that the model performs best at predicting the extreme ratings, with 69% accuracy for class 0 (lowest rating) and 81% for class 4 (highest rating). However, it struggles with mid-range ratings, particularly in class 1 and class 2, which are often misclassified into neighboring classes. For instance, class 1 is only correctly predicted 39% of the time, with frequent misclassifications as class 0 and class 2. Similarly, class 3 has a high misclassification rate with class 4. This pattern suggests that the model can easily identify very high or very low ratings, likely due to more distinct language or features, but has difficulty with subtle distinctions in the middle range. Improving the model's ability to capture these nuances could help reduce misclassification among these adjacent classes

Conclusion

In this project, I developed a Logistic Regression model for predicting Amazon movie review ratings using a combination of text and numeric features. By sampling 10% of the dataset for efficient training and implementing thoughtful feature engineering, such as TF-IDF vectorization and numeric metrics like helpfulness ratio and text length, I was able to build a model that captures meaningful patterns in review content and structure. Using GridSearchCV, I tuned key hyperparameters to optimize the model's accuracy, achieving a balanced setup that reduced overfitting and improved generalization. Despite these improvements, challenges remain, particularly in distinguishing between mid-range ratings due to overlapping language patterns.

Future Improvement

One way to improve this model is by focusing on the mid-range ratings, where it tends to make the most mistakes. While it does well with very high or very low ratings, it has trouble with the middle ones because the language used in these reviews is often similar. To address this, I could try using more advanced text analysis tools, like word embeddings (Word2Vec or GloVe), which better capture the context of words. I could also explore deep learning models, such as LSTMs or transformers (like BERT), which are good at understanding complex language patterns. Another idea is to use ensemble methods, combining Logistic Regression with other models like Gradient Boosting or Random Forests, to capture different patterns in the data. Additionally, adding features that focus on specific words or phrases common to each rating level could help the model better understand subtle differences, improving accuracy for moderate ratings.