The Kaggle competition tasked participants with predicting star ratings for Amazon Movie Reviews using metadata and textual information. The dataset consisted of 1.6 million training reviews and 212,192 test reviews, each accompanied by features such as UserId, ProductId, Text, Summary, and Score. The objective was to develop a machine learning model to accurately predict the star ratings while adhering to the competition's rules, which prohibited the use of deep learning techniques.

In developing the solution, I initially experimented with several algorithms, including K-Nearest Neighbors (KNN) and Logistic Regression. However, these models had limitations. KNN was highly sensitive to noise and struggled with the large dataset size, leading to suboptimal performance. Similarly, Logistic Regression was not effective at capturing the complex relationships present in the data, especially when it came to handling text and numerical metadata simultaneously. Consequently, I opted for the Random Forest Classifier, an ensemble learning method known for its robustness, flexibility, and ability to handle high-dimensional and imbalanced data efficiently. This model allowed for easy interpretation of feature importance and performed better in early tests.

A significant component of improving model performance was effective feature engineering. I created features both from the existing metadata and from extracting new insights from the text. The first key feature was the helpfulness ratio, calculated as the ratio of the HelpfulnessNumerator to the HelpfulnessDenominator. This feature provided an indication of how useful each review was perceived, with the assumption that more helpful reviews were likely to provide accurate star ratings.

Another important feature was the review length, measured as the number of words in each review. Longer reviews often provided more detailed opinions, which correlated with more extreme ratings (either 1 or 5 stars). To further capture the sentiment expressed in the reviews, I employed the VADER sentiment analysis tool from nltk. This allowed me to extract two additional features: polarity, indicating the overall sentiment (ranging from negative to positive), and subjectivity, measuring the degree of opinion in the review. These features were crucial in helping the model differentiate between varying star ratings based on the emotional tone of the text.

In addition to these basic features, I implemented text pattern features. I counted the number of exclamation marks and question marks in each review, as these punctuation marks often convey strong emotion or dissatisfaction. The presence of these symbols provided additional insight into the nature of the review, enhancing the model's ability to distinguish between different ratings. Furthermore, I calculated the average word length in each review, which served as an indicator of the review's tone and complexity. Longer words often signaled a more descriptive or formal review style, which I found to be associated with higher ratings. Lastly, I extracted the review

year from the timestamp to understand whether the timing of the review impacted the rating. Older reviews, for instance, were often rated lower, possibly reflecting changing viewer expectations or movie quality over time.

Once the features were engineered, the next step was to optimize the model. I used Grid Search Cross-Validation (GridSearchCV) from scikit-learn to fine-tune the hyperparameters of the Random Forest Classifier. The grid search focused on key hyperparameters such as the number of trees (n_estimators), maximum tree depth (max_depth), and the minimum number of samples required to split a node (min_samples_split). After extensive testing, the best model configuration was found to be 200 trees with a maximum depth of 20 and a minimum of 4 samples per leaf. This configuration struck a balance between minimizing overfitting and maximizing performance on unseen data.

Throughout the model development process, I identified several patterns and employed specific strategies to enhance accuracy. One of the most notable patterns was the imbalance in star ratings, where certain scores, such as 4 and 5 stars, were much more frequent than others. To address this issue, I applied class weights within the Random Forest model, giving more importance to the minority classes (1 and 2-star ratings). This adjustment improved the model's ability to correctly classify these underrepresented ratings, contributing to a higher overall accuracy.

Another strategy was the strategic combination of textual and numerical features. I noticed that reviews with extreme scores (1 or 5 stars) often had strong sentiment polarity. By integrating polarity, review length, and the helpfulness ratio, I could create a robust set of features that helped the model accurately identify these extreme cases. The polarity feature, in particular, played a significant role, as highly negative reviews were almost always associated with low star ratings, while highly positive reviews corresponded with higher scores. Additionally, using temporal information by incorporating the review year allowed the model to adjust its predictions based on when the review was posted, providing an additional layer of insight.

The final model achieved an accuracy of 0.54 on the Kaggle leaderboard, representing a significant improvement from earlier versions. Throughout the process, several assumptions guided the model development. First, I assumed that the textual features extracted, such as sentiment polarity, would have a direct impact on star ratings. This assumption was validated, as these features were among the most influential in the model. Additionally, applying weights to balance class distribution was based on the assumption that this would improve performance on minority classes, which proved effective. Another assumption was that longer reviews contained more detail, correlating with stronger sentiment, and this too was borne out in the model’s performance.

Looking ahead, there are several ways to further enhance the model's performance. Exploring XGBoost or other boosting algorithms could provide an additional boost to classification accuracy. Additionally, implementing TF-IDF (Term Frequency-Inverse Document Frequency) to extract more granular textual features could help identify specific words associated with different star ratings, leading to more precise predictions. Finally, using Bayesian Optimization for hyperparameter tuning could refine the model further, ensuring a more efficient and effective optimization process.

In conclusion, this project highlighted the critical importance of feature engineering and hyperparameter tuning in developing a successful machine learning model for text classification tasks. By strategically combining metadata and textual features and fine-tuning the model's parameters, I was able to achieve a robust performance while adhering to the competition's guidelines. The project demonstrated the power of traditional machine learning techniques when thoughtfully applied to a complex dataset, providing a solid foundation for future improvements.