Predicting Amazon Review Star Ratings Using Metadata

Abstract

This paper presents a predictive model designed to classify Amazon Movie Reviews based on star ratings. The challenge was to achieve high accuracy without using deep learning, focusing instead on linear classifiers, extensive feature engineering, and optimization techniques. The final model uses `LinearSVC` to classify reviews by leveraging both metadata and textual features, including user and product historical patterns, TF-IDF vectorized text, and helpfulness ratios. The model achieved a Kaggle public score of 0.64830 and a private score of 0.64952, highlighting the effectiveness of feature engineering in text classification tasks.

Introduction

With the growth of user-generated content on platforms like Amazon, sentiment analysis and review classification have become essential for business insights. Analyzing review ratings allows companies to understand customer satisfaction and make data-driven decisions for product improvement and service personalization. This project aims to classify Amazon reviews into star ratings (1 to 5 stars) based on metadata and textual data. The objectives include maximizing accuracy and computational efficiency while avoiding deep learning models. This work leverages linear classifiers, specifically `LinearSVC`, and feature engineering to create an interpretable and efficient predictive model.

Initial Approach and Model Selection

Baseline Models

Initially, baseline models such as K-Nearest Neighbors (KNN) and Naive Bayes were evaluated for their simplicity and interpretability:

- **K-Nearest Neighbors**: KNN classifies data points based on proximity to labeled neighbors, making it suitable for low-dimensional spaces. However, its high computational cost and inefficiency in handling textual data made it unsuitable for this project's high-dimensional input space.
- **Naive Bayes**: This probabilistic model assumes feature independence, which is often unrealistic for text data, where word occurrences are interdependent. Although straightforward, it achieved low accuracy and was unable to capture complex patterns.

Transition to Support Vector Classifier

Given the limitations of baseline models, we transitioned to `LinearSVC`, which is better suited for high-dimensional feature spaces. Support Vector Machines (SVMs) create a hyperplane to separate classes, making them effective with properly engineered features. To optimize the SVM's performance, `GridSearchCV` was used for hyperparameter tuning on `C` and loss settings.

Feature Engineering

Helpfulness Ratio

The `Helpfulness` feature was created by dividing `HelpfulnessNumerator` by `HelpfulnessDenominator`, providing a measure of reviewer engagement with each review. High helpfulness scores often correlate with reliable reviews, helping the model distinguish informative reviews from less-engaged ones. Null values were set to zero to avoid biases.

Temporal Features

From the `Time` column, we extracted `Year`, `Month`, and `DayOfWeek`, capturing seasonal variations and temporal patterns in review behavior. Temporal trends can reflect broader customer sentiment changes over time, improving the model's accuracy by identifying patterns that align with customer behavior shifts.

Textual Features

We added `Summary_length` and `Text_length` as indicators of verbosity. Longer reviews often indicate more detailed opinions, which can correlate with extreme ratings (positive or negative). These features allow the model to capture the depth and engagement level of a review through metadata alone.

User and Product Score Patterns

Historical patterns in user and product ratings provide valuable insights into rating behavior, allowing the model to learn from previous tendencies. For `UserId`, we derived four key features to capture user-specific patterns:

1. **User_avg_score**: This feature calculates the average score for each `UserId`, reflecting the overall rating tendency of the user. Users with high `User_avg_score` are likely to give consistently positive ratings, while lower averages suggest more critical or cautious reviewers.

2. **User_std_score**: This feature calculates the standard deviation of scores for each `UserId`, measuring the dispersion in the user's ratings. A high `User_std_score` indicates that the user's ratings vary widely, suggesting a range of opinions, while a low standard deviation reflects more consistent rating behavior.

Similarly, for `ProductId`, we included analogous features:

3. **Product_avg_score**: The average score for each `ProductId` represents the overall sentiment toward the product across all reviewers, helping the model recognize popular or highly rated products.
4. **Product_std_score**: The standard deviation of scores for each `ProductId` measures rating consistency for the product. Products with low variability in scores (low `Product_std_score`) typically have a stable reputation, while high variability suggests mixed reviews.

These user and product-based statistical features help the model leverage historical trends in user and product behavior, contributing to improved predictive accuracy.

Mode and Proportion

The most common score (mode) and its proportion for each user and product add granularity, helping identify patterns in user behavior and product reliability. This step was inspired by ensemble methods in text classification, which benefit from combining multiple feature perspectives.

TF-IDF Text Vectorization

We used TF-IDF (Term Frequency-Inverse Document Frequency) to transform text fields into numerical vectors, capturing the importance of specific words across reviews. This allowed the model to leverage common phrases, while TF-IDF reduced the dimensionality by focusing on meaningful terms. This approach is commonly used in text mining for its simplicity and effectiveness in representing word importance.

Performance Optimization

Scaling

`StandardScaler` was applied to ensure that all features were on a consistent scale, which is critical when combining sparse TF-IDF features with dense numerical data. This approach helps avoid feature dominance and ensures that each feature contributes appropriately to the model.

Sample Size and Cross-Validation

Given the dataset's size, we used a subset of 100,000 records for `GridSearchCV` optimization. This sample size balanced computational efficiency with model accuracy. Cross-validation with 3-fold splits provided robust performance evaluation without excessive processing time.

Sparse Matrix Handling

Sparse matrices from TF-IDF vectorization were combined with dense matrices using `hstack`, optimizing memory usage and computational efficiency. Handling sparse data efficiently is essential when working with high-dimensional text features in machine learning.

Assumptions and Observations

Assumptions

- **Missing Values**: Missing values in `User_avg_score` and `Product_avg_score` were replaced with the global mean, assuming that users and products without extensive review histories would not deviate significantly from the average.
- **Helpfulness as an Engagement Proxy**: The `Helpfulness` ratio was assumed to correlate with the reliability of a review, as highly-rated reviews often provide valuable insights.

Observations

Through model evaluation and feature engineering, we observed:

- **Reviewer and Product History**: High user or product averages were good indicators of future ratings. This highlights the value of incorporating historical patterns in user behavior and product reliability.
- **Text Length Correlation**: Verbose reviews often correlated with stronger opinions, which aligns with literature suggesting that text length can be indicative of rating polarity.
- **Class Imbalance**: Class distribution was skewed, with a majority of 5-star ratings. Addressing this imbalance through oversampling or class weighting may improve future performance.

Final Results and Reflection

The model achieved a test accuracy of approximately **65%**, scoring **0.64830** on the Kaggle public leaderboard and **0.64952** on the private leaderboard. Precision-recall analysis showed better performance on higher ratings (4 and 5 stars) due to the class distribution. The model demonstrated competitive performance for a linear classifier, underscoring the effectiveness of feature engineering in enhancing predictive accuracy.

Future Work

Potential improvements include:

- **Advanced Text Features**: Incorporating advanced textual features, such as sentiment analysis or embeddings, could capture nuances in language.
- **Handling Class Imbalance**: Applying oversampling or class weighting could improve performance on minority classes.
- **Reviewer History Analysis**: Analyzing reviewer history in more depth could yield insights into individual behavior trends.
- **Interpretability**: Tools like SHAP or LIME could be used to interpret feature contributions, enhancing model transparency.

References

1. Altman, N. S. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, 46(3), 175–185.
2. Zhang, H. (2004). The Optimality of Naive Bayes. *AAAI*, 3(1), 562–567.
3. Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
4. Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5), 1189–1232.
5. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.
6. Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*, 30.