Report for Amazon Movie Reviews Prediction Model

Introduction

The goal of this midterm was to create a predictive model to determine Amazon movie review ratings using a dataset of customer reviews. This report explains the steps I took to preprocess the data, engineer meaningful features, train a model using XGBoost, and evaluate its performance.

Provided Data and Features

The data was given to us in two files: train.csv and test.csv. train.csv contained many different fields such as ProductID, UserID, Score, Summary, Text, HelpfulnessNumerator, HelpfulnessDenominator, and Time. test.csv contained reviews that lacked the Score column, that we had to predict. The features provided to us were:

- HelpfulnessNumerator: The number of users who found the review helpful
- HelpfulnessDenominator: The total number of users who rated the review's helpfulness
- Time: The timestamp of when the review was posted
- Helpfulness: A ratio of HelpfulnessNumerator to HelpfulnessDenominator

While these features provided a good starting point, they are not enough to accurately predict the Score. Therefore, we must add more features to capture more information from the review data and improve model performance.

Additional Feature Engineering

To improve the model's performance, I added some features based on the data provided. My goal was to use information from the review text, user behavior, and product ratings to enhance the model's ability to predict the Score accurately. Below are the features I used and the motivation behind them:

- Helpfulness
    - A ratio of HelpfulnessNumerator to HelpfulnessDenominator
    - It helps determine how useful a review was based on other users reactions
- ReviewAge
    - Calculated by determining the days between the review date and current date
    - It helps find patterns of reviews on a product over time
- SentimentPolarity
    - Sentiment analysis applied to the Text field using the TextBlob library to compute a SentimentPolarity value ranging from -1 (negative sentiment) to +1 (positive sentiment)
    - It helps find the tone of the review text
- AverageProductScore
    - Calculate the average score for each ProductID across all reviews in the dataset
    - It helps understand the reputation of the product
- AverageUserScore

    - Calculate the average score given by each UserID across all reviews in the dataset
    - It helps find user rating tendencies
- ProductID
    - Unique identifier for each product
    - It helps find product-specific trends
- UserID
    - Unique identifier for each user
    - It helps find user-specific trends

Preprocessing and Data Handling

Many reviews had missing values for features such as Text, HelpfulnessNumerator, and HelpfulnessDenominator. Missing numerical values were filled with appropriate defaults (e.g., average scores or zero), while missing text was assigned a neutral sentiment. A small constant was added to prevent division by zero in the Helpfulness ratio. Infinite values were filtered out to ensure clean data. Numerical features were standardized using StandardScaler, and categorical features (ProductID and UserID) were encoded using OrdinalEncoder with Scikit-Learn library. This preprocessing ensured the data was consistent and ready for modeling.

Model and Evaluation

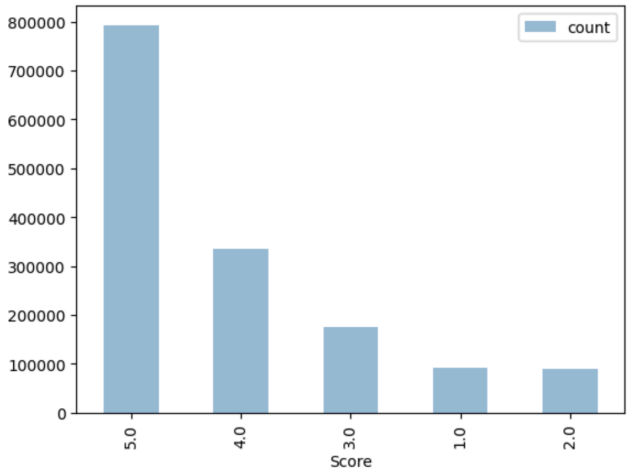
For the predictive model, I chose XGBoost, a powerful gradient-boosting algorithm known for its effectiveness in handling structured data. The hyperparameters were tuned to optimize performance while avoiding overfitting:

- max_depth=6: Restricts tree depth to prevent overfitting
- learning_rate=0.1: A moderate learning rate for better convergence
- n_estimators=100: The number of trees (iterations) used to fit the data

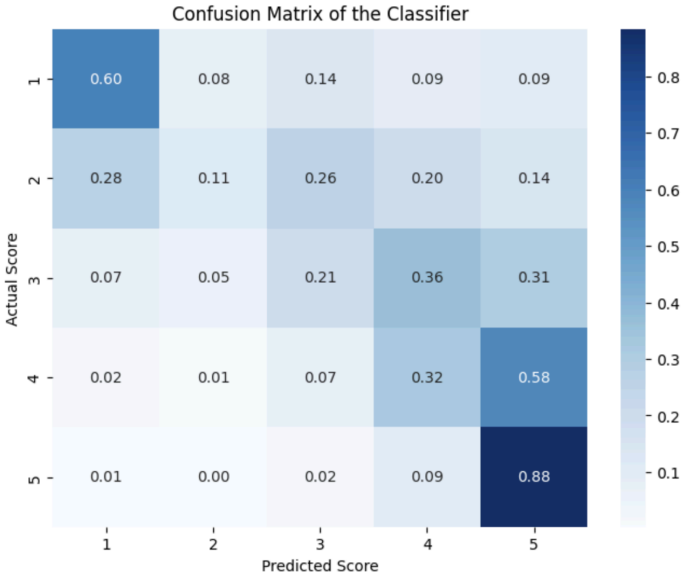
These hyperparameters were selected after experimenting with different values to balance model accuracy and training time. The final model was trained using the engineered features, and the target Score values were adjusted to fit XGBoost's classification format.

Visualizations

The bar plot below shows the distribution of Score in the training data. It shows that the dataset is imbalanced, with many more reviews having scores of 4 and 5 compared to the lower scores (1, 2, and 3). This imbalance can bias the model toward predicting higher ratings.



The confusion matrix (Heatmap) shows the performance of the model. It tended to predict extreme ratings (1 and 5) more accurately because these reviews often had stronger sentiment polarity or clearer user feedback. In contrast, middle ratings had more subtle differences, leading to frequent misclassifications between scores 2, 3, and 4.



Tools and Libraries

In addition to the original libraries provided, I used the following tools to implement the model and feature engineering

- from xgboost import XGBClassifier
    - Imported the gradient-boosting algorithm
- from textblob import TextBlob
    - Imported to analyze the sentiment value of the review text and output polarity

Conclusion

In this project, I focused on enhancing model performance through creative feature engineering and comprehensive data preprocessing. By adding features such as ReviewAge, SentimentPolarity, AverageProductScore and AverageUserScore, I was able to provide the XGBoost model with deeper insights into the data. While the model achieved an accuracy of 53.5% on Kaggle, it struggled with middle-range scores (2, 3, and 4), suggesting room for improvement in distinguishing review ratings.

Works Cited

- How I found XGBoost: https://www.educative.io/answers/classification-using-xgboost-in-python
- How I found TextBlob: https://aglowiditsolutions.com/blog/best-python-sentiment-analysis-libraries/#:~:text=Python%20libraries%20like%20NLTK%2C%20TextBlob,important%20natural%20language%20processing%20technique