Movie Reviews Rating Prediction Model

Andrew Liu

Abstract

This report provides an overview of the Machine Learning approach I implemented to predict Amazon movie review ratings. I explore various methods for handling missing data, feature engineering, TF-IDF vectorization, and a customized K-Nearest Neighbors (KNN) classifier with balanced weighting.

1 Introduction

The goal of this project is to predict the star ratings associated with user reviews in Amazon's movie review dataset. Given the text and metadata for each review, I aimed to build a model capable of identifying the likely rating based on patterns within the text, user behavior, and other review attributes. My approach primarily centers on data preprocessing, feature engineering, and developing a custom KNN model to address the imbalanced class distribution in ratings.

2 Methods

To create an effective prediction model, I approached the task in several stages, including data preprocessing, feature engineering, text vectorization, and a custom KNN classifier tailored to handle the data's imbalance, which I implemented with the help of ChatGPT (OpenAI, 2024). I also explored alternative algorithms, but the final chosen approach yielded the best balance between accuracy and computational efficiency. This section outlines the specific techniques and steps I implemented.

2.1 Data Preprocessing

Preprocessing included handling missing values and reformatting data to aid interpretability. Key steps included:

- **Handling missing values**: Missing values in helpfulness ratios were filled with zeros, and numerical divisions that could yield infinity were replaced with zeros for consistency.
- **Relative Time Feature**: The `Time` column was transformed into years since the review's publication (relative to the current year). This feature, labeled `YearsAgo`, helped capture temporal relevance, as recent reviews may carry different sentiment trends.

2.2 Feature Engineering

My feature engineering focused on extracting valuable insights from the text and metadata, which included:

- **Helpfulness Ratio and Bucket**: The helpfulness ratio (`HelpfulnessNumerator` divided by `HelpfulnessDenominator`) indicates the proportion of users who found the review useful. This was categorized into buckets (bad, medium, good) based on the set threshold of 0.7 to allow the model to capture trends across different levels of helpfulness.
- **Text and Summary Length**: The word count in the `Text` and `Summary` fields was added as `TextLength` and `SummaryLength` to identify longer or shorter review styles, which might indicate a more or less passionate review and thus higher or lower score.
- **Caps Lock Count**: This feature counts words in all caps, assuming they signify emphasized sentiment. I found that reviews with more capitalized words often express strong opinions and are more likely to have extreme ratings.
- **User Rating Habits**: To capture rating tendencies for each user, I calculated `UserAvgRating`, `UserRatingVariance`, and `UserReviewCount`. The model could

then incorporate user-specific behaviors, such as consistent high or low ratings.

2.3 TF-IDF Vectorization

To incorporate text analysis, I applied Term Frequency-Inverse Document Frequency (TF-IDF) vectorization on the review text (`Text` column). By limiting TF-IDF to the top 200 words, I captured essential phrases and terms without overfitting. This method helped quantify important language patterns in reviews while controlling for term frequency across the dataset.

2.4 Feature Scaling and Combination

The non-TF-IDF features were standardized using `StandardScaler`, ensuring they were on a comparable scale. I then concatenated these scaled features with the TF-IDF matrix to produce the final feature set. This combined set allowed the model to leverage both textual and numeric attributes effectively.

2.5 Balanced Weighted KNN Model

The model used was a custom KNN classifier with both distance weighting and class weighting. My custom implementation aimed to address the imbalanced distribution of rating classes (e.g., more 5-star ratings than 1-star ratings). The key elements of this model include:

- **Distance Weighting**: Neighbor contributions are inversely proportional to their distance from the target, so closer neighbors influence the prediction more heavily.
- **Class Weighting and Alpha Tuning**: A custom weight adjustment per class was incorporated using `alpha`, which scales the influence of each class. By tuning `alpha`, I controlled the model's sensitivity to minority classes.

This model modification aimed to balance the influence of frequent classes (5 star ratings) while giving minority classes (1−4) a fairer representation.

2.6 Hyperparameter Tuning

Initially, I implemented Stratified K-Fold Cross-Validation to tune `n_neighbors` and `alpha` which would have preserved the class distribution in each fold, providing a balanced evaluation across rating classes. However, due to an extreme decrease in time efficiency, I instead decided to set `n_neighbors = 200` and `alpha = 0.25` and skip the tuning.

2.7 Alternative Models Attempted

During the modeling phase, I also experimented with several other algorithms, including Naive Bayes, Logistic Regression, and Random Forest. These methods were initially tested to gauge their suitability for the rating prediction task:

- **Naive Bayes**: This model, while efficient for text classification, performed poorly with multi-class rating prediction due to its strong independence assumptions, resulting in low accuracy.
- **Logistic Regression**: Although logistic regression provided some interpretability, it struggled with the multi-class imbalance and was not able to handle a large number of categorical features/variables effectively. It is also vulnerable to overfitting. (DataCamp, 2024)
- **Random Forest**: Despite its ability to handle non-linear relationships, Random Forest was computationally expensive and showed limited accuracy improvements, especially in handling rare classes within the imbalanced dataset (GeeksforGeeks, 2024).

After comparing these methods, I ultimately opted for the custom KNN model, which offered the best balance between performance and runtime efficiency.

3 Results

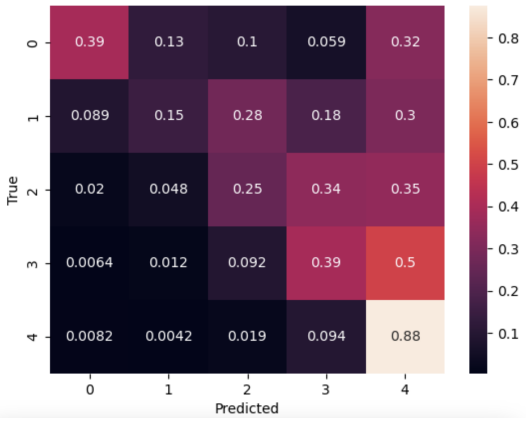

Figure 1: Confusion matrix of the final balanced-weighted KNN classifier

The final model achieved an accuracy of 61.7% on the test set when run locally and

58.3% on Kaggle. The confusion matrix (Figure 1) highlights the model's performance across classes, showing reasonable performance in predicting both the high-frequency and low-frequency ratings.

4 Conclusion

In this project, I applied a combination of data preprocessing, feature engineering, and a customized KNN approach to predict Amazon movie review ratings. By addressing imbalanced data through class weighting, leveraging user-based features, and carefully tuning the model, I was able to improve predictive accuracy.

References

DataCamp. 2024. Understanding Logistic Regression in Python. [Accessed 2024].

GeeksforGeeks. 2024. Random Forest Classifier using Scikit-learn. [Accessed 2024].

OpenAI. 2024. ChatGPT: A Language Model for Generating Human-like Text. [Accessed 2024].