CS506 Midterm Report

Clara Chung U17757445

Introduction

The goal of this project is to predict star ratings for amazon movie reviews. The provided datasets were train.csv, which consists of 1,697,533 unique reviews from Amazon Movie Reviews, and train.csv which consists of 212,192 unique reviews.

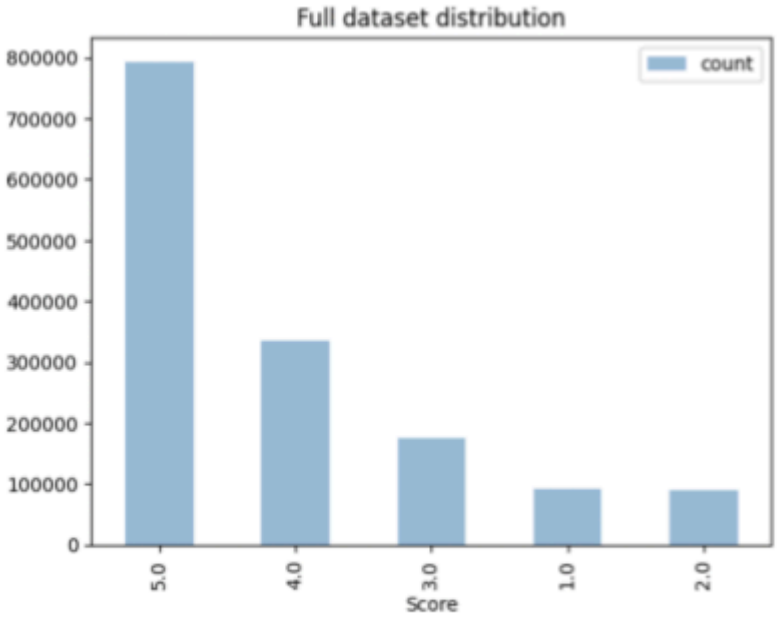
Data Processing

After loading both datasets, I removed any rows from test.csv with missing score values as these represent the target variable. This resulted in a dataset with 1,485,341 rows. To improve efficiency, I trained my model on a randomly generated subset of 10% of the original training data.

I handled missing values in ProductId and UserId columns by replacing them with placeholder values 'UnknownProduct' and 'UnknownUser' respectively. I replaced missing text and summary values similarly. I performed text cleaning by removing irrelevant punctuation and converting text to lowercase. I calculated the helpfulness ratio and handled cases where 'HelpfulnessDenominator' was zero and could cause errors. In cases where UserID was missing or null I generated a unique ID based on the index of the review.

Data Visualization

A



B

```
Full dataset class distribution:
Score
5    0.533994
4    0.225691
3    0.118547
1    0.061393
2    0.060375
Name: proportion, dtype: float64

Subset class distribution:
Score
5    0.533992
4    0.225692
3    0.118545
1    0.061393
2    0.060377
Name: proportion, dtype: float64
```

Figure 1. Data Visualization (a) Bar graph of star ratings in original training dataset. (b) Dataset distribution in full dataset and randomly generated subset of 10% of original dataset.

To understand overall patterns in the provided dataset, I began by plotting the distribution of scores in train.csv. This revealed that the dataset contains more 4 and 5 star reviews. I wanted to

ensure the subset I used for training was representative of the overall train.csv to prevent overfitting. I evaluated this by computing the class distribution for the full dataset and my subset. From this data, I made the assumption that the relative proportion of reviews in train.csv and the subset are consistent (Figure 1b).

Text-based features

Text-length and summary length were extracted to capture differences in review length. Exclamation marks, question marks, and occurrences of positive and negative words were extracted to gauge sentiment polarity. Based on the difference between positive and negative words, a 'sentiment score' was calculated. To provide a more advanced understanding of review sentiment, I implemented textblob's polarity score [CITE]. Additionally, vocabulary richness was calculated from the ratio of unique words to total words in a review, and readability was calculated with Flesch Reading Ease and Flesch-Kincaid Grade metrics Flesch Reading Ease and Flesch-Kincaid Grade metrics from textstat library [CITE]. TF-IDF

Model selection

1. *KNN*

I began by modeling with K-nearest neighbors (KNN). However, I found that this model had a high computational cost. Even though I separated a subset of my data for training the model, each prediction in KNN requires computing the distance between input instances and all training samples to find nearest neighbors. This made KNN implementation not ideal for the amount of data I was handling. Additionally, since I am working with high-dimensional data KNN distance metrics were less informative. However, the main reason I ended up not choosing KNN was my imbalanced dataset. KNN treats all classes equally which resulted in significant biases towards the majority classes (**Figure 1a**), and underrepresentation of minority classes. Even after resampling the data to balance the classes, I found KNN was still highly biased towards the majority class.

2. *RandomForest*

The next model I tested was RandomForest. RandomForest can adjust for class imbalance as it has a class weight parameter. I found that setting class weight to balanced and using Random Forest significantly improved the model's accuracy. While more accurate than KNN, RandomForest was very slow when dealing with my dataset so I decided to try implementing Hist Gradient Boosting Classifier (HGB Classifier).

3. *HGB Classifier*

HGB classifier generates histograms to bin continuous feature values before finding the best splits in the dataset. This reduces the computational complexity, making it ideal for working with larger dataset and reducing memory consumption. Additionally, HGB leverages multithreading, allowing it to utilize multiple CPU cores during training. With HGB, I was also able to specify class balance with the class weight parameter. I found that HGB slightly improved the performance of my model while being less computationally costly, so I used HGB for modeling.

Model optimization

1. *Data Imbalance*

Despite adjusting for class imbalance by specifying a balanced class weight parameter, I found that my model was still biased towards the majority class. To improve accuracy, I resampled the dataset. I tested different resampling methods, including SMOTE, SMOTENC, ADASYN, RandomUnderSampler, and SMOTEENN. I found that ADASYN resulted in the best performance so I used this for training my model. Overall, I found that resampling with ADASYN and modeling with HGB classifier was the best combination for my dataset.

2. *Hyperparameter Tuning*

To find the optimal combination of parameters for my model, I performed hyperparameter tuning. I tested a range of parameter distributions using randomized search cross-validation (RandomizedSearchCV). I chose RandomizedSearchCV because it evaluates a fixed number of parameters, improving efficiency.

Final Results

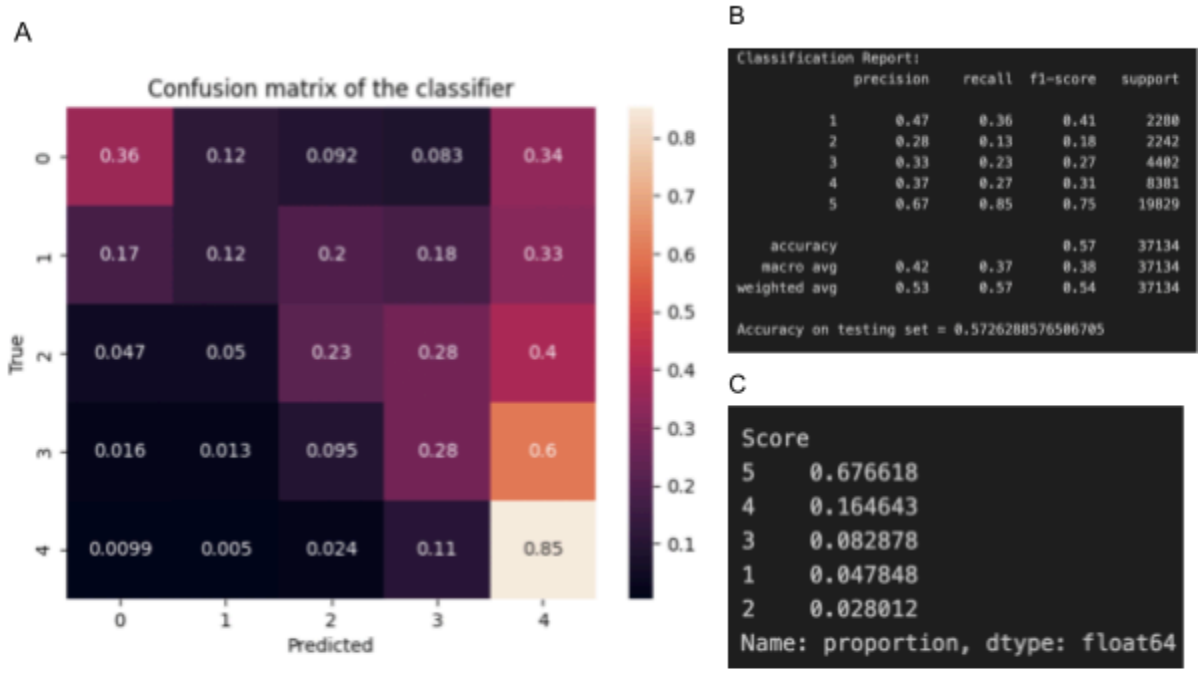


Figure 2 (a) Confusion matrix of final model after resampling with ADASYN and modeling with HGB classifier. (b) Corresponding classification report (c) Final score distribution in submission.csv

The final model had an accuracy of ~53.39% on the training set. The classification report (**Figure 2b)** shows that my model performs strongly when predicting the majority class (score 5), but is comparatively weak for minority classes and displays lower precision, recall, and F-1 scores. Similarly, the confusion matrix reveals bias towards the majority classes (**Figure 1a**). Since I spent a significant amount of time tuning the model and parameters, possible next steps include improving the extracted features. In particular, it is important to determine which features are most the most relevant predictors of score. Additionally, including more advanced features could improve the accuracy of my prediction.