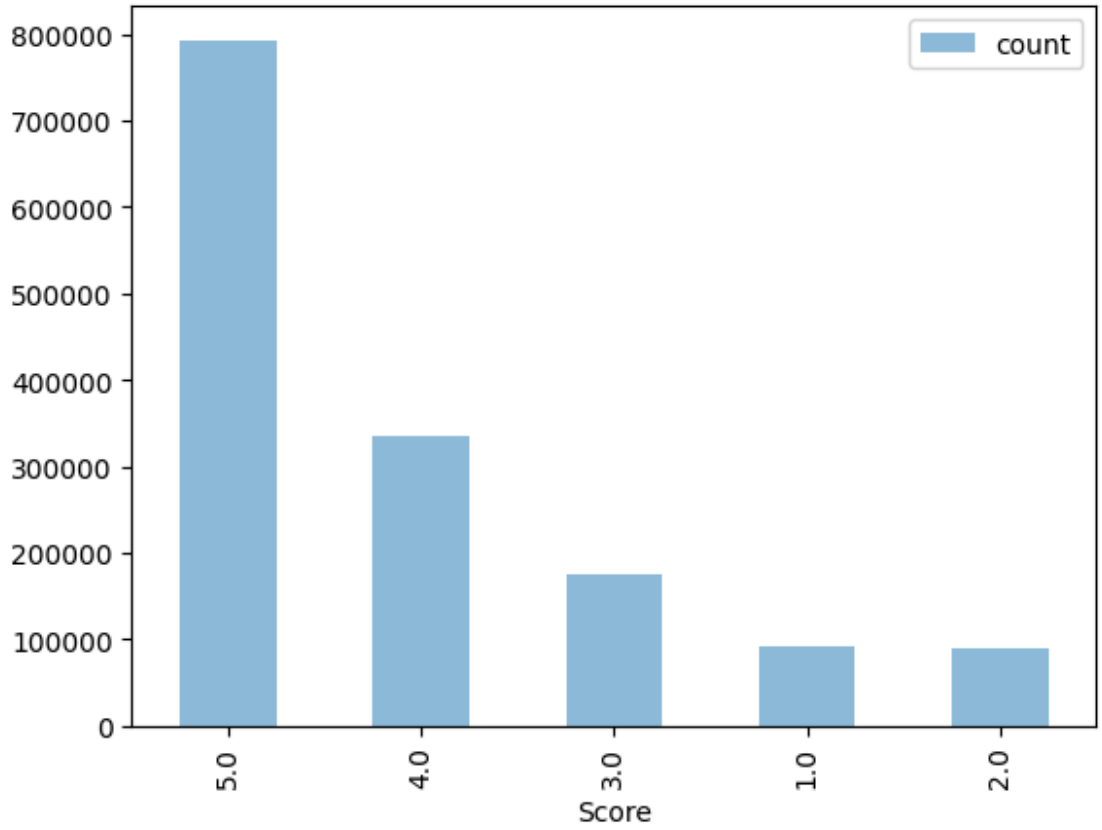
Introduction

The task given was to predict the star rating based on amazon product reviews. The dataset provided had over a million data entries and multiple data fields that included data such as Time, the review text, Helpfulness Numerators and Denominators (number of users who found the review helpful / number of users who indicated whether they found the review helpful), etc. With the given task and and the dataset provided, the challenge was to then develop a predictive model that was able to accurately classify the star ratings based on the review provided while also tackling problems such as class imbalances and high-dimensional text data.

Data Exploration and Pre-processing

We first explored the dataset provided to understand the distribution of data, which resulted in the graph we see below:

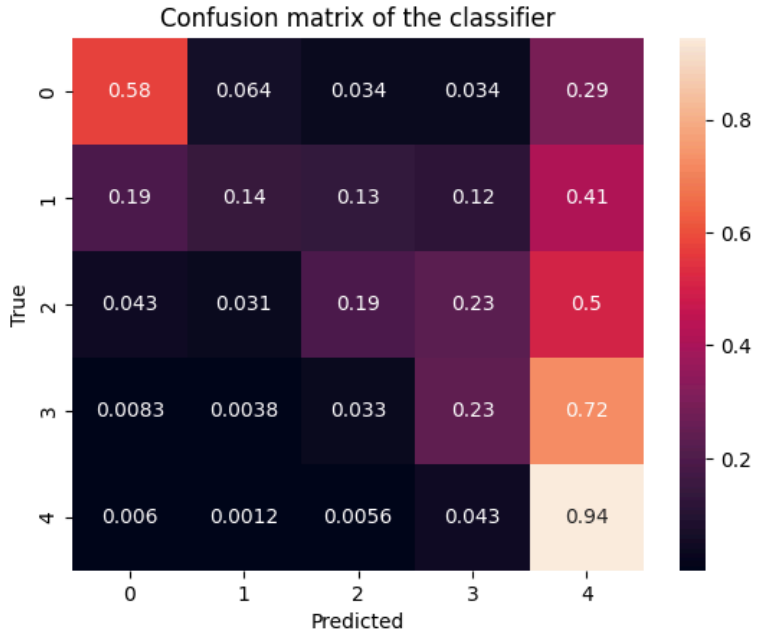


As we can see from this histogram, we can see a huge class imbalance where most reviews are skewed towards 5 star ratings. Thus, this was an issue I tried to fix but also faced challenges as will be discussed later.

For the Data preprocessing, the methods I used were based on what we learned in class for LSA (Latent Semantic Analysis), I then used the TF-IDF (Term Frequency-Inverse Document Frequency) method to help recognise patterns in the text data from reviews while also using the nltk library to get common english stop words that do not convey important information and filtering them out of the text data in the reviews. The summary and text data fields were combined into a single CombinedText field for more clear and efficient processing. I also processed the Time data field by converting it into a more readable format and splitting it into the Year and Month to represent a temporal trend that could affect star ratings. Any missing data within the data fields were also cleaned up by using the fillna method within the pandas module so as to avoid any nullpointer errors when parsing the dataset. The helpfulness feature provided was also used, and any division by zero errors caused by a zero HelpfulnessDenominator value was treated as a zero value. Then, all of the numerical data fields was scaled appropriately using the StandardScaler method so as to prevent the different scales in the different data fields from causing significant errors during the modeling and prediction. StandardScaler is a method that is commonly used in machine learning for normalizing data and is expanded upon in the scikit-learn documentation. Similarly, TF-IDF is also a common vectorization method used for text data within the scikit-learn library, and is expanded upon in their documentation.

Addressing Class Imbalance

As seen in the chart above and the confusion matrix shown below: (index goes from 0 to 4 instead of 1 to 5 due to constraints of using xgboost)



We can see that there is a significant class imbalance where most predictions are skewed towards 5 star ratings. The natural assumption is then that oversampling minority classes would balance the dataset and increase the accuracy of the model; however when trying to implement SMOTE(Synthetic Minority Oversampling Technique), I actually saw a decrease in accuracy to about 0.55. This could be due to a variety of reasons from just applying SMOTE incorrectly on my end leading to data leakage or the oversampling resulting in overfitting; but due to the increase in computation time when using SMOTE, i was unable to extensively test for the issues and in the end was unable to resolve the issue of class imbalance in my model.

Model choice

The choice of XGBoost was made because it returned the highest accuracy score out of all the models tested. Logistic regression returned with about 0.6 accuracy, Random forest returned with about 0.59 accuracy.

Special Tricks

I adjusted the score column of the dataset to range from 0 to 4 instead of 1 to 5 when modeling with XGBoost so as to fit within its constraints of indexing, More comprehensive stopwords using the nltk library was also used to provide a more accurate and improved TF-IDF vectorization. The usage of the pipeline import allowed me to combine and streamline all of the preprocessing and modeling.

Works Cited

"6.3. Preprocessing Data." *Scikit-Learn*, 2022,

scikit-learn.org/1.5/modules/preprocessing.html.