Anneke van Oosterom

Midterm Report

The first step of building the model that produced my best accuracy on Kaggle was data exploration. Data exploration on the train dataset revealed a disparity in score distribution among the 1,697,533 unique Amazon reviews. Five-star ratings dominated the data, while one and two-stars were the least represented. Along with this, I found that throughout the months, the distribution of scores would change as well, with fewer reviews in the fall compared to the spring. Additionally, certain scores were associated with longer or shorter reviews. Finally, I also read through about 100 of the first rows of data scanning to see if I could pick up on any precursory patterns, noting that the use of special characters, grammar, and style of writing could really vary but would sometimes clue into what a score would be.

After doing this exploration, I invested a significant amount of time to add features to the dataset. Due to the text-heavy nature of the dataset and KNN's requirement of numeric inputs, transforming the summary and text variables was crucial. The first features I added were length and word counts for summary and text. These were not only their own columns but were also used to help build other features.

I also added features, including sentiment scores, readability, and special character counts, each grounded in my observations about how the text might reflect a user's rating tendencies. I added text and summary sentiment because just as a negative review is likely to be associated with a lower score, a positive review is likely to be associated with a higher score. This variable is scaled from -1 to 1, where -1 is very negative, and 1 is very positive. I also assigned features for text and summary readability by using the Flesch Reading Ease score, with higher scores indicating the text is easier to read. This was an interesting addition founded on the idea that someone who is more upset may take less care in writing a review. A more positive review with a higher score may spend more time checking grammar or spelling.

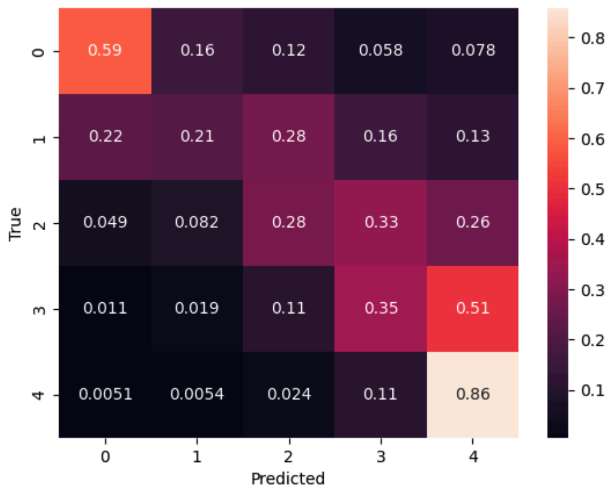
Next, I added a summary length ratio variable that compares the length of the summary to the length of the text of a review. If this variable is higher, it suggests that the summary was quite long, which may indicate that the main text covered multiple points or topics. This may help in classifying the extreme reviews where people are particularly happy or dissatisfied with the product, as they could have more points to cover. Another variable I added was a helpfulness-to-length ratio, which highlighted short but effective reviews. This could capture intermediate scores where users felt relatively neutral and were concise about product quality. Additionally, I further distilled the content of the text to see how certain features, such as special characters, question marks, and exclamation points, impacted scoring. This was based on the idea that a user may use special characters such as exclamation points more frequently in reviews with a higher score and question marks in reviews with a lower score.

Alongside the other numeric features mentioned were those pulled from non-text columns in the dataset. I added a feature called user review count that helps identify if someone leaves many reviews for different products. This could aid in finding bots or spam reviewers who would give similar scores for all of the products they review. In addition to this, I added a user average score, which provides the average score given by each user. Special care was taken here to ensure that the model did not learn on scores it would not have access to in a real application or on the Kaggle dataset. This same consideration was applied to another feature, product average score, that found the average score of each product. A final variable I added looked at user ID frequency and the time since their last review, which used the time in seconds between a user's current review and their previous review (based on the Time column). This variable provided information about whether a user would post a lot of reviews at once or do it more sporadically, potentially also checking for bots or spam users.

Another group of features I added used the helpfulness columns. First, I defined both helpfulness and unhelpfulness. Helpfulness was defined as the helpfulness numerator divided by the helpfulness denominator, which represented the ratio of people who found the review helpful. Its score ranges from 0 to 1 and likely helps assess if a review is genuine or accurately represents the product. The unhelpfulness feature was defined as the helpfulness numerator minus the helpfulness denominator. An unhelpful review may be unhelpful because it is biased and has an extreme score. Finally, I also contrasted helpfulness and unhelpfulness to find reviews that were based on very strong opinions, which could indicate an extreme score.

After feature engineering, I put the data through several preprocessing steps. I imputed missing values in numeric features using the mean of all scores, and I replaced text-based missing values with an empty string. Next, I scaled numeric features with the standard scaler to normalize their range. Due to the nature of text data in the summary and text columns, I further processed them with Term Frequency-Inverse Document Frequency (TF-IDF) to generate weighted features for each review. This transformation highlights words that are uniquely significant within each review as compared to the overall dataset, helping to differentiate higher or lower-scoring reviews. To reduce the dimensionality TF-IDF produces, I employed Singular Value Decomposition (SVD), which reduces the number of features and ensures that the most valuable information is retained. This reduction helped ensure the model could process the data efficiently without sacrificing predictive power.

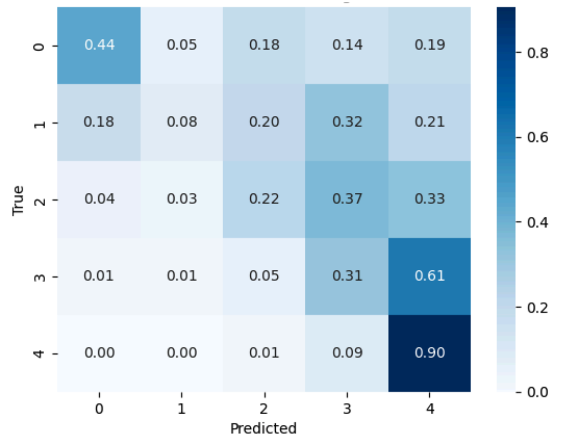
I fit all of this information to a KNN model with 20 nearest neighbors using a TF-IDF of 50 with an SVD of 10. During testing, this was found to be the best way to predict the class while not leading to too much overfitting. This model, with all of the features alongside the aforementioned preprocessing steps, yielded the best accuracy with a local score of 0.62 and a score of 0.603 on Kaggle (Figure 1).



```
Accuracy on testing set =  0.6210817157506948
```

Figure 1. Confusion Matrix of 0.603 submission on Kaggle

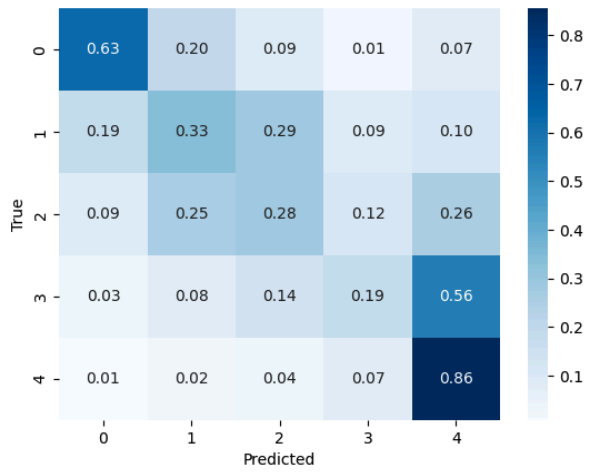
I did other testing on subsetted training data to see if fitting additional models or techniques would improve accuracy. First, I tried to see if outliers were the cause of the lower accuracy, so I removed them from the model, but this again only brought accuracy down. Despite considerable testing with Random Forest, XGBoost, and voting classifiers, the accuracy did not improve (Figure 2). Additionally, I employed reverse feature elimination to see if the lower accuracy was because of noise from redundant features. While this did select fewer than all of the features available, it did not improve the local accuracy of the model (Figure 2).



```
Accuracy on testing set = 0.6116
```

Figure 2. Confusion Matrix of Voting Ensemble Model with Reverse Feature Elimination

One trend I noticed was that the models were always much better at predicting scores of 5. I thought since this was likely due to the unequal distribution of scores, I would try to resample or generate synthetic examples of underrepresented classes. My attempts to balance score distributions using SMOTE by creating synthetic examples of undersampled scores did not raise accuracy (Figure 3). Unfortunately, constraints in time and computational resources prevented further testing with larger TF-IDF and SVD dimensions, though this remains a potential area for future improvement.



```
Accuracy on testing set =  0.5912
```

```
Original class distribution: Counter({5.0: 4000, 4.0: 1680, 3.0: 890, 1.0: 480, 2.0: 450})
SMOTE sampling strategy for undersampled classes: {1.0: 2000, 4.0: 2000, 3.0: 2000, 2.0: 2000}
```

Figure 3. Confusion Matrix of Voting Classifier Model with Smote

Overall this model achieved an accuracy of 0.603 on the Kaggle private leaderboard. After listening to the top six on the leaderboard in class, it seems that adding a logistic regression model may have aided the accuracy. Along with this, XGBoost or Random Forest alone may have out performed KNN. However, many of the leaders had similar feature selection and engineering which was validating.