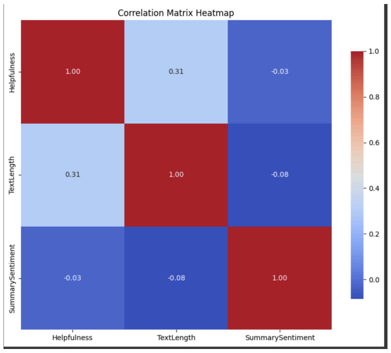
By Bryce HIraoka

Midterm

Process

1. Initial Data Exploration and Analysis

- I first ran the starter code to understand the distribution of ratings, the nature of each feature, and potential correlations within the data. Visualization tools like histograms, scatter plots, and correlation matrices provided a clearer view of feature interactions and distributions.
- Analysis also showed a tendency toward class imbalance, with certain ratings (e.g., 5-stars) dominating. I originally thought that balancing the training set would be beneficial but after banalcing the training set my accuracy score decreased.



2. Testing and Refining Features

- I experimented with a variety of features derived from the review data, focusing on aspects that could reflect user sentiment and engagement. Features like *Helpfulness*, *TextLength*, and *SummarySentiment* emerged from initial trials as consistently valuable, while others (e.g., separating *Helpfulness Numerator* and *Denominator*) did not perform as well, showing limited predictive power.
- The *SummarySentiment* feature, calculated using TextBlob, proved useful for gauging the overall tone of reviews, while *TextLength* provided insight into the depth of a review, often correlating with stronger sentiments. Testing and refining these features iteratively ensured that only the most predictive elements were included.

3. Model Selection and Experimentation

- I initially experimented with a range of models, including K-Nearest Neighbors, Logistic Regression, Random Forest, and XGBoost, each chosen for specific characteristics:
    - **Logistic Regression**: Explored for its interpretability and effectiveness in binary and multiclass classification. While logistic regression showed potential, it faced convergence issues that required tuning.
    - **Random Forest and XGBoost**: These models were tested for their capacity to handle complex relationships and capture non-linear interactions in the data. However, they proved resource-intensive and tended toward overfitting, limiting their viability in the current setup.
    - **Stacking**: I considered a stacked model approach to leverage the strengths of multiple algorithms. However, due to computational constraints, implementing an ensemble model was challenging.
- Each model was tested through cross-validation, with performance evaluated based on accuracy, precision, and recall. However some were unable to run because of computer issues

5. Conclusion and Reflection

- In the end my code was a blur and I could not revert my more complex code to a working state. With little time left I decided to start from scratch and use my most successful features from past experimentation (excluding text sentiment analysis)

Features:

---

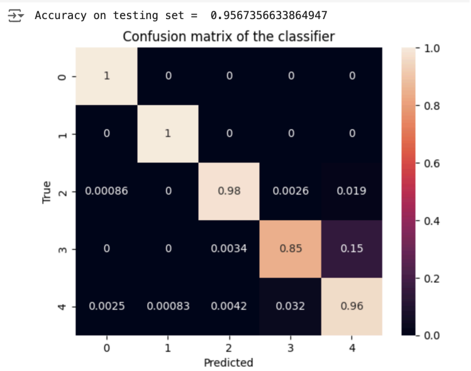
1. **Helpfulness**
   - This feature was calculated by dividing the *Helpfulness Numerator* by the *Helpfulness Denominator*, providing a measure of the proportion of users who found the review helpful. Initially, I tested other forms of the helpfulness statistic, such as using the numerator and denominator as separate features. However, analysis revealed that the ratio between them was more predictive, as it directly reflects the collective judgment of other users.
   - **Rationale**: Reviews with high helpfulness scores may offer valuable insights or well-articulated opinions, which are often associated with extreme ratings (1-star or 5-star). Detailed and helpful reviews tend to polarize, leading to clearer positive or negative sentiment. This feature also emphasizes user engagement, giving more weight to reviews that are marked by others as informative.
2. **TextLength**
   - This feature measures the character length of each review's text, capturing the detail and potential thoroughness of the review.
   - **Rationale**: Review length can be indicative of the sentiment intensity. For example, highly enthusiastic or critical reviews might be either succinct and to-the-point or comprehensive and detailed, depending on the user's experience. High ratings may often correlate with short, enthusiastic comments (e.g., "Loved it!"), while negative reviews tend to be longer as they delve into specific complaints. This assumption helped streamline feature selection by emphasizing that text length could capture the strength of sentiment.
3. **SummarySentiment**
   - To capture the sentiment of each review concisely, I computed the sentiment polarity of the *Summary* field using TextBlob, with polarity values ranging from -1 (very negative) to 1 (very positive).
   - **Rationale**: The summary sentiment provides a quick snapshot of the reviewer's general impression, with higher sentiment scores typically indicating positive ratings and lower scores reflecting negative ratings. This feature offers a straightforward indication of the review's tone, making it particularly valuable in the context of predicting star ratings.

These features together provide a balanced approach, combining quantitative measures (such as *Helpfulness* and *TextLength*) with qualitative insights (e.g., *SummarySentiment*) to better approximate the target ratings. This approach was designed to prioritize features that reflect engagement, depth, and sentiment to improve model performance.

Shortcomings & Challenges

Throughout this project, I encountered several technical and practical challenges that impacted the model's development and performance. These included limitations in computational resources, and issues with overfitting. Below is an overview of these challenges and how they affected my approach.

1. **Sentiment Analysis Limitations**
   - I originally wanted to apply sentiment analysis on the text data however before it would finish my laptop would run out of RAM. Although I was able to use TextBlob for summary sentiment, analyzing the full text sentiment could have provided additional valuable insights.
2. **Overfitting with Advanced Models**
   - Attempts to implement more complex models like XGBoost and Random Forest encountered issues with overfitting. While these models are powerful, overfitting led to inflated accuracy on the training set and poorer performance on unseen data. I was able to achieve 95% accuracy on the testing set but this clearly wasn't realistic.



```
Accuracy on testing set =  0.9567356633864947
```

Confusion matrix of the classifier

True \ Predicted	0	1	2	3	4
0	1	0	0	0	0
1	0	1	0	0	0
2	0.00086	0	0.98	0.0026	0.019
3	0	0	0.0034	0.85	0.15
4	0.0025	0.00083	0.0042	0.032	0.96

3. **Computational Limitations**
   - Due to limitations in my laptop, running resource-intensive models like Random Forest, XGBoost, and Stacking proved challenging. These models could not be executed efficiently, which limited my ability to experiment with more complex ensembles or fine-tune them. This constraint led to a focus on simpler models, like K-Nearest Neighbors, which, while effective, may not fully exploit the data's predictive power.
4. **Logistic Regression, Random Forest, XGBoost, TF-IDF Vectorization**
   - Although these techniques were considered, their implementation presented technical challenges. Logistic Regression initially faced convergence issues, while Random Forest, XGBoost, TF-IDF would crash my google colab.

5. **Stacked Model Implementation Issues**
   - I explored the possibility of a StackingClassifier to combine the strengths of multiple algorithms. However, this led to overfitting or the crashing of my computer.
6. **Time Constraints**

- Along with the computing power issue I face a self imposed issue of starting the competition later than I should have. I acknowledge that this is a big reason in my projects shortcomings.