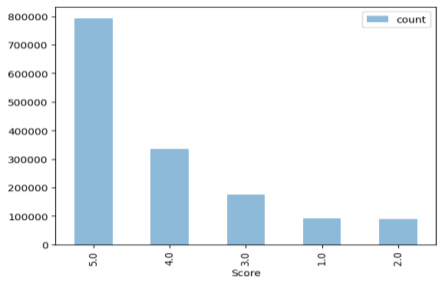
Logistic Regression for Amazon Movie Review Star Prediction

Data Preprocessing:

We used train.csv to extract the reviews and metadata, while test.csv was used for evaluation. We can see that the train.csv contains 1,697,533 reviews with metadata and associated ratings (score). And also, the test.csv contains 212,192 reviews but without the Score column. Our task is to predict these missing scores based on the features available. By checking the data shape, we can get training set has 1697533 rows and 9 columns, meanwhile test set has 212,192 rows and 2 columns. In the descriptive statistics, we can see that Helpfulness Numerator and Helpfulness Denominator both have many zero values, which suggests many reviews were not rated for helpfulness). The average score is around 4.19, which suggests a bias towards positive reviews. The timestamp (Time) ranges from older to more recent reviews, with a minimum value of around 950,745,600 (which corresponds to 2000) and a maximum of around 1,406,074,000 (which corresponds to 2014). The dataset is biased towards positive ratings, with 4 and 5 stars being more frequent.

To account for the uneven frequency of the classes, we tried to limit the number of entries by their helpfulness. We tried values of 0.5, 0.7, 0.8, etc but the reduced dataset never helped to increase the overall model testing accuracy. So in the end (best runs), we trained the model on features extracted from the entire dataset provided.



Bar Plot of Star Rating Distribution

Adding Features:

The feature engineering process involves a mix of numerical transformations (like scaling and ratios) and textual preprocessing (like lemmatization and stopword removal). These carefully selected features align with the goal of improving model performance while maintaining generalizability.

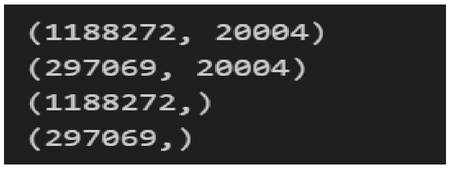
In feature engineering, we use the Helpfulness Ratio to indicate how impactful or useful the review is to other users. And for cases with a denominator of 0 (indicating no user-provided feedback), the ratio is filled with 0. We use the timestamp to extract year, considering that the review's timestamp (in Unix epoch format) can capture potential temporal patterns in user behavior. The year feature helps capture seasonal or time-based trends in the data. Reviews without content could provide noise in the data. It is essential to handle these cases to ensure that all fields are usable. The length of the review or summary can indicate the level of detail in the feedback. Generally, more detailed reviews (longer text) may correlate with higher or lower ratings. We concatenate product id, review, and summary text into a single field ensuring that both sources contribute to the model's predictive ability. By extracting these features, we create a rich feature set that captures multiple aspects of the data (text, time, usefulness, etc.), ensuring the model has enough information to make accurate predictions.

Sample + Split into training and testing set:

This segment handles the text vectorization using the TF-IDF approach, merges the text features with the additional numerical features, and then splits the data into training and test sets for model validation.

We use TF-IDF (Term Frequency-Inverse Document Frequency) to transform the review text into numerical features by measuring how frequently words appear in the text, scaled by how unique those words are across the dataset. The text features (TF-IDF vectors) and numerical features (like Helpfulness, Year, LenText, etc.) are merged to create a comprehensive feature set for the model. And then after merging text and numerical features, we split the data into training and test sets with the ratio of 80%:20% to evaluate the model's performance. The number of n-grams extracted was played around with. Higher number of n-grams allowed for improved model accuracy at the cost of computation time. In total there were roughly 300,000 size one and size two ngrams in the entire dataset and we used only the top 10000 or top 5000 for the majority of our runs.

This process ensures that the model receives a comprehensive and well-prepared feature set, maximizing its ability to predict star ratings accurately.



```
(1188272, 20004)
(297069, 20004)
(1188272,)
(297069,)
```

Result analysis: Training Feature Set (X_train_split): 1,188,272 samples and 5,004 features. Validation Feature Set (X_test_split): 297,069 samples and 5,004 features. Training Labels (Y_train_split): 1,188,272 target values. Validation Labels (Y_test_split): 297,069 target values.

Model Creation:

By tracking the time taken, we assess computational efficiency, especially when comparing different models.

In the beginning, we considered using a Support Vector Machine (SVM) as well as Naïve Bayes method as model. I started out testing SVM using the sklearn library LinearSVC. I ran the data on just the textual data and was able to achieve a training accuracy of 60-65%. However, upon submission to Kaggle, the accuracy dropped to 48% and I decided to try another method. While using LinearSVC, I also tested out SVC but that was even slower and used too much memory for my computer to handle if I included too many features. LinearSVC was faster than SVC but still used up a lot of memory as well.

Naive Bayes was another candidate model that had potential to be well-suited for text classification tasks. However, after a couple trials, it's learning was too poor and logistic regression was preferred due to its ability to handle continuous features and regularization more effectively.

We chose logistic regression as a model, considering that it outputs probabilities for each class, which makes it useful for classification tasks. Balanced class handling ensures that the model adjusts for imbalanced classes by assigning higher weights to underrepresented ratings. We included the parameter "class_weights='balanced'" in our initial models to account for the skew in the dataset. However, after the discovery that not penalizing class frequency led to a 5% increase in test accuracy, this parameter was dropped. We set the max iteration as 1000 to ensure that the model has enough iterations to converge on larger datasets.

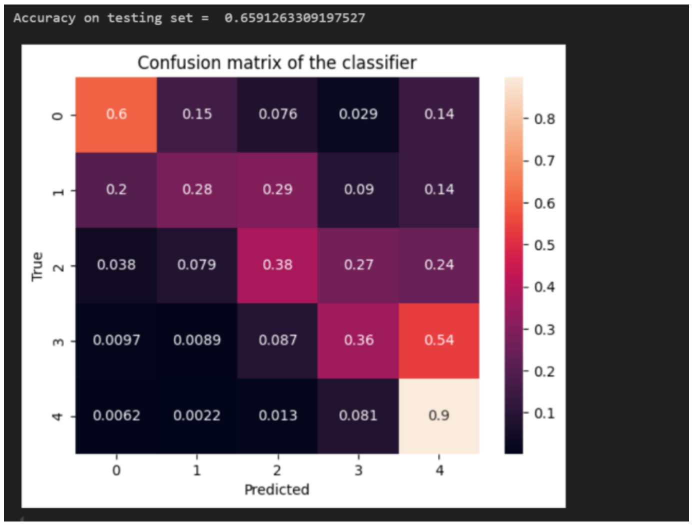
The best model was trained using the entire training set, without balanced class weights to handle the imbalance of score ratings. Training time was tracked to assess computational efficiency/memory useage. After training, the model was used to predict score ratings for the validation set.

Model Evaluation:

We evaluated using the accuracy metric and how a confusion matrix is plotted to gain deeper insights into the model's predictions.

For the accuracy metric, it measures the proportion of correct predictions (i.e., the number of times the predicted rating matches the true rating). Considering that accuracy could be misleading when the data is imbalanced (e.g. if most reviews have 4 or 5 stars). Then additional metrics – confusion matrix is also evaluated.

The confusion matrix compares the true labels with the predicted labels to show where the model performs well and where it struggles. The diagonal cells show the percentage of times the model correctly predicted each star rating. A perfect classifier would have 1s on the diagonal (all predictions correct) and 0s elsewhere. A heatmap visualizes the confusion matrix, making it easier to identify patterns and errors.



```
Accuracy on testing set =  0.6591263309197527
```

Accuracy and Confusion Matrix

Result Analysis:

Confusion Matrix: Class 0 (True rating = 0): 60% of 0-star ratings were predicted correctly. Class 4 (True rating = 4): 90% of 4-star ratings were predicted correctly. Class 1 predicted as Class 0: 20% of 1-star reviews were misclassified as 0-star. Class 3 predicted as Class 2: 8% of 3-star ratings were predicted as 2-star. Class 2 predicted as Class 3: 27% of 2-star ratings were confused with 3-star ratings.

The model performs better on high ratings (Class 4) with 90% accuracy, indicating that the model can effectively capture patterns for these ratings. Only 38% of 2-star ratings were predicted correctly, with 27% being misclassified as 3-star ratings. Similarly, 54% of 3-star ratings were misclassified as 4 stars. There is noticeable confusion between adjacent classes (e.g., 2 vs. 3, 3 vs. 4). This indicates that the model struggles with subtle differences between adjacent star ratings. The accuracy also reflects the influence of class imbalance, especially if some ratings (like 4 or 5 stars) are more common than others. The model might be biased towards predicting these common ratings, resulting in more correct predictions for them.