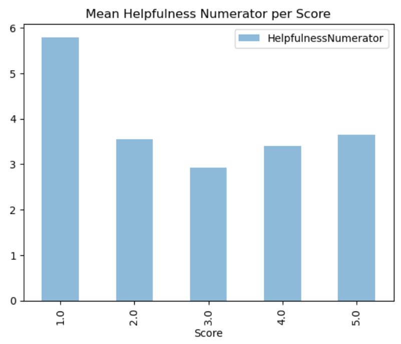
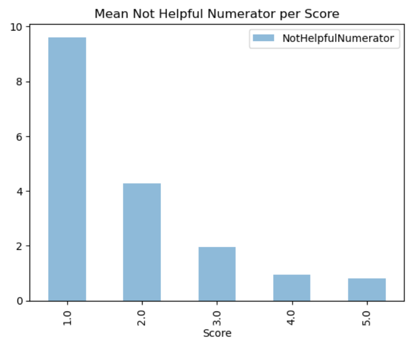
Kaggle Username: emmaleegomez

Introduction:

This model aims to predict Amazon Movie Review scores by analyzing 122,283 reviews with their associated star rating scores and features. These features include unique identifiers for the product and user, how many people found the review helpful/unhelpful, a brief summary, and the review text itself. The main problems encountered included the conversion of textual and categorical data into numerical data suitable for classification and the skew of 5 star reviews causing a majority class bias. To address these problems, features were vectorized and normalized into SVC and Logistic Regression. Additionally, a voting classifier was integrated to increase the model's accuracy.

Analysis and Exploration





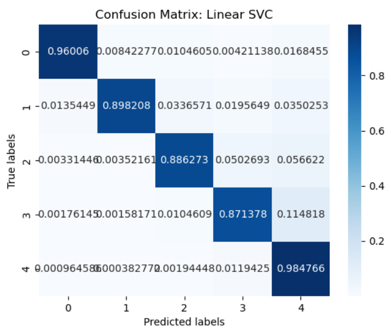
Anticipating that the summary and text data would be central to the model, I looked for other supplemental features. My objective was to identify significant variations between scores. As such, I opted to include the unique product/user identifier and the numerator representing the helpfulness/unhelpfulness of the review. I decided to use unique identifiers because higher-rated movies increase the likelihood of higher individual ratings, and potential user bias. For the numerators, the figures above show that scores of 1 tend to generate more negative and positive feedback. The second figure is especially informative because it illustrates a declining trend in unhelpful feedback vs score increase. I also explored the possibility of extracting features related to users expliciting stating the number of stars in their reviews; however, these proved to be too limited in scope and unreliable for practical use.

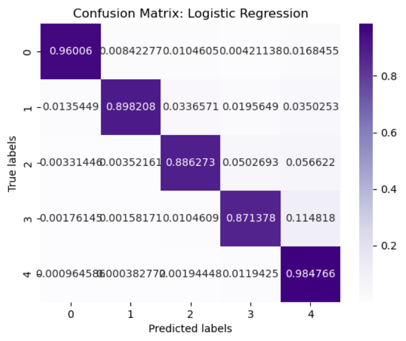
Feature Processing

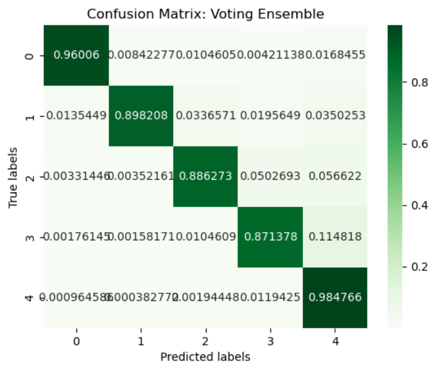
First, a One-Hot Encoder is used to convert categorical data such as 'ProductId' and 'UserId' into numerical data. Next, a standard scaler is applied to normalize the numerical features. Prior to vectorizing the text data, some preprocessing occurred including removing any special characters, turning all the text lower case, and removing stop words. Notably, the removal of only basic stop words improved the performance. Then the processed text was put into a TfidVectorizer. Several resources online like the *International Conference on Computer and*

Informatics Engineering claim that TFIDFVectorizer + SVM algorithms make the best pairing for Sentiment Analysis. This makes sense as tf-idf is a product of term frequency and inverse document frequency; it rewards words that are used often in a particular review and penalizes words that appear often in all reviews. As such, terms like "great" will have a relatively high tf-idf score when compared to other terms. Finally, I concatenated the processed summary text, review text, categorical, and numerical categories together to input into the classifiers.

Creating Models







Linear SVC:

After exploring various SVC kernel functions, I chose a linear kernel because text data often exhibits linear separability/ has a lot of features. Also, its quick runtime and simple optimization is ideal for trying different combinations of features. I employed a LinearSVC with the parameters C=0.2 and dual=False. Dual was set to false because the number of samples is much larger than the number of features. The C parameter was fine-tuned to control the regulation strength, finding a balance that minimized the RMSE of the testing set. On the validation data, it performed fairly well with an RMSE of about 0.80.

Logistic Regression:

Subsequently, I decided to test the efficacy of another linear classifier: logistic regression. This model also yielded promising results, with an RMSE of about 0.77. The only model parameter I changed was the max_iterations because without it, the model would not converge. Logistic regression excels at binary classification which suits sentiment analysis because it can differentiate reviews that are positive, neutral, and negative. Furthermore, logistic regression can effectively accommodate a broad range of features. Interestingly, when I applied logistic regression to solely the vectorized text, the results were not great due to overfitting. The inclusion of categorical and numerical features led to a substantial improvement in the classifier's performance.

Other Models:

As a third classifier for the ensemble, I was planning to use Naive Bayes, specifically the MultinomialNB classifier, which applies Bayes' theorem under the naive assumption of feature independence. To avoid potential overfitting, I created a custom class that favored sentiment

scores in the range of 2 to 4. Although this model did not perform as well as the previously mentioned ones, it yielded an RMSE of approximately 1.13. However, since it used different vectorized pieces of data to fit from SVC and Logistic Regression, I ended up not using it.
I experimented with various other classifiers, but the majority of them exhibited a significant bias towards the majority class. For instance, the Random Forest classifier showed initial promise but ultimately overfit the training data and performed poorly on the validation set, even after adjusting hyperparameters such as n_estimators and max_depth.

Voting Classifier
In order to enhance the accuracy of the model, an integration of a voting classifier was employed. When combining predictions from multiple base models, I got more accurate predictions compared to using any single model alone. The RMSE was notably reduced to 0.73, a clear indicator of the efficacy of this approach. As for the parameters, I needed to change voting to “soft” because there were only two base models. Tie-breaking decisions needed to be made on the confidence levels of these models in their respective predictions. This decision introduced a notable challenge, as my initial choice of a LinearSVC model lacked the "probability_proba" functionality required for soft voting. As such, I had to change my model to SVC with a linear kernel, which took much longer to run.

Validating Models

To validate my models, I split the train data 90-10 train to validation. Then, I ran the classifiers on the validation data + entire data and adjusted the parameters to minimize the RMSE. I created normalized confusion matrices to visualize the predicted vs actual scores. If the scores were concentrated around 5s, independent from the actual scores, I knew the model was biased and needed to add more variance by introducing new features or using a model that adds more variances. If the scores were too accurate on the entire dataset, I knew the model had overfit and I had to either adjust the parameters or use a new model. In the end, I fit my tuned models to the entire training dataset.