Enhancing Textual Data Analysis for Review Predictions without Neural Networks

In this project, my main goal was to build a model to analyze textual data of reviews from customers and make accurate predictions based on review ratings. To achieve this, I experimented with several models, including Random Forest, Naive Bayes, KNN, and Logistic Regression, along with various feature engineering techniques that I learned in my previous machine learning classes, which ultimately led to an accuracy of 62%. I initially identified two text columns: Text and Summary. To capture key terms effectively, I used TF-IDF. However, my first attempt to vectorize the text without filtering stop words generated thousands of features and overwhelmed my computer, causing it to crash. I realized that the most computationally expensive part was the model having to process all the features that I created, because it would freeze everytime I train the model. By incorporating stop words in TF-IDF, I was able to reduce unnecessary noise as my computer stopped freezing up and was able to give me an accuracy score, in which it improved from 49% to 59%, highlighting the crucial role of processing text data in the training set. Initially, I expected that using only Time and Helpfulness features would be sufficient to achieve around 68%, but this project showed me just how challenging it can be to train a model that accurately distinguishes the slight variance of classes of ratings.

An issue I often ran into when using TF-IDF was that my computer runs out of memory. So to improve memory efficiency and reduce processing time, I applied a few techniques that proved effective. One major change I had already mentioned above was adding stop words to TF-IDF, which cleaned up the text data significantly by removing words that contributed little meaning to the ratings. This trick not only saved memory but also reduced the runtime to 30 minutes, enabling me to avoid crashes and maintain a workable dataset size. When it came to model selection, I tested multiple classifiers that I learned from previous ML classes, including KNN, Logistic Regression, Random Forest, Naive Bayes, and SVC, using a consistent set of features for each. Through trial and error, I noticed that LinearSVC achieved the highest accuracy, despite other models (KNN, Naive Bayes, RandomForest) yielding slightly lower accuracy but comparable results. To optimize the LinearSVC model, I used GridSearchCV to explore different hyperparameter settings, splitting the training data and defining a grid for the regularization parameter C and loss function. The cross-validation approach ensured that my model was robust and not overfitting to the training data. After fitting the model with the best hyperparameters, I evaluated its performance on the test set, achieving an accuracy score that reflected the effectiveness of this hyperparameter tuning process.

A recurring challenge in the project was handling the imbalance in the dataset, as most reviews had a rating of 5. Initially, I attempted to address this imbalance with SMOTE in combination with RandomForest. However, I quickly discovered that applying SMOTE on high-dimensional data could lead to overfitting, especially since my model couldn't reliably generate synthetic samples for sparse data like TF-IDF vectors. I also attempted to speed up processing by reducing the number of rows of the training data file. However, this change impacted the format of the submission file, which relied on the original training set, preventing me from submitting it to Kaggle. To address the imbalanced dataset, I used `class_weight='balanced'` in the LinearSVC model to give each class an equal influence on the model's training process. This technique is particularly valuable because it addressed the heavily imbalanced data and paid more attention to minority classes of ratings to mitigate bias toward the majority class of rating 5's. This method is better than downsampling, which can inadvertently lead to the loss of valuable information. By maintaining all the data, I ensured that the model had access to a wider range of examples, allowing it to learn better from the available patterns in the data.

The most important thing I learned from this project was prioritizing clarity over complexity, and feature preprocessing over model selecting. Initially, I did not expect product ID or user ID to be important, thinking instead that the type of model would have a greater impact on accuracy. However, as I developed my model, I recognized the importance of testing the model with additional feature columns beyond just Helpfulness, Time, and TF-IDF. One significant enhancement I made was incorporating columns for the average and standard deviation of user score and average product score, which helped capture trends in the ratings for each user and product. These feature values were derived exclusively from the training set, ensuring that no information from the test set was included, preventing any data leakage where the model might access future data. Additionally, I calculated the mode of the scores for each product and user and determined the proportion of times the mode occurred relative to the total number of reviews. Again, these mode values and proportions were generated solely within the training set, maintaining the integrity of the test set. The addition of these features resulted in a 2% improvement in accuracy, bringing the accuracy of 59% up to 61%. On the other hand, I noticed that by adding the feature columns for DayofWeek – by one hot encoding the days of the week into binary values and then converting those binaries into numbers – resulted in a decrease in model accuracy because it may have introduced unnecessary complexity into my model, leading to overfitting.

Needless to say, this project’s tight deadline, combined with processing limitations, emphasized the need for practical decisions around feature engineering and efficient algorithm design. Although challenging, this experience solidified my understanding of ML workflows and showed me the importance of balancing computational resources with meaningful feature patterns. By focusing on high-quality features and adapting to system constraints, I was able to develop a model that performed reliably within the constraints of the project.