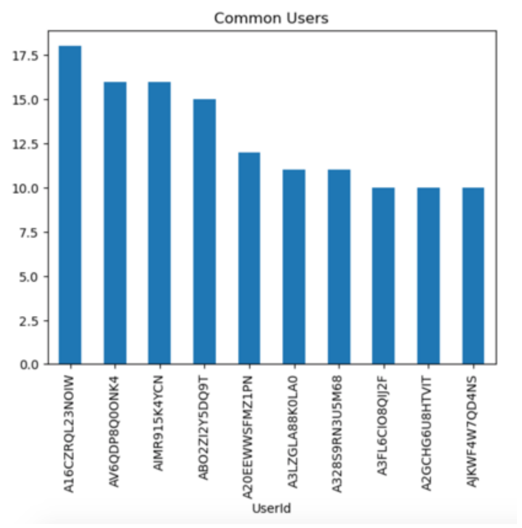
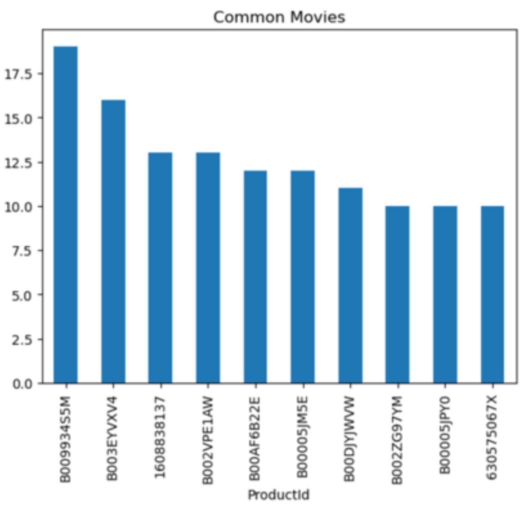
Amazon Reviews Report

Temima Muskin

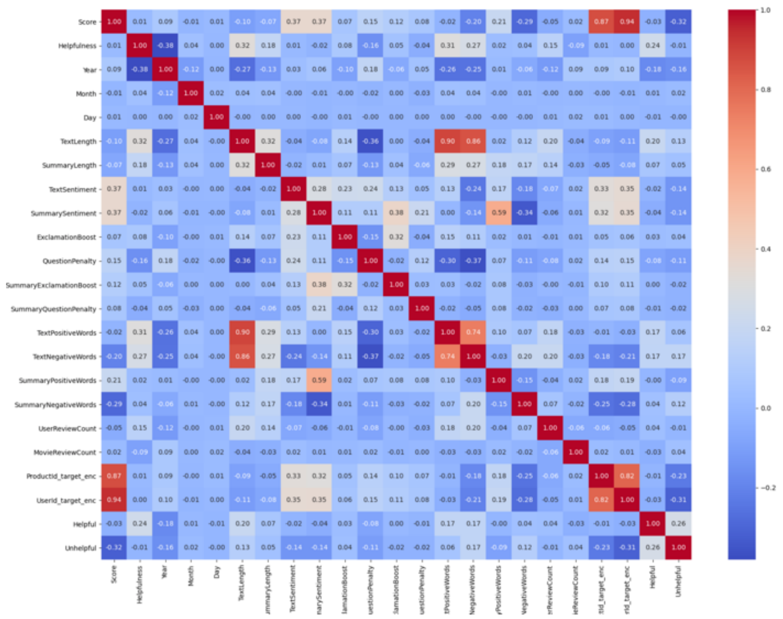
Introduction: In this project we were tasked with predicting the star rating for user reviews from Amazon movie reviews using various features extracted from the review text, product and user distinguishers, and metadata from the review. The data is comprised of 1.7 million reviews and contains the following fields: ProductId, UserId, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, Text, and Id.

Data Exploration: I started by going through the data to try and understand what was given to us. The first thing I noticed was that the scores of the reviews were imbalanced, with the majority of the reviews being five out of five stars. Originally, I had taken this into account and balanced the data, but since our goal is to predict the exact score and evaluate on accuracy, it did not make sense to balance the data. I then printed out the total unique users and total unique movies, in order to see if reviews could be by the same user, and if there were multiple reviews on the same movie. I found that both were less than the dataset, telling me that there were multiple reviews by the same user and multiple reviews on the same movie. Knowing this, I looked to see the total number of reviews of the top 10 users and movies, to ensure that none were significantly more than the others and were therefore not making too big of an influence on the prediction. As seen below in the graphs, there is no one user or movie that has significantly more reviews, and thus I did not need to account for this in my data preprocessing. I then used the python library *Missingno* to print out graphs to see if any information was missing from the data and found that only the scores column was partially missing data, which was to be expected.





Feature Extraction: In order to enhance the data processing, I engineered several different features as well as used various natural language processing techniques, such as sentiment analysis and TF-IDF. After training on my features, I created a correlation matrix as seen in the figure below, which captures the correlation from each feature to every feature. Since we are only predicting the Score of the reviews, I looked at the correlation of every feature to the score column to determine which features were helpful to my model, picking features that had a high or low correlation, neglecting features near zero.



You might notice in the correlation matrix, that Productid_target__enc and Userid_target_enc, which takes the mean of the scores for each Productid and Userid, has a very high correlation to the Score. However, I realized that I did not calculate these features correctly and that it was using the testing dataset when taking the mean. Due to time constrains I was not able to correctly calculate the target encodings, and so I decided to leave these features out of my final feature selection.

Below is a description of each of the features I extracted:

Helpfulness: I calculated the helpfulness ratio by taking the HelpfulnessNumerator which is the total number of users that found the review helpful and divided it by the HelpfulnessDenominator which is the total number of users that found the review helpful and unhelpful. Before dividing I removed the rows in which the HelpfulnessNumerator was larger than the HelpfulnessDenominator; this seems to be an error in the data collection, and filled in the missing values with zero. This feature provides the perceived helpfulness of the review from other users. I also found the amount of people that found the review unhelpful by subtracting the HelpfulnessNumerator from the HelpfulnessDenominator. By finding these features we can begin to understand if other users agree with each review and how much weight and consideration we should give each review.

Time: I extracted the year, month, and day from the Time field for each review. This feature captures seasonal, cyclical, and other time dependent trends that occur when writing reviews.

Text Length: Using intuition, I noted that usually longer reviews have lower scores since the user is typically complaining about the movie in the review. Due to this, I computed the length of the title of the reviews (Summary) and the reviews itself (Text). This feature finds the relationship between length of the reviews and its score. For the reviews that didn't have Summary or Text fields, I filled in the data with an empty string such that its length would be zero.

Sentiment: Analyzing the sentiment of the reviews is a big distinguisher in this project as the score is heavily related to the review that the user provides. I used Vader (Valence Aware Dictionary and sEntiment Reasoner) library to analyze the sentiment of the Text and Summary fields. Vader returns scores ranging from 1 (positive) to -1 (negative) that reflect the overall sentiment. After finding the sentiment score, I then look for exclamation marks ('!') and question marks ('?') in the Summary and Text fields and if there exists an exclamation mark, I add a 0.3 boost to the sentiment, and if a question mark exists, I subtract a 0.3 penalty to the sentiment. I add these boosts and penalties to the sentiment score because often when there are question marks within the review, the user is upset, and the review is overall negative. However, when there are exclamation marks the user is typically leaving a positive review.

Positive/Negative Word Count: I count the total number of positive and negative words in each review using NLTK's (Natural Language Toolkit) Opinion Lexicon, which contains a list of predefined positive and negative words. Using this list, it counts how many of those words are in each review. This feature will help in understanding if the review is overall positive or negative -- when the count is higher for positive words it is a more positive review, and when the count is higher for negative words it is a more negative review.

Review Counts: This feature calculates the number of reviews for each user and the number of reviews for each movie. Since we know there are multiple reviews by users and multiple reviews for each movie, counting the number of reviews gives us a sense of how often a user reviews movies and if their review should hold more weight. For instance, if a user only has one review that tends to indicate they don't leave reviews unless it's very negative or very positive. How many reviews a movie has indicates how popular that movie is, which indicates the overall sentiment towards the movie.

Target Encoding: For each UserId and ProductId I took the mean of the scores and encoded that averaged score based off the reviews.

TF-IDF: I used TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to capture the semantic and contextual information from the text for the Text field, using scikit-learn's TF-IDF library. For the Text field I extracted 1,000 features and ran it on a bigram model, and for the Summary field I extracted 200 features and ran it on a unigram model. Since TF-IDF measures the relevance of a word to the text and adjusts for common words, the model can learn which words are important to the reviews for both the Text and Summary.

Normalization: After finding all the features, I normalized the numerical features so that all the features contribute proportionally to ensure that the model is efficient and stable.

Since the trainset is so larger, I used a portion of the training set to train and extract all of my features, and then applied those features to the full training set and testing set.

Model Creation: I chose to use LightGBM for the model since it is known for its efficiency and performing well with large datasets. I used LGBMClassifer from LightGBM and and used Optuna to find the optimal hyperparameters to provide the best accuracy. My model performed with 65.43% accuracy on the training set and 60.65% accuracy on the testing set.