Introduction

This report will detail my attempt to create a model to accurately predict the star rating associated with user reviews from Amazon Movie Reviews using the given features of ProductId, UserId, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, Text, and Id. Specifically, the process of feature engineering, preprocessing, and the eventual decision to use XGBoost.

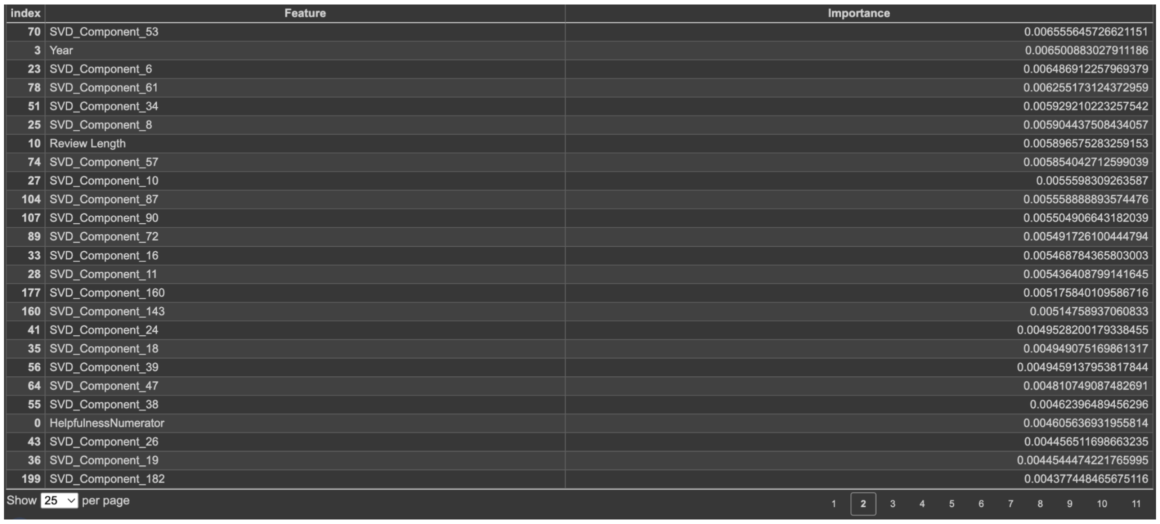
Analysis

The training data provided has 1,697,533 unique reviews, each with the following feature columns: ProductId, UserId, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, Text, and Id. Before detailing the features, it is worth noting that scaling and balancing the dataset were not done. Firstly, scaling was not done as most of the data was already within a small enough range that I believed that scaling would not have that big of an impact. Next, balancing was not done as I did not have enough time nor RAM to conduct the task and when I tried random undersampling into random oversampling, the training accuracy decreased. In addition to the use of these features, new features were created to better predict the star rating. The created features will be discussed in the paragraphs below.

Helpfulness

This feature was one of the default features given in the starter code. Although it was originally given, it was not found in the original dataset. Instead, it was made by dividing the given HelpfulnessNumerator feature by the HelpfulnessDenominator feature. In theory, it should represent how helpful a review is to other users, where 1 indicates everyone who reacted found it helpful and 0 the opposite.

This feature was used in the final model as removing it decreased the overall accuracy by a value between 0-1%. It is worth noting that in contrast to HelpfulnessDenominator and Helpfulness, HelpfulnessNumerator wasn't included as the built-in feature importance functionality of XGBoost ranked its importance quite low compared to the other two as seen in the figure below. As such, it was removed in an attempt to reduce potential overfitting



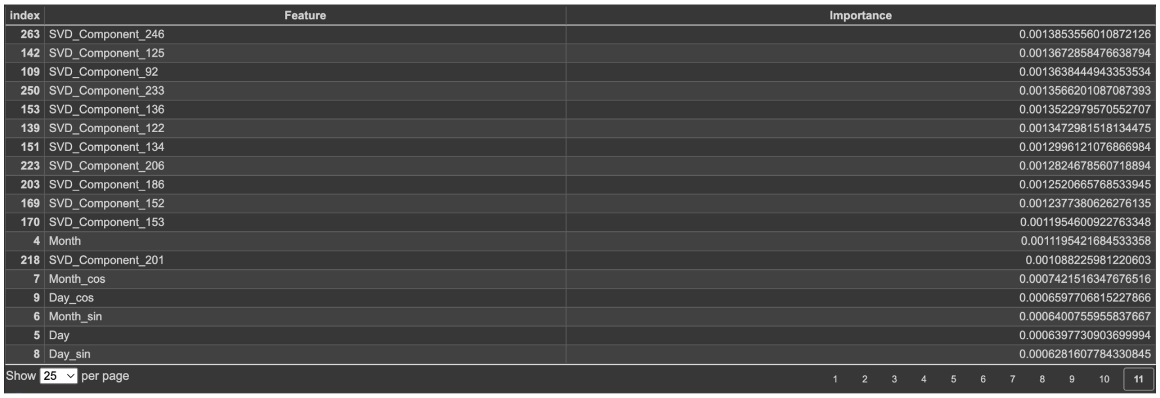
index	Feature	Importance
70	SVD_Component_53	0.006555645726621151
3	Year	0.006500883027911186
23	SVD_Component_6	0.006486912257969379
78	SVD_Component_61	0.006255173124372959
51	SVD_Component_34	0.005929210223257542
25	SVD_Component_8	0.005904437508434057
10	Review Length	0.005896575283259153
74	SVD_Component_57	0.005854042712599039
27	SVD_Component_10	0.0055598309263587
104	SVD_Component_87	0.005558888893574476
107	SVD_Component_90	0.005504906643182039
89	SVD_Component_72	0.005491726100444794
33	SVD_Component_16	0.005468784365803003
28	SVD_Component_11	0.005436408799141645
177	SVD_Component_160	0.005175840109586716
160	SVD_Component_143	0.00514758937060833
41	SVD_Component_24	0.0049528200179338455
35	SVD_Component_18	0.004949075169861317
56	SVD_Component_39	0.0049459137953817844
64	SVD_Component_47	0.004810749087482691
55	SVD_Component_38	0.00462396489456296
0	HelpfulnessNumerator	0.004605636931955814
43	SVD_Component_26	0.004456511698663235
36	SVD_Component_19	0.0044544474221765995
199	SVD_Component_182	0.004377448465675116

Figure 1. Image of Feature Importance Table with HelpfulnessNumerator

Year

This feature was extracted from the given Time variable. It represents the year in which the review was published. This feature was created because I believe the specific language people use can be dependent on the time in which the review was created. Also, scoring inclinations may change over time.

Originally, time was split into the features of Year, Month, Day, Month_cos, Month_sin, Day_cos, and Day_sin. The sin and cos variables were created to see if there was any cyclical nature of reviews found within the month and day variables. However, only the Year variable was used to train the final model as the rest of the time variables did not seem to have that big of an impact on the overall decision tree. This can be seen in the figure below where the lowest importance features, excluding Year ranked at 26/268 features, were the other time variables.



index	Feature	Importance
263	SVD_Component_246	0.0013853556010872126
142	SVD_Component_125	0.0013672858476638794
109	SVD_Component_92	0.0013638444943353534
250	SVD_Component_233	0.0013566201087087393
153	SVD_Component_136	0.0013522979570552707
139	SVD_Component_122	0.0013472981518134475
151	SVD_Component_134	0.0012996121076866984
223	SVD_Component_206	0.0012824678560718894
203	SVD_Component_186	0.0012520665768533945
169	SVD_Component_152	0.0012377380626276135
170	SVD_Component_153	0.0011954600922763348
4	Month	0.0011195421684533358
218	SVD_Component_201	0.001088225981220603
7	Month_cos	0.0007421516347676516
9	Day_cos	0.0006597706815227866
6	Month_sin	0.0006400755955837667
5	Day	0.0006397730903699994
8	Day_sin	0.0006281607784330845

Figure 2. Image of Lowest Ranked Features in Feature Importance Table

<u>Review Length</u>

This feature was extracted from the length of the text column from the given data. Originally, I wanted to vectorize the text column and train the model on it. However, my google colab kept crashing and running out of memory when I tried to vectorize the text. Not to mention, I was also unable to save the data since the amount of time it took for colab to write to a csv was egregious and my google drive could not store that much data. Thus, I decided on incorporating other aspects of the text into my model.

Similarly to the previous features, this feature was kept to train the model as its importance was ranked pretty highly at 32 out of 268 features.

<u>User Average Review Score</u>

This feature was extracted from the score column. As the name implies, it describes the average score that each user gives based on all of their reviews.

This feature had the highest importance at 0.0875 out of all 268 features used to train the final XGBoost classifier model. This feature was added after having realized that I could make use of the score column in helping me create features to train the model with. Originally, I believed we were not able to use the score column at all since score is the feature that we are predicting. However, after looking at Piazza and reading other people's reasoning, I decided to make use of the score column. This feature was created as logically, users would create more than one review and scoring is typically dependent on users. Users who score more harshly are more included to give lower scores on average, while the opposite is true as well.

<u>Vader (Text and Summary)</u>

This feature was extracted through the use of the SentimentIntensityAnalyzer from nltk using the vader-lexicon. This feature was the first feature I thought of adding since a large part of the given data comes from the review itself. The sentiment from the text would most likely indicate the star rating I originally thought about using another sentiment analysis library such as textblob, but settled on vader since it included a negative, positive, and compound sentiment score as opposed to textblob which only had polarity and subjectivity. These features help to capture and quantify the overall sentiment a user has towards a movie. A high negative sentiment would correlate with a lower score while a high positive sentiment, a higher score.

These features, especially the compound sentiment score which combines the negative and positive score, had the next highest importance for the XGBoost model after user average review score. Considering that the text and summary undoubtedly play a major impact on what the user scores the movie, this feature was kept.

<u>SVD_Component_x</u>

The last feature extracted was a vectorized version of the summary column with 1000 max words run through a TruncatedSVD model of 250 components. The values for the parameters were chosen based on the values used during lab4. I also tested out values greater than a thousand for the max words parameter, however, Google Colab did not have enough memory to run the following SVD after 1200 max words. As for the number of components parameter for the TruncatedSVD, 250 was chosen as it seemed to preserve the majority of the information whilst keeping the total amount of added features to a relative minimum. 250 to 1000 components were about a 0.5-1.0% difference at most. Additionally, I also tried using PCA to automatically determine the amount of components I should keep by setting it to capture about 90% of the variance. Despite this, it had about similar success as SVD, so I stuck with SVD since we have not really used PCA in class yet.

These components had the greatest effect on the model's overall accuracy. By just adding the vectorized version of the summary to train the model, the accuracy improved from the base ~48% to about 57%. Similarly and dissimilar to the sentiment analysis features, these components capture the bulk of the data from the review in addition to potentially more data as it returns a lot more columns, and thus have the most influence on the model's accuracy. Less data is lost from the process. Ideally, I would have also trained my model on the vectorized version of the text. Sadly, Google Colab and my own laptop were not powerful enough to accomplish this. Not to mention, the amount of time it would have taken for me was unfeasible. Hence, only the summary was vectorized.

Model

A couple of different models were tested before finally settling on the XGBoost Classifier model. Namely, that of KNN from the starter code and Random Forest. Firstly, as KNN was found within the starter code, I tested it for a couple of iterations on the given features. However, as I knew from prior experience that better models such as XGBoost existed, I pivoted quite early on. As for Random Forest, I ran one iteration of the model after having run a GridsearchCV on the XGBoost classifier to no avail to see if it would potentially work better. Nevertheless, it performed about equal to the XGBoost model but with the major caveat that it took about 30 minutes to an hour to train in contrast to the XGBoost model which took about 3 minutes at maximum. As a result, I settled on using the XGBoost Classifier.

As previously mentioned, the final model implemented was a XGBoost Classifier model. XGBoost was chosen as from previous experience, I learnt that XGBoost has been proven to work well with large amounts of data, and it runs pretty efficiently as well. Considering that I did not have that much time for this task, it seemed perfect for the task at hand. To ensure reproducible results, the random state was set to 0. The model was trained on the above discussed features: Helpfulness, year, review length, user average review score, vader_neg_summary, vader_pos_summary, vader_compound_summary, vader_neg_text, vader_pos_text, vader_compound_text, and the 250 SVD components, as well as the given feature of HelpfulnessDenominator. As discussed above, the features were chosen as they seemed to have the biggest effect on the model's final accuracy. Additionally, the default parameters of XGBoost, n_estimators=100, learning-rate=0.3, max_depth=6, subsample=1.0, and colsample_bytree=1.0, were used, as alluded to in the model above, it did not seem to have any drastic improvements in the training or test set after having run a GridSearchCV with cv=3 for about 8 hours.

It is also worth noting that since the XGBoost classifier requires the output data to be zero-indexed, I had to shift the testing columns by 1 when training and creating the submissions.

Results

The model I trained got about 0.641 on the testing set, and 0.660 on the training set. However, only about 0.58 on the Kaggle dataset. From this, it can be concluded that the model most likely overfit on the training and testing set. In future competitions, it would probably be better to reduce the amount of features used in an attempt to reduce overfitting.