Introduction/Task

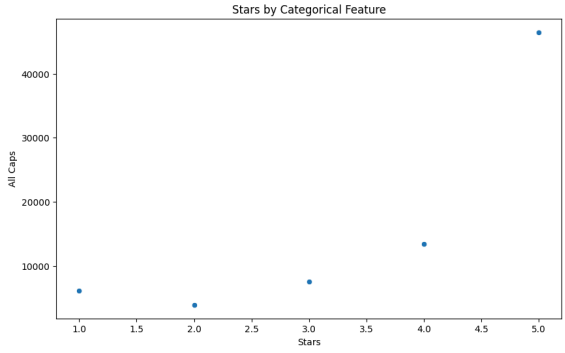
As data science continues to grow as a field, there are increasing applications for data analysis. One such application is the prediction of star ratings of user reviews from Amazon Movie Reviews based on the available features of the reviews. The goal of this project is to identify which models can better predict the star reviews, as well as which features can be selected or engineered to produce a stronger prediction. The data used for this project consisted of 1,697,533 unique reviews from Amazon Movie Reviews, where reviews consisted of the rating itself and other metadata. Metadata included: ProductId, a unique identifier for the movie; UserId, a unique identifier for the poster; HelpfulnessNumerator, the number of users who found the review helpful; HelpfulnessDenominator, total number of users who indicated any helpfulness of the review; Time, the timestamp of the review; Summary, a summary of the review text; Text, the text of the review; Id, a unique identifier for the review itself. It is possible that certain metadata are missing for each review, and for the reviews missing the score, those were used as the test set that the model would predict the score for. Ultimately, the best approach I found was using various features such as sentiment analysis, polarity, and TF-IDF as part of a stacked ensemble model with XGBoost, RandomForestClassifier, and KNN, which I will explore more in depth in the remainder of this report.

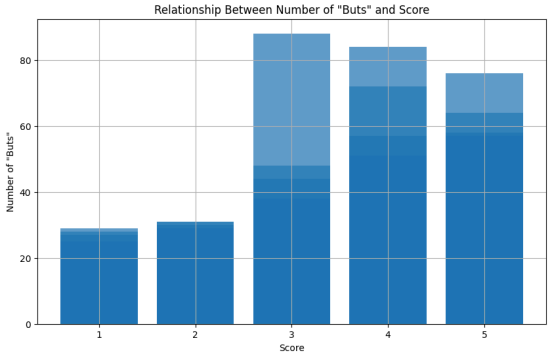
Feature Selection

As a general method, I decided on the best features to use by plotting the relationship between the rating of the review and whatever feature I was evaluating. These are definitely not the best features and I try to evaluate the advantages and potential drawbbacks of each feature.

The first feature I included was called Stars, which checked if the text of the review itself contained a number between 1 and 5, since I thought that there may be a chance that some reviews explicitly stated how many stars they gave the movie. However, I believe a big drawback is that if the user uses a number in the review but didn't actually rate it thatt, then the model may mispredict. An example of a sentence like this could be "I would've given this 5 stars but, ...", for a 4-star review. Upon implementation, this feature did increase my accuracy, so I included it.

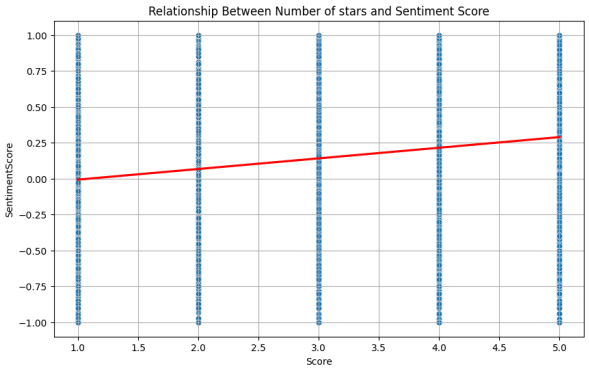
Another feature I analyzed was called AllCaps, which denoted if a review's summary was in all capital letters. Plotting this relationship below left showed that there is a correlation between all capital summaries and 5 star reviews, so this feature aided in predicting 5 star reviews. This could be due to the fact that the dataset had mny more 5-star reviews, but I do think it made accurate predictions.

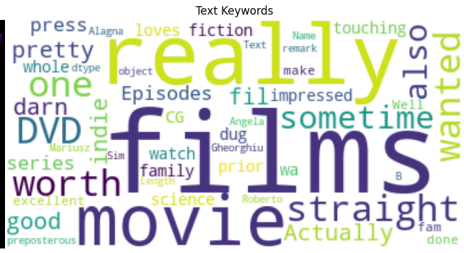




The next feature I analyzed was Buts, which was a feature that finds the total number of times the word "but" appeared in a rating class. As mentioned before, the Stars and AllCaps features were able to find 5-star reviews, so I thought that reviewers using the word "but" would most likely be giving subpar reviews. Upon plotting as seen above right, I found that there is some correlation between the number of "but"s and being able to distinguish between 1, 2 star reviews and 3, 4, 5 star reviews. Some drawbacks I think that could arise are that similar sentiments could be expressed, but not through the word "but", so that would be something I try to account for later.

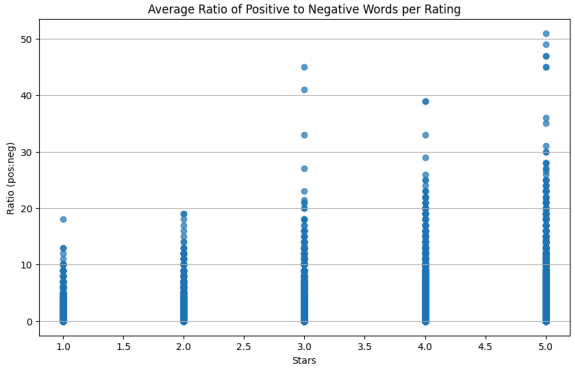
Another feature I looked at was the sentiment scores of both the text of each review. To compute sentiment score, I utilized the Textblob package from NLTK. I hoped that sentiment analysis would allow me to target some of the drawbacks of using the occurrences of "but", since this would evaluate overall sentiment rather than just one word. From my plots, I saw that the regression line showed some relation between sentiment score and rating, so I wanted my model to utilize this.

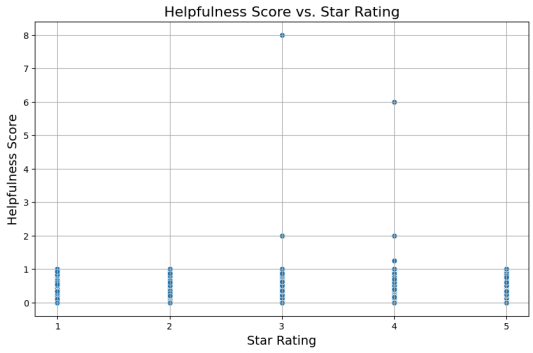




In addition to the sentiment of the text, I also analyzed the sentiment of the summary, as I thought that the summary sentiment would be able to enhance the sentiment analysis of the text as summaries are more concise and there may be less space for error. From the wordcloud above I created with the most commonly used words in the summaries and texts, we can see that there is not a complete overlap of words used, giving some reason to having sentiment analysis for both text and summary. Ultimately, there would be some repeated words, so drawbacks could be that this just was added workload. However, accuracy was increased with this feature.

Similarly to sentiment analysis, I also had a feature called Ratio that evaluated the ratio of the number of common positively-connotated words to the number of common negatively-connotated words in the text. I thought that this ratio would be complementary to sentiment score as it is able to target some nuances of sentiment score such as mixed opinions, the user's writing style, and the balance between positive and negative sentiment. From the plot below to the left, we can see this relationship helped distinguish the 1 and 2 class from 3, 4, and 5 star class.





Another feature I included was Helpfulness, calculated as $\frac{HelpfulnessNumerator}{HelpfulnessDenominator}$. Though the plot above right was a little sparse, it did show that some middling reviews, like 3 and 4, had higher helpfulness score, so despite this being a very subtle relation, I decided to include it.

Another two features I included were an average product score for each product and an average user score for each user. I thought that these features would enhance my model as the model would be able to use the averages to make a better trained prediction. The drawbacks of this I think are that a review may have rated the movie very different from the mean, or there could be overfitting, however overfitting should not have occurred since I used only a portion of training data and did not fit to the test data.

The final feature I implemented was TF-IDF, which we discussed in class. For this, I first preprocessed the data by making everything lowercase and removing the stopwords, as specified by NLTK's stopwords, then implemented TF-IDF and utilized the features that TF-IDF produced. I utilized what we discussed in class to determine that TF-IDF would be beneficial. Additionally, to supplement my learning, I read about how TF-IDF is able to aid in information retrieval and importance of a term among a corpus of documents.

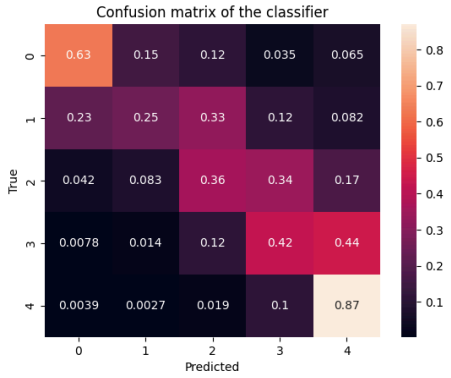
Model

For the model, I used a stacked ensemble with XGBoost, RandomForest, and KNN. While KNN was the original model used in the starter code, I decided to use XGBoost since XGBoost has greater parallelizability and more efficiently evaluates possible splits, while also

countering overfitting by utilizaing smaller trees and similarity scores. Upon reading about XGBoost's growing popularity in Kaggle competitions, possibly also due to its accessibility in packages, I decided to use it for Kaggle as well and it increased my accuracy score from 0.43 to 0.57. I wanted to also use RandomForest since I have read that RandomForest provides the highest accuracy. The benefits of RandomForest that caught my attention was that RandomForest is able to balance data sets when one class is more frequent/ infrequent than others, which was characteristic of this dataset, as there was a large proportion of 5-star reviews. I wanted to continue to use KNN as it is a much simpler machine learning algorithm that we also learned in class. While this seems like a lame reason, I decided to keep KNN in my stack since it is simple, easy to understand, and does not require much time for training since it does not require a training phase. While XGBoost and RandomClassifier are already ensemble models, I read an article explaining the benefits of stacking XGBoost, CatBoost, and LightGBM. While my code did not utilize CatBoost or LightGBM, I thought that stacking would still be able to benefit me as it can combine the strengths of different models. I thought that this would allow me to use both the speed of XGBoost and balancing of RandomForest in my model. For my meta model in my stack, I used logistic regression, as I read that logistic regression is better for discrete classes while linear regression is better for continuous classes. I learned that logistic regression makes binary decisions which was confusing to me at first, but I thought about each review being in each class as a binary choice.

Validation

To validate my model, I utilized the accuracy score from sklearn that was implemented for us in the starter code. From this accuracy score, my best submission had an accuracy of 0.65. While I did not have the time to implement any cross validation, I did have some ideas on the validation of my model based on the features I chose. From my confusion matrix, we can see that my model did a better job at classifying 1 star and 5 star reviews and struggled to classify 2, 3, and 4 star reviews. I hypothesize that this is because the features I chose mainly distinguish 1 from the others, 5 from the others, or 1 and 2 from 3, 4, and 5 stars. Thus, I believe that one possible explanation for my accuracy and validation is due to the features.



Challenges

I faced many many challenges during this project. Primarily, I spent lots of time trying to familiarize myself with the data to determine which features would be best, which definitely ate away at some of the time to actually try out different models and hyperparameters. Additionally, since this is the first data science course I have taken, I definitely felt like I was a step behind when it came to understanding different models and feature engineering. However, attending office hours and using resources helped me to some extent.

Future Iterations

There is so much more that I would like to complete for this project that I didn't have enough time to or honestly didn't focus on prioritizing when I should've. Something I hope to do in the future is hyperparameter tuning using GridSearchCV from sklearn, since this would help me determine the best parameters for each of my models, and thus increase accuracy. However, when I initially began to use GridSearchCV, it took a long time to run and never finished, potentially due to the stacking, so I decided to use my runtime on other aspects. Additionally, I hope to add LightGBM to my stack ensemble, as it is a very quick and scalable model for classification, and upon hearing from the top competitors in the class, LightGBM was definitely a popular choice. In general, I also hope to better engineer some more features, specifically to enhance the classification of 2, 3, and 4 star reviews. Though my model eventually got better at classifying these reviews, I hope it is something I can continue to work on. I also want to determine how I can use time, as I had plotted time and seen some correlation between timestamp and rating, but actually using the feature decreased my accuracy, so this is something I would definitely explore.

References

Simha, A. (2021, October 7). Understanding TF-IDF for Machine Learning. Capital One. https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

Lev, A. (2022, December 19). XGBoost versus Random Forest | Qwak's Blog. Www.qwak.com. https://www.qwak.com/post/xgboost-versus-random-forest

Nvidia. (2024). What is XGBoost? NVIDIA Data Science Glossary. https://www.nvidia.com/en-us/glossary/xgboost/

Random Forest. (n.d.). Corporate Finance Institute. https://corporatefinanceinstitute.com/resources/data-science/random-forest/

Hachcham, A. (2021, July 12). The KNN Algorithm - Explanation, Opportunities, Limitations. Neptune.ai. https://neptune.ai/blog/knn-algorithm-explanation-opportunities-limitations

Rout, A. R. (2023, January 10). Advantages and Disadvantages of Logistic Regression. GeeksforGeeks. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/