CS 506 Midterm Report

Matthew Cheng

Before implementing my algorithm, I went through the columns in our Amazon Movie Review data to determine what we could use to predict an ordinal five-star rating. I dropped the 'Time' column as it provided an exact date (when converted from Unix time) of when the review was posted, which did not seem that important. 'HelpfulnessNumerator' represents the number of helpful votes and 'HelpfulnessDenominator' represents the sum of helpful votes and unhelpful votes), which we could then grab counts and a ratio from. For instance, you can calculate the number of people that thought a review was unhelpful by subtracting the numerator from the denominator (in my dataset, I decided to name it 'Dislikes'). This actually became my second best feature importance from my XGBoost model. For the unique identifiers like 'ProductId' and 'UserId', I used the groupby() function to aggregate counts of how often they appear. Users could be much more active by frequently writing reviews on multiple products and products with a lot more reviews should be more popular.

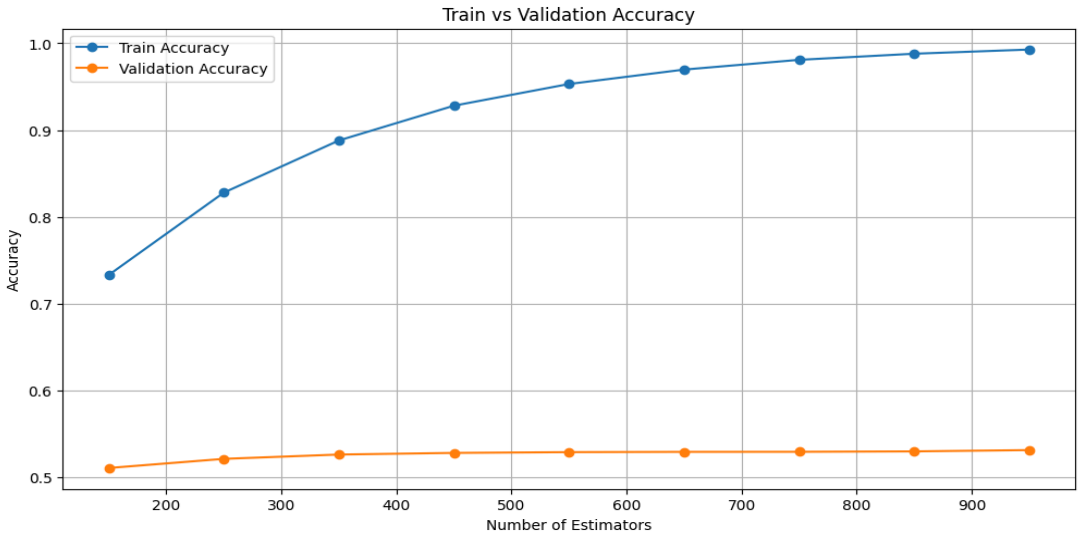
For 'Summary' and 'Text', there were two steps that I took, one being preprocessing and the other being feature engineering. For my text cleaning and standardization, I decided to convert everything to lowercase to ensure consistency, removed special characters, removed common words that wouldn't add any meaningful information (also known as stopwords, i.e. "the", "is", and "at"), and converted words to their base form (using a lemmatizer with part-of-speech tagging, i.e. "running" becomes "run" and both are verbs). For my feature creation, there were a lot of methods I used. Starting with basic text statistics, I calculated the length of a review, the number of exclamation marks and question marks, the number of capitalized words, and the count of different types of words being used (verbs and adjectives). Longer reviews might indicate that someone is more thorough and might have a nuanced opinion, leading to star ratings between 2 and 4. Furthermore, exclamation marks may be related to someone having a positive reaction while question marks means someone doesn't understand an aspect of the movie (could be the way a scene was shot or the way an actor plays their role).

I utilized sentiment analysis methods like TextBlob and VADER. Both of these libraries allowed me to calculate a numerical measure of how positive/negative (in the range of -1 to 1) and subjective/objective (0 to 1) the writing is. TextBlob is known to be well-suited for longer texts, so I used it on the full reviews so it can analyze overall language patterns and contextual sentiment. On the other hand, VADER was designed for short-form text like social media, so I used it on the brief summary of each review to hopefully capture the use of informal expressions, punctuation, and capitalization due to how short they usually are. I additionally implemented TF-IDF (Term Frequency-Inverse Document Frequency), which combines how often a word appears in a single document (term frequency) with how rare that word is across all documents (inverse document frequency). This gives more importance to words that are

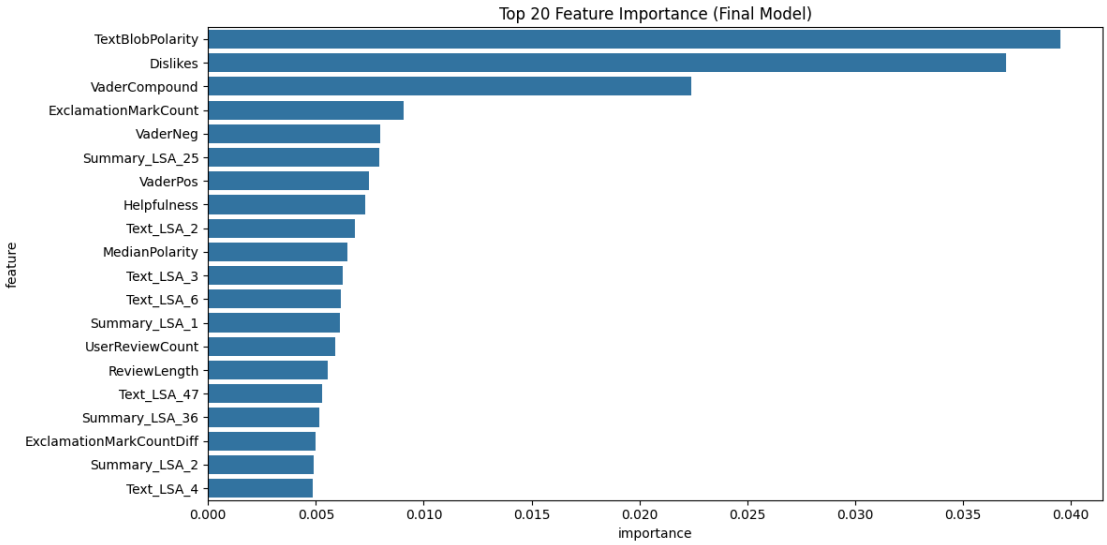
distinctive to specific documents rather than common words that appear everywhere. I decided to fit and transform with TF-IDF for the 'Summary' and 'Text' columns separately instead of doing it in combination to process text embeddings. This was the most computationally intensive part of the task, even after trying to process the text in batches. The second worst in terms of time needed was using TextBlob as a way to provide sentiment scores. Afterwards, I used LSA (Latent Semantic Analysis) to reduce the dimensionality of the document-term matrix produced by TF-IDF to produce a manageable and more meaningful set of features I can use.

After processing the text columns, I combined it with the basic text features (number of capitalized words, number of exclamation marks, number of question marks, sentiment scores, helpfulness), leading to a large, high-dimensional dataset. To help with computation speed and performance, I decided to sample each of the classes equally (20000 observations for each star rating, so a total of 100000 observations). There were a few reasons to balance out the classes for my training dataset. This is because a model that is trained on imbalanced data would tend to become biased towards the majority class, in this case, 5 stars. Ensuring fair learning across categories would prevent the model from just predicting the most common rating and allow for better evaluation across all classes. However, there were drawbacks to this. I am most likely not reflecting real-world rating distributions of how movie ratings are and undersampling majority classes could lead to potentially losing important information. In the end, I decided to stick with balanced classes but I believe it was the wrong decision on my part, as the performance of the model was lacking.

I stuck with XGBoost as my classifier, which is an ensemble method that consists of multiple decision trees. Compared to random forests which typically build out large trees and average the predictions, gradient boosting uses weak learners and takes previous residuals to fit the next model. In the paper, "Handling High-Dimensional Data and Classification Using a Hybrid Feature Selection Approach", the authors Shobit Agrawal and Dilip Kumar Sharma concluded that XGBoost and LightGBM classifiers are high performing, robust models. When writing the actual code, there were many hyperparameters to choose from. I tried adjusting the number of n_estimators or the number of gradient boosted trees, from 150 to 1050 (with increment sizes of 100), while keeping other parameters like learning_rate and max_depth the same. However, I could have done cross-validation with methods like randomized search and grid search, so I could try out multiple combinations of parameters. Doing my original method allowed me to After doing a simple train and validation split on my training data, these were my results (image is on the next page, as it is pretty large).



Based on the graph, there are some things that can be taken from this model. First is the overfitting issue, as the model is reaching 100% accuracy as the number of estimators increases. In other words, as we add more decision trees to the ensemble, the training accuracy increases because the model begins to perfectly fit the patterns and even the noise of the training set. My validation accuracy plateauing at around 55% is concerning, as it indicates that my model is unable to generalize to new patterns it has not seen before.



The feature importances bar graph provides a more understandable interpretation. We see a wide variety of features, from sentiment to basic sentence statistics. The feature that plays a role in decision making was ‘TextBlobPolarity’, which is a sentiment score that ranges from -1 to +1. Then, it is followed by the number of ‘Dislikes’ or unhelpful votes, ‘VaderCompound’ (another sentiment measure), number of exclamation marks, and then a mixture of LSA components from both the ‘Summary’ and ‘Text’ columns.

Works Cited

Madala, Sudheer. “Building a Text Summarizer in NLP - Scaler Topics.” Scaler Topics, 7 Mar. 2023, www.scaler.com/topics/nlp/building-a-text-summarizer-in-nlp.

S. Agrawal and D. K. Sharma, "Handling High-Dimensional Data and Classification Using a Hybrid Feature Selection Approach," 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 2022, pp. 2168-2172, doi: 10.1109/ICACITE53722.2022.9823878.

“What Is XGBoost?” NVIDIA Data Science Glossary, www.nvidia.com/en-us/glossary/xgboost.