CS506 Midterm - Fall 2024

Joshua Lee

The final algorithm/model I ended up implementing was the Random Forest model. Initially I tried using models we learned in class such as KNN and Multinomial Naive Bayes. However, KNN did not produce the most accurate results and I ran into several runtime errors with Multinomial NB, which led me to using Random Forest. I concluded that the RandomForestClassifer would be effective for this project for several reasons: it can handle mixed feature types, it identifies the most important features, handles non-linear relationships, less sensitive to outliers, and is very efficient with large datasets. I emphasize the large datasets because that was my major issue throughout the process, finding ways to minimize and sample smaller in order to complete the running process. This model successfully completed the task faster than any other model I attempted.

Random Forest according to Wikipedia is an ensemble learning method for classification and regression that creates multiple decision trees while training. In our case, it classified reviewer Ids into the discrete variable ‘Score’. For the modeling, I went with a lower number of estimators (n = 50) because my biggest problem was time. I failed to manage time properly and didn’t realize training and data processing would take up to hours to complete. Therefore towards the end of making improvements and changes I had to save time and minimize the number of trees used. This may have led to a more inaccurate and unstable model in the end.

From here, I experimented with different features and methods to increase accuracy. In the data processing step, I worked with various sample sizes. I had to resort to sampling because the size of the dataset was difficult to work with on my computer. By setting the training set to sample sizes of 20%, 10% and 5%, it seemed like 10% was the best option performance wise.

Increasing to 20% actually ended up decreasing the accuracy score. However, again with the issues of time and RAM I had to resort to sampling only 5%. Creating my own features was exciting, and I was left with the given Helpfulness feature and features related to the 'Text' column. The Helpfulness feature needed to be filled in with 0 for missing values. For the 'Text' column, I used Length and Word Count as additional features. And because it was text based, I implemented something similar to Semantic Analysis in Sentiment Analysis, which scores the emotional tone of the text. I felt like this would be a good feature given a good rating most likely had a positive tone in comparison to a bad rating. Unfortunately, I had to fix a couple issues with the Sentiment Analysis because the SentimentPolarity (emotional) ranged from -1 to 1 so for feature selection I had to apply a Min-Max scale to avoid errors dealing with negative values. SentimentSubjectivity captures the subjectivity of the text which I assumed meant an objective review would have a score of 0 and be more neutral compared to a positive or negative review which might be more subjective. Finally we had to drop the Score column because that is what we are trying to predict and categorize. One thing I wish I had the time to implement was any feature related to the 'Summary' column. It was a shorter version of the 'Text' which might have helped with runtime and RAM usage. Furthermore, it's another text-based attribute that could have improved my model's accuracy.

In the end I ended up focusing my attention on the 'Text' column, using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization which reflects the importance of a word. Setting the max features to 500 meant it was capped at the 500 most important words across all the reviews. Therefore the reviews became a numerical matrix that could now be more easily analyzed. Increasing the features would have made the model more detailed and precise but for time efficiency I left it at 500. Latent Semantic Analysis was utilized, reducing the

dimensionality of the TF-IDF features using Singular Value Decomposition. It captured the overall topics and generalized the model making it more manageable to work with. Throughout the process I had to change several values and types of vectorization that would fully run on my computer. Ultimately it helped with reducing features from 500 to 50, keeping the most notable topics and reducing overfitting.

The feature selection was short, where I ended up removing methods that handled outliers and selected the most important features. My accuracy slightly decreased when I had them in my code so I figured it was best to leave them out in my final submission. My thought process is that I wasn't able to explore every relationship/combination between variables, and did not go specific enough in my features to remove unnecessary features. Therefore having every outlier and feature was important to my model, giving it the specification and detail it needs to be more accurate. One thing I would do better is work more on trying different combinations of features and possibly testing them one by one with visualizations or test data to compare and decide what feature is necessary.

This whole process started with trying out different methods, ending up with decision trees and RandomForest to model my data because of its efficiency. If possible, I would have tried more resource-exhaustive models such as Support Vector Machines to create a more powerful model. RandomForest was a good alternative because of its flexibility with numerical and textual data. The biggest step was dealing with the text, which could provide what I believe to be the most meaningful and distinctual features. Another driving factor of my model was to be memory efficient and time saving given how little time I gave myself to complete this midterm. Lastly it was good for me to leave out some steps that unnecessarily got rid of significant data (outliers, specific features) to keep my model producing at its highest accuracy.