# CS506 Midterm Report

For this midterm, I utilized the starter code. Initially, I wasn't very confident with what I was looking at, so I spent some time trying to interpret each line of code to become familiar with what was happening. I then went to our lab and asked the TA some questions that I had regarding the starter code. One key idea I learned about during this initial process was the idea of "seeds" and random state. I've heard these terms being mentioned in lectures and labs, but wasn't exactly sure what it really did. I also become more familiar with methods such as df.describe and df.drop.

After I understood the starter code and felt comfortable to proceed, I jotted down some approaches that I wanted to take. These consisted of checking if a user only rated things when they have negative experiences, checking the words and emojis a user uses and whether or not they were positive or negative, using some form of sentiment analyzer, checking if there were words or terms that brought contrast being used in the user's review such as "but," "however", "on the hand," etc. These would suggest there were good and bad parts in the rating so the user likely rated it either a 2, 3, or 4. Once I had these approaches in mind, I did research to find if there were any resources that would make it easy for me to implement any of my ideas in my code.

This is when I discovered the TextBlob library in Python that helps process textual data. I mostly just used it to help me get the sentiment of the texts and summaries in our data. Initially, I disregarded the subjectivity part of the tuple that is returned by blob.sentiment, but I later used it which increased my accuracy percentage. This could be because it provides more context to the polarity provided by blob.sentiment. I realized that the range of my different features at this point varied a lot and decided to test what would happen if I multiplied the polarity and subjectivity by different numbers. My model then began predicting certain ratings much more often but I couldn't make out why.

I then realized I should use the StandardScaler class to standardize my features to have a mean of 0 and a standard deviation of 1. This helped my algorithms perform better since they were all on the same scale.

From this point I went back a bit and did more preprocessing and chose more features. Here, I began lemmatizing words that could be found in the text and summaries and also removing stopwords. I then decided to create a TF-IDF matrix, removed the stopwards from it, and combined it with some of the other original features. I added the word count of texts as a feature as well and these new features increased my accuracy score.

Something that I failed to take note of early on that hindered me was the imbalance in the data. About 50% of the scores were 5 stars and so when I created my model, it almost always predicted a score of 5. I searched up different ways to handle this issue and discovered a few methods that either undersampled or oversampled data to create a more balanced dataset. I utilized RandomUnderSampler and SMOTE to do this which resulted in a much better Confusion matrix, but my accuracy percentage did drop.

My assumption is because since most of the data consisted of scores of 5 and we predicted 5 most of the time, our naive model that didn't take into account the imbalance of data performed better than our model that did take account the imbalance. However, I knew I was headed in the right direction because the range of my predictions were very accurate. When the score was a 5, my model would mostly predict 5 and 4. When my score was a 1, it would predict 1 or 2, and when my model was in the middle, it would usually predict between 2, 3, and 4. Below are two plots to look at. Note that the y values for the two plots differ(I would like to fix this, but i'm running something right now and can't recreate the figure until it finishes running).

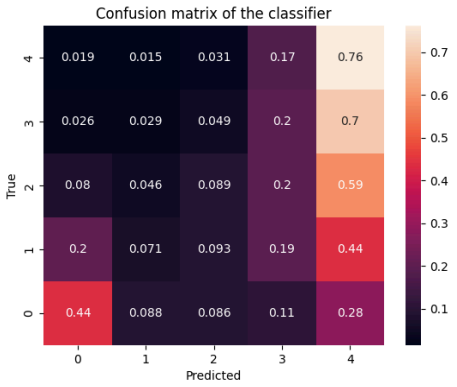


Figure 1:

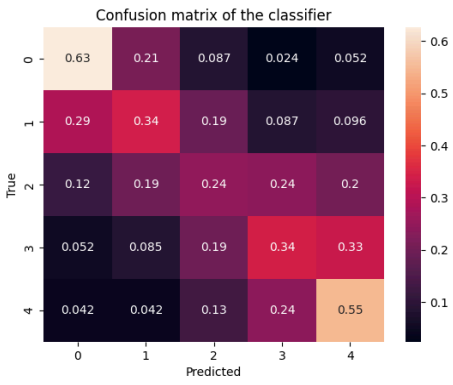


Figure 2:

We see in figure 1 what our plot looks like with the imbalance. We predict a score of 5 much more than any other number. On the other hand, figure 2 shows a more balanced diagonal like. We predict in the right range and see a more light diagonal indicating that our predictions are more accurate overall. I then decided to look at different types of modeling algorithms and experiment with them. Instead of using the KNeighborsClassifier, I implemented

the RandomForestClassifier which resulted in better accuracy. It's important to note that I learned more about GridSearchCV and RandomizedSearchCV in this process. I initially used GridSearchCV with multiple parameters which took a very long time and moved towards using RandomizedSearchCV.

Now, I'm currently working with a Linear regression model which I used on a small sample of the data already. It produced promising results and gave an accuracy of 66% on my data. Also, I went back and created TF-IDF features in parallel so my code runs faster. My Linear regression model hasn't finished working with my entire dataset, but this is what the confusion Matrix looks like when I used it on 2% of the dataset.