Sentiment Analysis on Amazon Instruments

Ravinder Rai - Mar 26, 2023

Introduction

This report is on a sentiment analysis project, where the text data comes in the form of product reviews, and the products are instruments from the online shopping platform Amazon. Online shopping has revolutionized the way people buy and sell products, and this of course includes instruments. With countless types of instruments and related accessories available on the websites like Amazon, customers rely heavily on product reviews to inform their purchasing decisions. As a result, sentiment analysis has become an increasingly popular tool for businesses to understand customer feedback and improve product offerings. Here we will explore the results of this sentiment analysis project, where we will go over all steps of the data science lifecycle. Of course, the purpose of sentiment analysis is to find valuable insights and the strengths and weaknesses of products, as well as identify areas for improvement, but this is typical and not hard to interpret. Thus, we will focus more on the technical aspects of this project, and show what the best model is in detail, and how it was obtained.

Problem Statement

In essence, sentiment analysis will simply tell you if a review is good or bad. It might seem pointless because we could just read the reviews for ourselves, but this is both slow and prone to human error. Human error can come from subjectiveness, especially in regard to reviews that take a neutral stance on a product. And humans take time to read too. Even if the reviews are short, they might come in high volumes, which will be hard for humans to go through on their own. In contrast, sentiment analysis will allow us to automate the process of analyzing reviews and provide a more objective view of the overall sentiment toward a product.

Thus, the problem we aim to address with this sentiment analysis project is to deal with the lack of actionable insights that instrument manufacturers and retailers have from customer reviews on Amazon, but without human error or slow speeds. Technically speaking, we will use the review text for the instruments to predict whether the reviewer liked the product or not, or just thought it was okay.

Data Wrangling

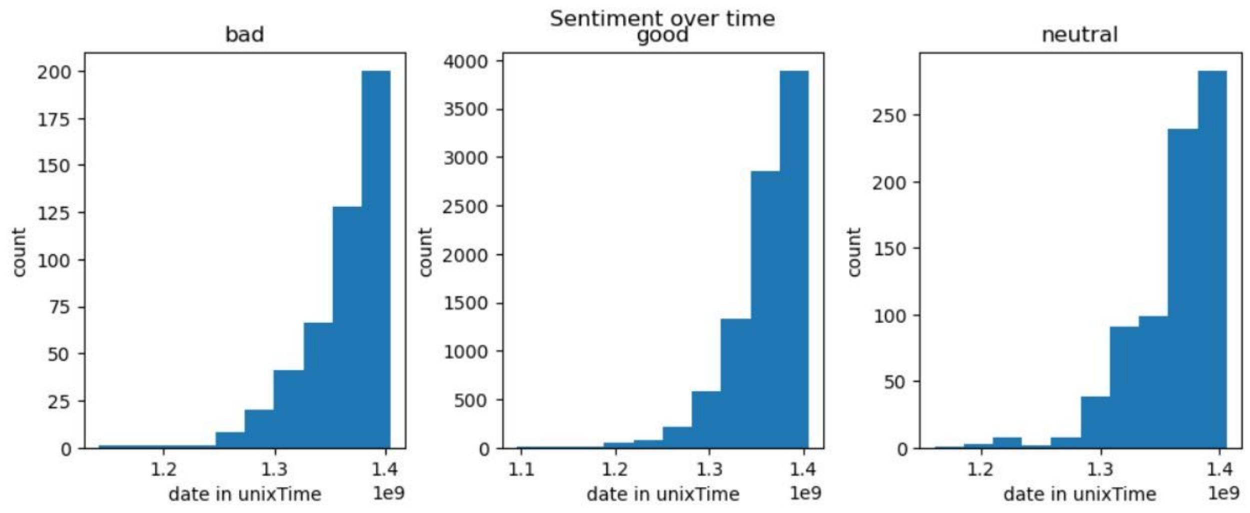
To start this project, the first step was of course to collect the data, which in this case was simple as it came from Kaggle. The data had about 10,000 reviews for different types of instruments, along with their associated sentiments. Interestingly, Amazon product reviews have a title for their reviews, which can also be useful. You can actually go to Amazon right now and see this, but in any case, this text column will be used later alongside the review section. Some other columns were included in this dataset, like the reviewer name, the data, and two ID columns, one for the reviewer and one for the product. These columns were not of any use here, but there was one more called "helpful", which indicates how helpful the review was.

Now when it comes to cleaning, there were luckily no missing values. In fact, there was not really any cleaning to be done on the overall data, with the exception of text cleaning. So, before exploring this data, the next thing done was text cleaning. The text cleaning performed was standard, so to sum it up we performed tokenization, stopword removal, punctuation removal, and lemmatization. There was overall not much text, so lemmatization was used over stemming since it is in practice known to be better, and run times were not too long to run it. It should be noted that data wrangling, exploration, and pre-processing steps were not all done linearly in order, so some of these steps came after or before some data exploration, which we will talk about now.

Data Exploration

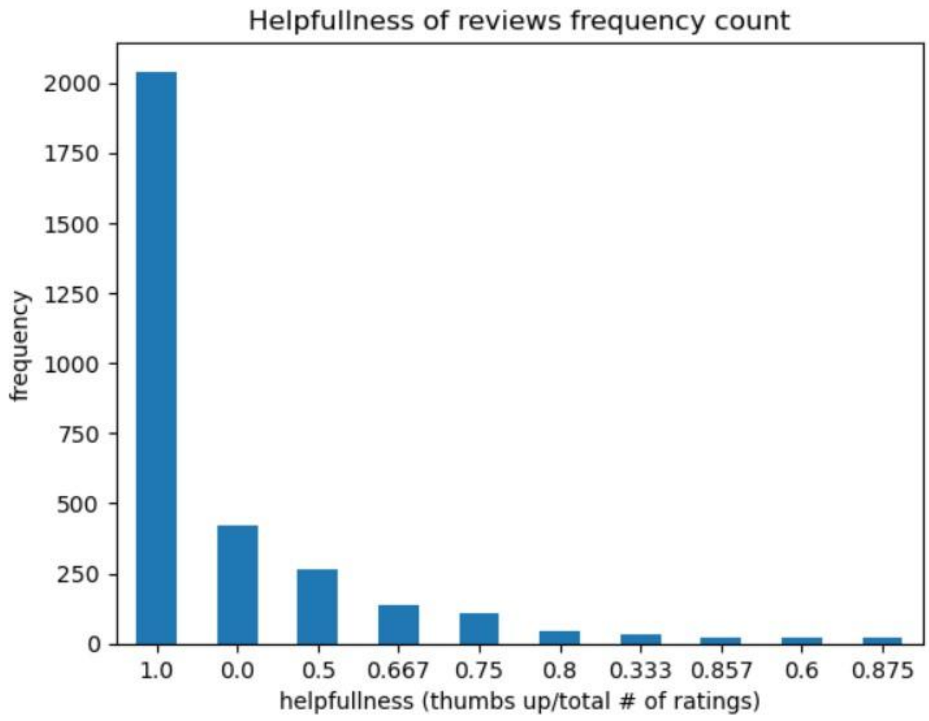
We'll cover each column one at a time, roughly. To start, the first column explored here was the ID column. Although they were discarded, and it was known they would likely be early on, it is always important to explore every column anyways. In this case, the only notable thing found here was that most products and reviewers had 5-9 appearances in the data. This means that each instrument had 5-9 reviews associated with them, and each reviewer reviewed 5-9 instruments. information like this could potentially be useful when exploring individual reviewers, but more likely individual products, to see how many good or bad reviews they have.

The next exploration done was on the date columns. This step was simple, as all we wanted was to see if the sentiments changed over time, which they didn't, as you can see in the following image.



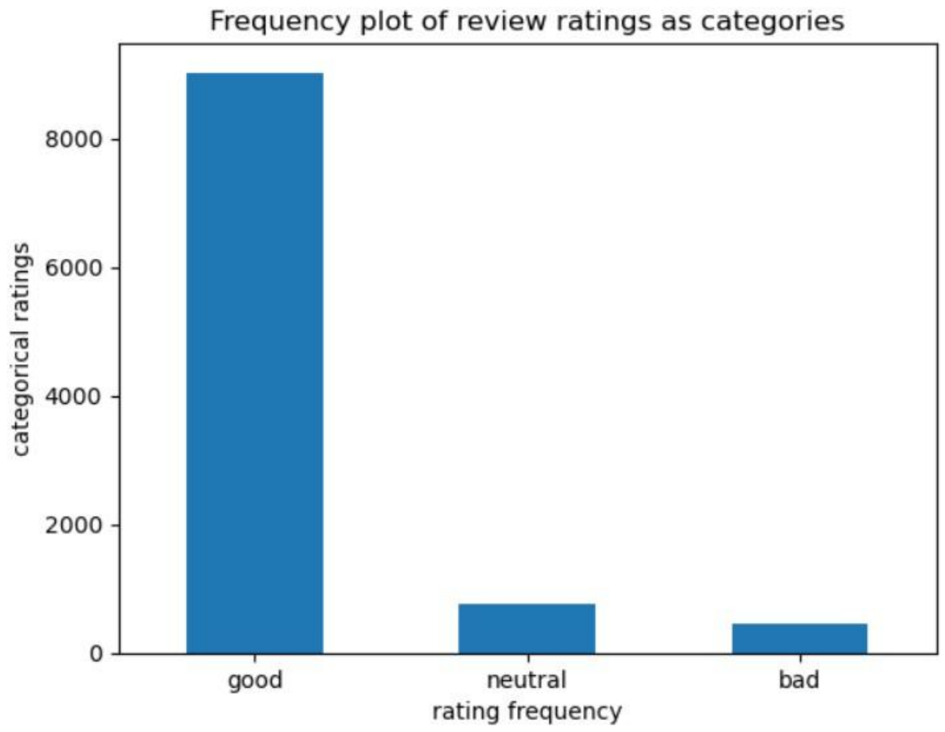
It is because the trends are all the same for each sentiment that tells us that the sentiment overall made no change. In reality, all that happened was that the total number of reviews increased, which makes sense since the Amazon website continues to grow in popularity.

Moving on to the helpful feature. This feature is actually quite interesting, as it can tell us how good a review actually is. The values in this feature come in the form of pairs, where the first element is the total number of thumbs up a review got, and the second is the total number of ratings a review got (i.e. number of thumbs up plus the number of thumbs down). We can then interpret this column in different ways, but the most obvious is to take a ratio or fraction, dividing the first element by the second. This next plot shows the distribution of the feature when we do this.



So you can see that most reviews were rated positively. Interestingly, it turns out that there were a decent amount of reviews that had many dislikes, which you can see in the second bar with a zero ratio or fraction. You can look at this more closely by looking at reviews with many more dislikes than likes, essentially giving you reviews that were themselves review-bombed, so to speak. This means we could potentially build a second sentiment analysis project around this column, which would tell us if a review was good or not, and thus if we should even consider it. A problem for another time, perhaps.

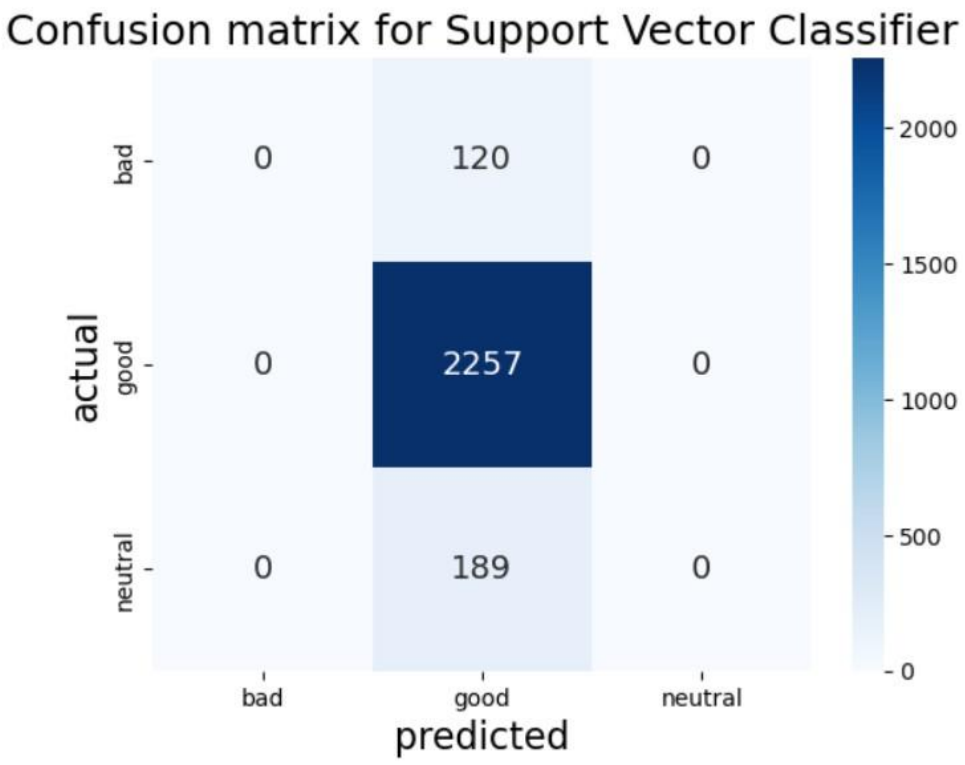
Next would be the text columns, with the reviews and the summary of the reviews, but the cleaning was covered in the previous section, and there isn't much to discuss in terms of data exploration, so we will move into the sentiment feature. This feature was named "overall", which had values of 1 through 5. These values are just ratings out of 5. For this sentiment analysis project, we turned these into sentiments: good, bad, or neutral. This left us with the following distribution for the target variable.



Clearly, we have quite an imbalanced dataset. This will impact the modeling process, which we get into now.

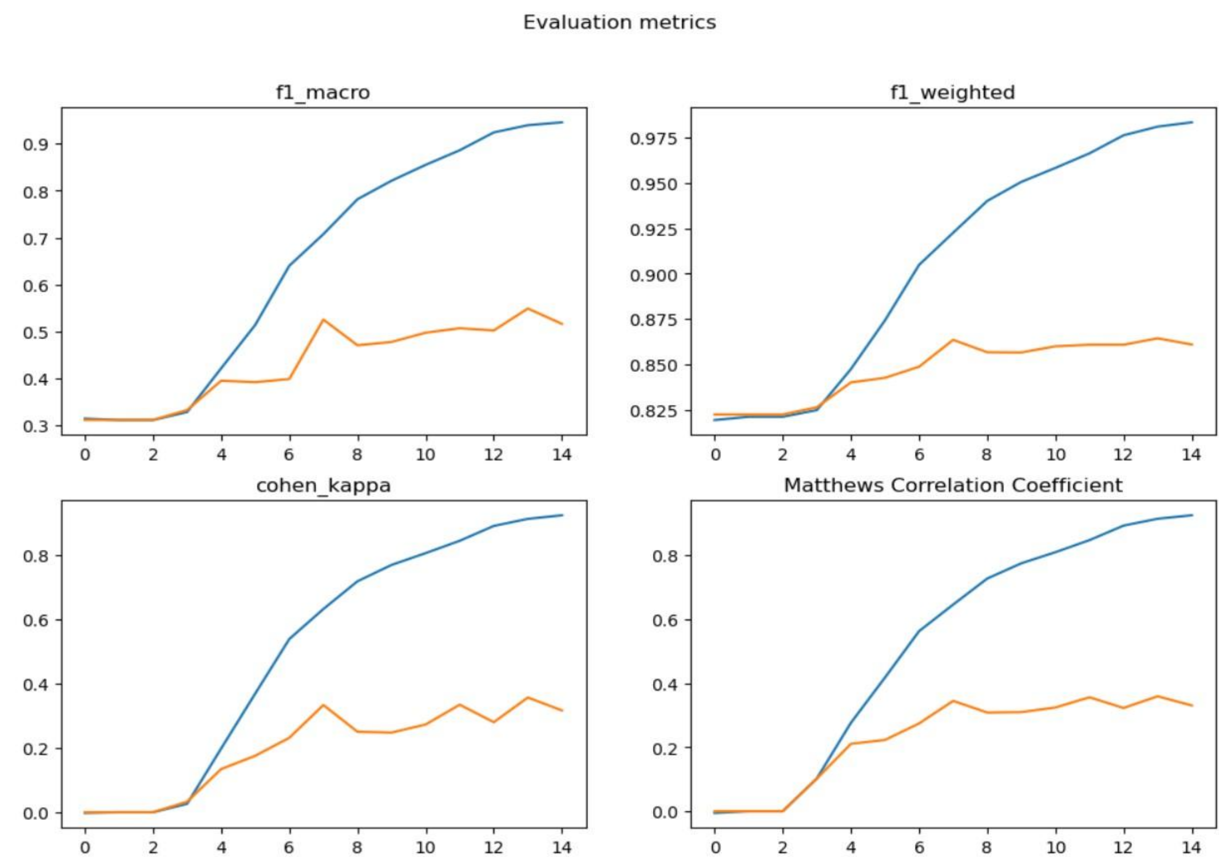
Modeling

For the modeling section, there were essentially four different models attempted. The first two were Naive Bayes and a support vector machine, with the others being neural networks, one with an LSTM and one without. We will first go over the Naive Bayes and support vector machine models as they did not work too well. The first thing to do to get these models to work was to implement a bag of words model. This was done with the python tf-idf function. Neither of these models did well in any case, as you can see in the below confusion matrix.

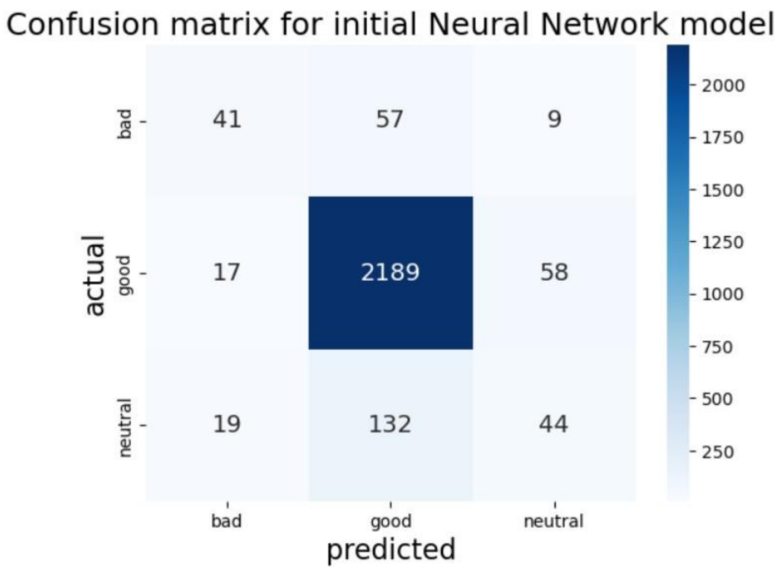


The confusion matrix for the Naive Bayes model was the same, and this is naturally due to the imbalanced nature of the dataset, and how it has so many more good sentiments than the others.

Before moving on, it is important to note what metric was primarily used. From the above confusion matrix, you can see that even though this doesn't look good, this model would achieve high accuracy. This is precisely why accuracy was not the metric used in the end. When testing the neural networks below, there were four metrics tested with them, namely the macro average f1 score, weighted average f1 score, cohen kappa score, and Matthews correlation coefficient. Below are all trends for these scores for some neural networks. This is just to illustrate that all these metrics follow the same trend, thus seemingly indicating that all these metrics would be fine. In the end, though, the f1 macro averaged score was used for final comparisons because the weighted average f1 scores were misleadingly too high, and the others were too low.



Now let's finally talk about neural networks. The first was a simple neural network with just one dense layer followed by an output dense layer with 3 units (one for each class). The Adam optimizer with a learning rate of 0.001 was used, with the categorical cross-entropy loss function. Before the dense layer though, the network starts with an embedding layer followed by a global average pooling layer, to handle text data as input. For an initial test of this model, with non-optimized hyperparameters, the following is the confusion matrix as a result.



So right from the start, we can see that neural networks do much better, just from actually at least predicting more than one class. After lots of testing, getting a model to be better than this initial model was actually quite challenging. This led to the belief that overfitting was too big of an issue for hyperparameter tuning or early stopping to solve. Thus, we now move into many oversampling techniques that were attempted to try and solve this.

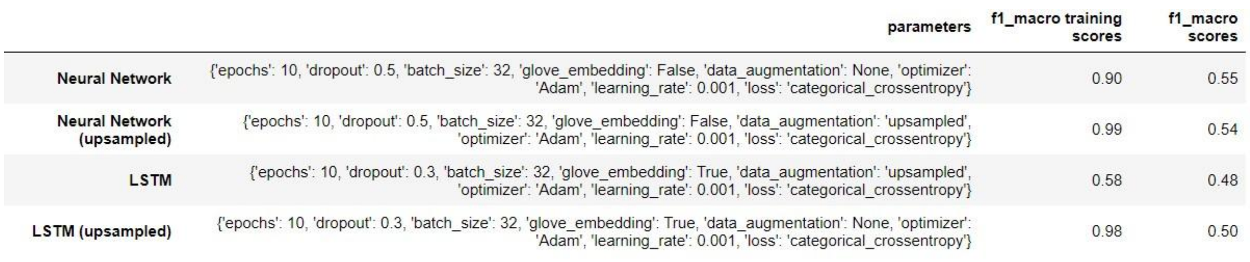
Oversampling is when we try to add data to a minority class, or a class that has much fewer samples than other classes. The first technique here was to simply add duplicates of random samples in the training data of the bad sentiment and neutral sentiment classes. Both of these sentiments were upsampled to have the same number of samples as the good sentiment class. On the other hand, downsampling is when we do the opposite, and remove samples from the majority classes, which was also done to match the number of samples in the bad sentiment class. Unfortunately, with the upsampled data, the f1_macro score (0.51) decreased slightly in comparison to the original training data (0.54), and the downsampled data gave a significant decrease in the f1 macro score (0.33). This takes us to the next oversampling technique, SMOTE. SMOTE is a technique that uses the k nearest neighbors model to create and add similar samples to the minority classes. Like the downsampled data, the f1 macro score dropped noticeably (0.44).

From here, there were only two more techniques attempted. The first was to simply add class weights to the model, which would hopefully help the model care more about the minority classes. The other thing was to use a pre–trained embedding layer, using the glove word vectors. We will skip the detail for these, as neither of these techniques helped at all.

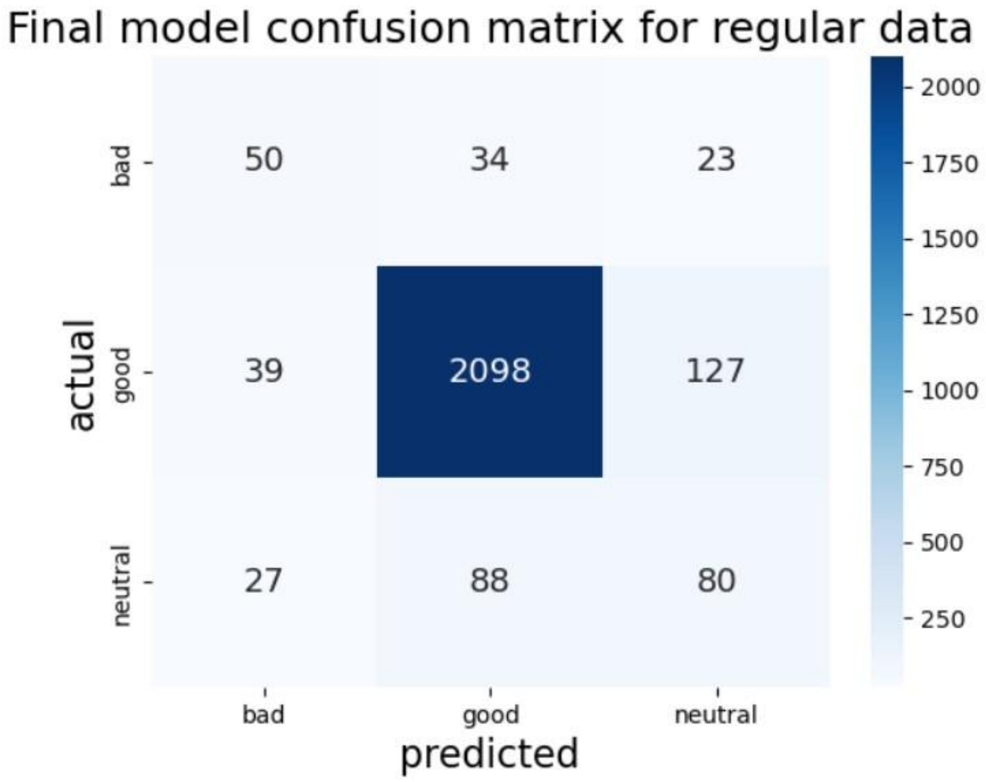
Finally, we have our best sampling technique based on the f1 macro scores, which we can further tune with grid search. The f1 macro scores for both the original and upsampled data were pretty close, so a grid search was performed on both of them, to get the best model. The parameters we searched through were: epochs, batch_size, and dropout. For both models, the optimal values were 10, 32, and 0.5, although it should be noted that not a lot of parameter values were in the parameter grid due to long run times.

Very briefly, we will now cover the LSTM model. This model is the exact same as the neural network model except we replaced the first (and only) dense layer with an LSTM layer. We then tested all sampling techniques here as well, and did grid search on the best ones again. We found that in this case the best models were with the pre-trained word embeddings, alongside the original training data and the upsampled training data, with both giving equivalent f1 macro scores (0.48). After grid search, the optimal parameter values found were the same as the neural network, but with a smaller dropout of 0.3.

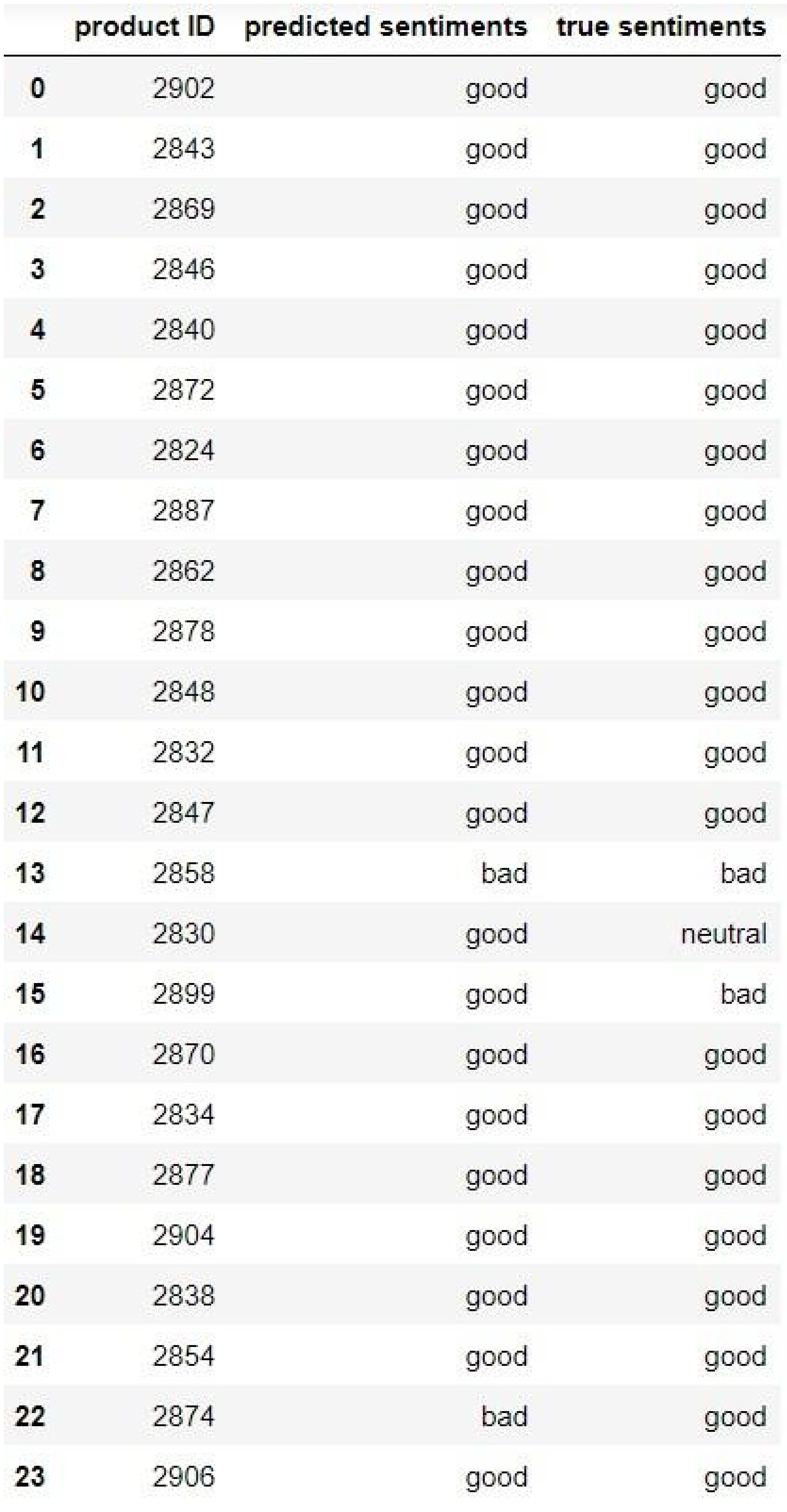
To finish, as is standard practice, we did all model tuning with training and validation data, so to get the final metrics we put them back together and tested the models on the test set. Below you can see a table with the final model metrics, alongside the confusion matrix for the best model, which in this case is actually the neural network (without the LSTM) with just the original training data.



	parameters	f1_macro training scores	f1_macro scores
Neural Network	{'epochs': 10, 'dropout': 0.5, 'batch_size': 32, 'glove_embedding': False, 'data_augmentation': None, 'optimizer': 'Adam', 'learning_rate': 0.001, 'loss': 'categorical_crossentropy'}	0.90	0.55
Neural Network (upsampled)	{'epochs': 10, 'dropout': 0.5, 'batch_size': 32, 'glove_embedding': False, 'data_augmentation': 'upsampled', 'optimizer': 'Adam', 'learning_rate': 0.001, 'loss': 'categorical_crossentropy'}	0.99	0.54
LSTM	{'epochs': 10, 'dropout': 0.3, 'batch_size': 32, 'glove_embedding': True, 'data_augmentation': 'upsampled', 'optimizer': 'Adam', 'learning_rate': 0.001, 'loss': 'categorical_crossentropy'}	0.58	0.48
LSTM (upsampled)	{'epochs': 10, 'dropout': 0.3, 'batch_size': 32, 'glove_embedding': True, 'data_augmentation': None, 'optimizer': 'Adam', 'learning_rate': 0.001, 'loss': 'categorical_crossentropy'}	0.98	0.50



Oddly, the training score for the LSTM model with the original data doesn't seem to be overfitting as much as the others. Time did not permit further exploration, but perhaps this could be the best model with more tuning. In any case, we can finish off by showing some model predictions. Filtering for a certain product in the test set, you can see how the model did below. There are 24 predictions, meaning this product had 24 reviews in the test set. It is no surprise the model did well in predicting mostly good reviews correctly, with a few misclassifications being in the minority classes.



	product ID	predicted sentiments	true sentiments
0	2902	good	good
1	2843	good	good
2	2869	good	good
3	2846	good	good
4	2840	good	good
5	2872	good	good
6	2824	good	good
7	2887	good	good
8	2862	good	good
9	2878	good	good
10	2848	good	good
11	2832	good	good
12	2847	good	good
13	2858	bad	bad
14	2830	good	neutral
15	2899	good	bad
16	2870	good	good
17	2834	good	good
18	2877	good	good
19	2904	good	good
20	2838	good	good
21	2854	good	good
22	2874	bad	good
23	2906	good	good

Conclusion and Future Work

Unfortunately, despite trying lots of oversampling and other techniques to deal with overfitting and this imbalanced dataset, it seems they only make things worse. A few notable things still left to try though, are:

1. Try these oversampling techniques with the Naive Bayes and SVM models.
2. Augment the data with newly created data by replacing words with similar words.
3. There are other types of upsampling techniques similar to SMOTE to try.
4. A mix of upsampling and downsampling.

Also, there is some future work to continue this project that could be done, as alluded to earlier. This was building a second model to predict the sentiment of the reviews, that way we don't have to worry about the quality of the reviews.