Programming for Data Analytics 2

Task 1

Kieran Glezer-Jones
ST10041889

Sept 2022

Contents

Introduction

The ability to determine sentiment from text is a process that could immensely benefit any business with the capabilities of utilizing it. The ability to identify positive and negative responses from messages could help with grouping and understanding certain demographics.

The assignment demonstrates the actions taken to create a model that achieves this with a large data set. The data set in question hosts two very important features which allow us to perform sentiment analysis on it. the data set is that of a few hundred thousand cellphone reviews. The reviews are accompanied by a score from 0 to 10. This score can be analysed and encoded to display either a 0 or a 1. A 0 is a negative review or a 1 is a positive review. We start by defining a sentiment based on the score of the review, in practice the best results came from using scores above 6 as positive and scores below 7 as negative. After a bit more cleaning a model can be trained which takes an X input and predicts a Y output. In this case, the X input will be the new review and the output will be whether it is positive or not.

The dataset in question is a compilation of 1.4 million records of cell phone reviews from many different countries and many different phones. A working model of this dataset could prove to be extremely useful in the presence of a company wise enough to use it. The model could be used to collect data from social media, such as Facebook, Twitter, and Instagram to get an idea of opinions from large groups of users. The fact is that many people don’t leave reviews in the designated review forums that the companies have access to. Instead, they will leave reviews on these social media platforms, which makes it difficult for the company to get an idea of the general opinion. However, with the ability to automatically gain sentiment from these reviews, the companies can programmatically load all tweets with the hashtag #SamsungS8, for example, and then gain a tally of what percentage of the public like it. If necessary, it can be the backbone in seeing which regions like it more, which regions are more likely to give positive reviews, what people are complaining about, etc.

The ability to gain sentiment can aid phone companies in making better business decisions. For example, Samsung might announce that they are taking the charging port on their phones away, this might stir up a lot of negative tweets about the phone on social media. This could be picked up with the model and they might decide that it would be a bad business decision to proceed with that plan.

1 Background

This section looks at what we already know before we start cleaning the data.

1.1 The dataset

The dataset originates from Kaggle and hosts 1.4 million cell phone reviews. It comes equipped with quite a few interesting records but for the most part, we will mostly be focused on the [EXTRACT] feature and the [SCORE] feature.

The [EXTRACT] column is a qualitative column that contains the review of the phone in question. This feature will be the main focus of this task as we will be using it as our only X variable. This means it will need to be pre-processed so that it can fit within a neural net. Not all the records are in English however, so we will first need to filter those out.

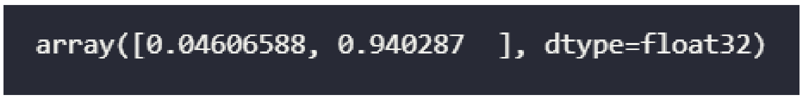
The [SCORE] column is a quantitative column that holds a float value from 0 to 10. This will be used as our sentiment, where records greater than 6 will be 1 and records lower than 7 will be 0. This will act as our dependent Y variable to train the model with and will be our target in the future. The process of converting the scores into 0s and 1s is known as 1-hot encoding.

There are some other useful columns such as the [PRODUCT] column which has the name of the product being reviewed as well as the [LANG] column which has the language code within it. Both of these features will be used in the descriptive analysis portion of the assignment.

1.2 Text Classification

Classification is a supervised machine learning technique that trains the model on known inputs and outputs. The goal of classification is to output a category that differs from the goal of a regression model that outputs a continuous value. The outputs from a classification model are often numeric.

In the case of neural networks, the classification models will often output multiple continuous values. But these values differ from the regression values as they are confidence levels for each category.



```
array([0.04606588, 0.940287  ], dtype=float32)
```

Figure 1 Confidence Levels

For example, the array above shows there is a 94% chance that the text is a 1 which is positive.

Text becomes quite difficult to work with as there is no way for the computer to correctly guess what the meaning behind each word is, as many words can be ambiguous. So, we have to break the words down and pre-process them with either stemming or lemmatization as well as tokenization.

1.3 Exploratory and Descriptive Analysis

Descriptive analytics revolves around the concept of taking raw historical data and then mining it to discover trends and relationships (The Independent Institute of Education, 2022).

When given enough text data some interesting descriptive analysis can be done on it. Word clouds can be created to see which words are popular under different circumstances and certain words can be isolated to get a clearer picture of what is going on. Depending on what we find during the descriptive analysis we may decide to change the way our model works.

1.4 Predictive Analysis

The predictive analysis takes a large amount of historical data to find patterns, trends, and relationships in an attempt to predict "what is going to happen next" (Cote, 2021).

With sentiment analysis it looks at it a bit differently, instead of looking to see if something is going to happen, we look if something is positive or negative. That isn't to say that the model couldn't further be developed into something that predicts and prescribes. For example, the algorithm could be used by Google to make advertising decisions. If you have a Samsung and are speaking positively about it, the model could be used to predict how likely you are to get a new Samsung, in which case google could advertise the newest one to you. If you didn't like Samsung, then google could advertise a different brand to you.

1.5 Sentiment Analysis

Sentiment analysis is a form of text mining it is used to gain insight into people’s opinions, emotions, and attitudes towards something (Medhat, et al., 2014). It’s a form of predictive analysis that is used to predict sentiment.

The end goal of the task is to create a model that can take text as input, and then output whether people are speaking positively or negatively about the product.

1.6 Usability

The recurrent Neural Network needs to be fed data that is appropriate for it. There are a few things to consider when choosing data for a recurrent neural network that looks to output a binary classification.

1.6.1 Legitimate Source

If the data used to train the model is completely fabricated, then it might severely struggle to perform in a real-world environment. The model will ultimately be fed real-life data, for it to correctly gain the sentiment from this it needs to have been trained on real-life data, to begin with. Many people have many different ways of speaking and expressing themselves, and it’s unrealistic to assume that fabricated data could accurately represent all those different mannerisms.

The dataset used consists of real-life reviews of real phones from real people. It allows us to cover a large number of different mannerisms and attitudes. It will be even better if the model is constructed for a phone company since most of the language used is to express how they feel about phones. This can shake off a lot of ambiguity as if someone mentions ‘home’ then they are likely to be referring to the menu instead of their physical abode.

1.6.2 Categoric Target

There needs to be a categoric variable that the model can use to train itself. If we have a review, there needs to be something that says whether that review is positive or negative. Unsupervised text classification can be attempted with the Latent Dirichlet Allocation (LDA) technique, but it is not very reliable, and you might not find the outputs you were hoping for.

Fortunately, the dataset in question has a score variable that can be encoded into a 0 or a 1. This allows us to use that as the target for our model. So, the model knows that the text allocated with a 0 is supposed to be negative while the text allocated with a 1 is supposed to be positive.

1.6.3 Consistent

The model needs to be fed data that has been processed consistently, if it was trained using a panda DataFrame then it needs a panda DataFrame to make predictions. A very important aspect though when choosing the DataFrame is that if the model is trained in English, then all the data need to be in English. By introducing other languages, you add a whole new set of words that may have completely different meanings. It also makes it difficult to eliminate stop words from the mix as their language needs to be specified.

Concerning this dataset, there is a very useful feature which is the [LANG] feature, this feature holds the language of the review, which means we can easily take only the English records.

1.6.4 Enough Data

If there are not enough records, then the model may struggle to grasp the real trend of the data. More often than not the data you are working with is a mere sample of the full population. For instance, the review data that we are using is not every review ever left on a cell phone. Therefore, the model must be able to grasp the trend of the data. Since the model needs to be able to handle data that it has never seen before. if the data it's not large enough that the model would likely not perform in the real world or potentially even testing.

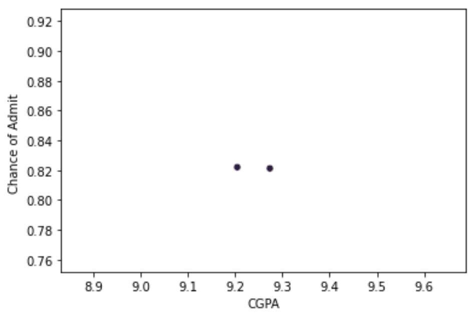


Figure 2 Too little data on a scatterplot

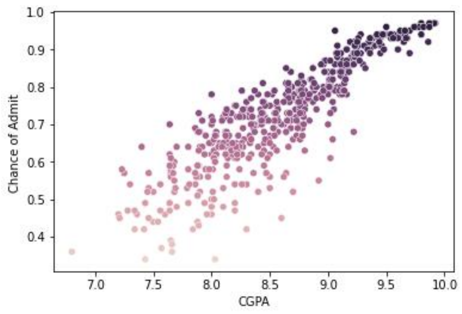


Figure 3 Enough data on a scatterplot

Luckily for this assignment, we are using a data set that originally has over 1.4 million records. After dropping all null values and only taking the English records we are still left with over 500,000 records to work with. After preprocessing and splitting the data into positive and negative reviews we are still left with over 120 000 negative reviews. this means that there are significantly more positive reviews than negative reviews, but there are enough negative reviews to hopefully train the model to grasp them correctly.

2 Data Wrangling

This section is broken up into how we perform data wrangling. There are six steps of data wrangling. Discovering, Structuring, Cleaning, enriching, validating, and publishing. The process describes the process we take when getting the data ready for the model.

2.1 Discovering / Data Exploration / Descriptive Analysis

When we explore the data, we are hoping to find trends and patterns that may prove useful to the company that tasked us. For text analysis in this context specifically, we want to see which words pop up in what context as well as which countries are thinking what.

2.1.1 Data Collection

The first step in the process is to collect the data. The dataset consists of six CSV files which are all from different regions. The format of these files is identical apart from the content of each record. This makes it easy to load each file individually.

The trick here, however, is to not combine each file into a single DataFrame before converting to a NumPy array. If this is done, then the conversion to NumPy will cause problems. The DataFrame needs to be loaded into the driver's memory all at once to convert it into a NumPy array, this means that large datasets cannot be converted this way. The trick here is to not concatenate the DataFrames but to instead concatenate the NumPy arrays since they are much better in terms of memory management. So, we break the data into 6 smaller DataFrames to load to NumPy and then concatenate the 6 NumPy arrays, bypassing the memory issue. This is convenient because the data is already split into 6 from the start, but this can be done on larger datasets by taking, for example, 100 thousand records at a time and loading them into NumPy arrays until you run out of records.

2.1.2 Visualising the Data

The first thing we want to do is see exactly which words are popping up the most. This will give us insight into what is popular amongst these reviews. Finding this information out allows us to make decisions based on what people are prioritizing. We can start by creating a word cloud of all the English reviews, both good and bad.

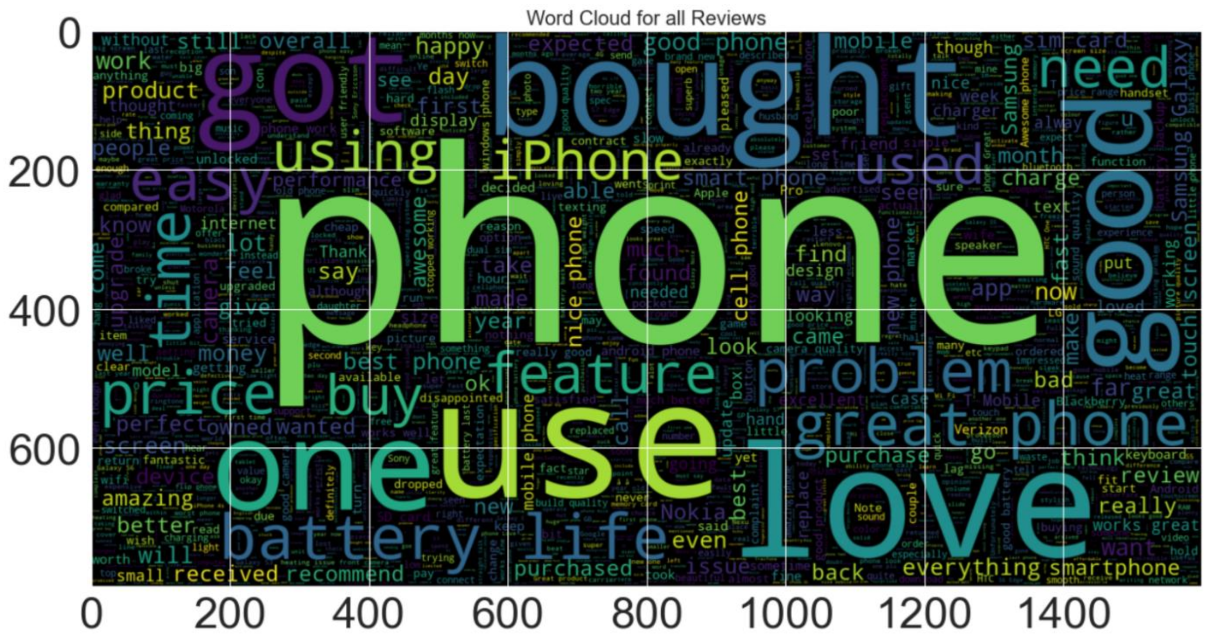


Figure 4 Word Cloud for English reviews

Immediately we see the words ‘Battery Life’ having quite a prominent presence within the phone reviews. However, we do not know if the words are spoken positively or negatively. To find out which is the case, we can segregate the reviews into good reviews and bad reviews and then create word clouds of each. We can start with the good reviews.

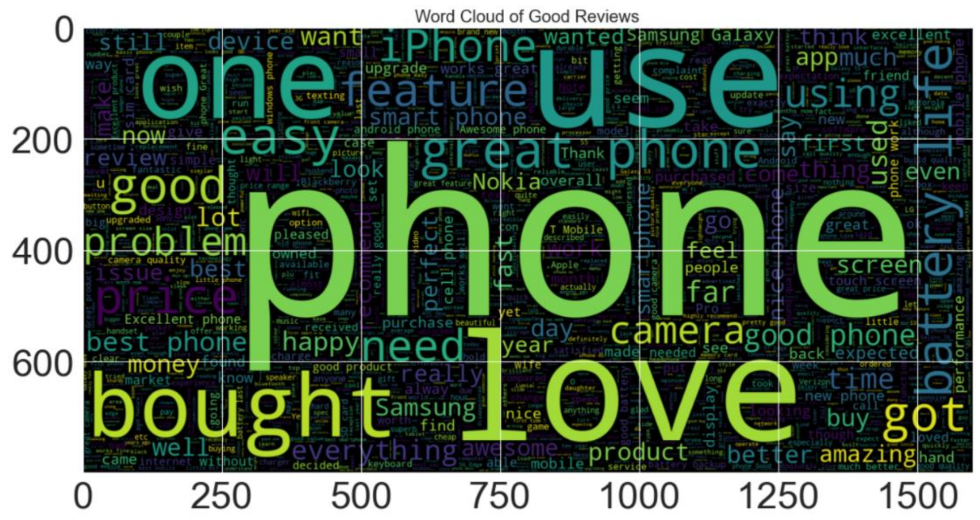


Figure 5 Word Cloud for Good Reviews

We can see that the words ‘Battery Life’ are quite common in good reviews. If you refer to the figure below, we see that the words are more present in the good reviews than the bad ones. Since the reviews are not for 1 singular phone, we can’t make a large generalization that all phones have a good battery life, but we can conclude that battery life is very important to people. Interestingly enough the words appear to be more prominent than even the word camera. This could indicate one of two things, either people value the battery life more than the camera, or the camera isn’t very good. Another thing we can conclude is that ‘battery life’ and ‘camera’ seem to be the only features mentioned. The rest of the words tend to be expressive words like ‘love’, ‘great’, or ‘good’ as well as arbitrary words such as ‘bought’ or ‘use’. These words will likely host greater weight towards being positive in the final model.

Let’s take a look at the negative reviews.

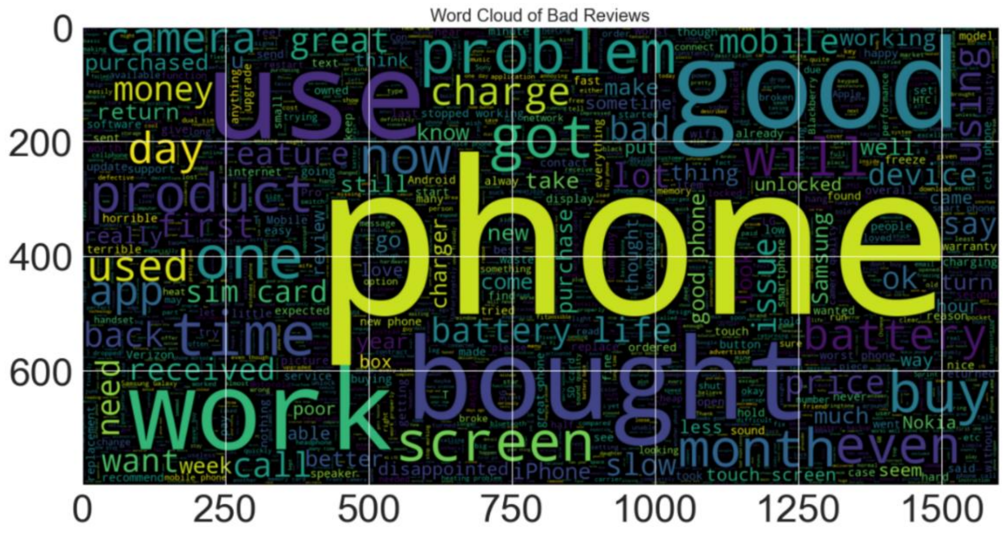


Figure 6 Word Cloud for Negative reviews

For the negative reviews, we get a lot of information. We see more features such as the charge, screen, and sim card. Since these words are only prominent, we can safely assume that they are being addressed negatively. Let's see what people are saying about the screen by programmatically grabbing a few records which mention the screen.

"I've now found that I'm in a group of people that have carried their phone in their pocket without problems until the S8. Day one screen has crack from being in my pocket. Bluetooth on my 1st trip struggles to stay connected."

"It is an extremely advanced and truly a Smart phone. Great apps, great look. BUT, the durability is that of a fresh egg. Don't consider this phone without insurance. The screen is fragile and one crack voids warranty. Won't buy another Samsung based on their lack of support for a poor design."

"This phone is admittedly breathtaking however it cannot survive even the simplest of mishaps. Both a close friend of mine and myself have broken ours within the first two weeks of receiving them. Mine feel from less than 2 feet and shattered the ENTIRE screen."

From the three reviews above we can deduce that the screen is too fragile and cracking easily. Had this data been from a specific phone, I would have heavily recommended that the screen be improved in terms of sturdiness.

Because we have access to the country feature there is also a lot more we can do with this data. For instance, we can see which countries have a higher tendency to leave reviews.

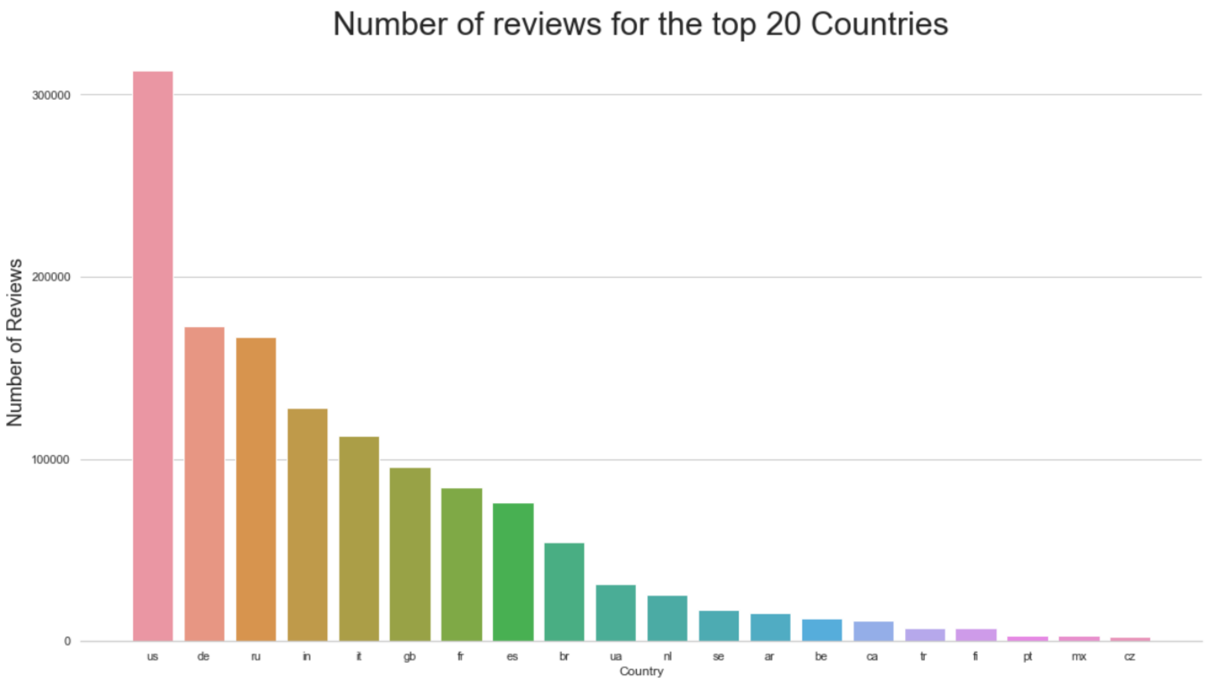


Figure 7 number of reviews per country

We can see that the vast majority of reviews are coming from the United States and that Germany and Russia also tend to leave quite a few reviews. If we can make a statement so bold as to say that the data was gathered fairly then we can assume that Americans are more likely to leave reviews. We can, however, break the data down even further. Let's take the percentage of how many of the reviews per country are positive.

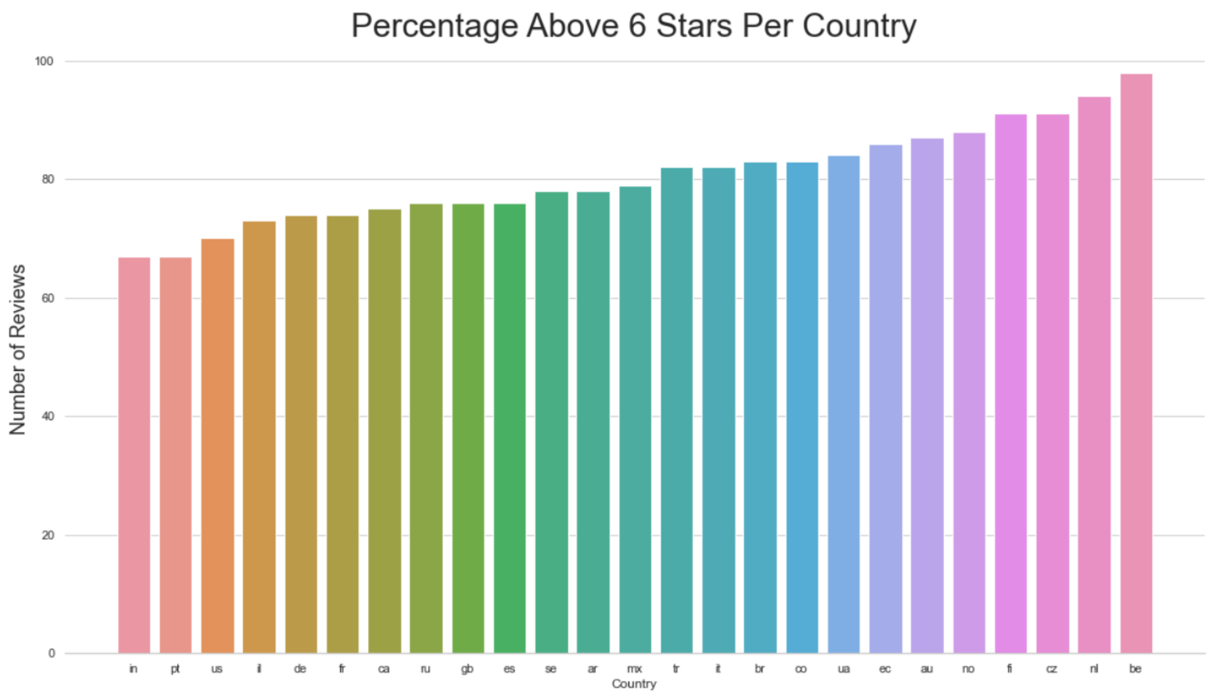


Figure 8 Percentage of reviews above six stars

We see that although the US has the most reviews, they are only sitting at a 70% positive review rate. This is the same for Germany and Russia which have slightly higher review rates but significantly fewer records. The records with over 90% positive review rates are records with too few records to care about. Given a lot more data in these specific regions we could come to very interesting conclusions. An example of this is Italy, which has the fifth most records but still maintains an 80% positivity rate. What companies could do is aggressively push to get their products reviewed in these countries with very high positive review rates. This could result in the phone receiving a much higher average review score. Alternatively, they could push a lot less aggressively in these countries with very low average review rates to achieve the same effect. Looking at the previous word clouds it might also be wise to prompt them to leave reviews soon after they get their phones. This gives them less time to discover its flaws. A good way of prompting the user would be to build a message prompt asking them to leave a review after they have used the phone for a day after purchasing it. They could also add a small incentive like R100 off your next uber eats meal. And then all phones can come with Uber eats pre-installed to make the deal more palatable for Uber.

As an example of how the data can be utilized to get more specific information, let’s only take records that have the Samsung Galaxy S8, and then see how many of each rating there is.

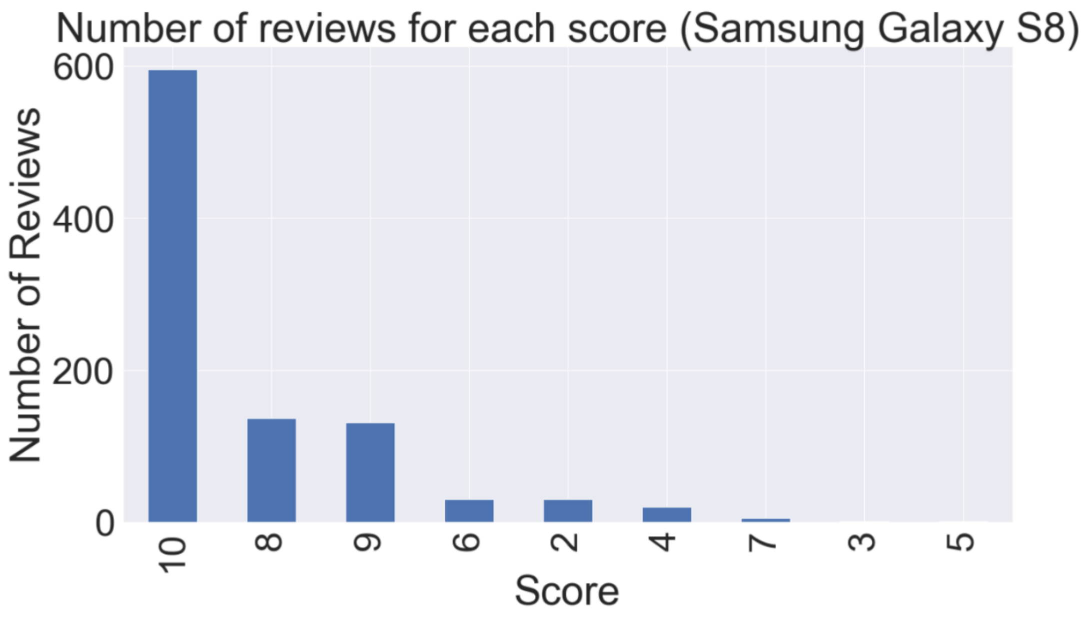


Figure 9 How many of each star for S8

The most we can get from this chart is that the Samsung Galaxy S8 is very well received.

2.2 Structuring

When we receive the dataset there are a lot of columns we do not need. Most of them are fairly clean and useful but there are a few things that need to be changed. The first change is to only take records that are in English. After that, we can convert all object-type features into string-type features for consistency and to avoid clashing in the future from methods that expect strings.

After that, we can take the necessary features which are the [extract] and [score] features. Lastly, we can run a script to add an extra column in to give us the sentiment based on the score. We can then remove the [score] feature which has been replaced by the [sentiment] feature.

What we are left with is a two-column DataFrame of only English records.

2.3 Cleaning

There is quite a lot that needs to be done in terms of cleaning the extract column specifically.

2.3.1 Drop Nulls

After the data has been structured, we can drop any null values left in the DataFrame as they may cause inconsistencies with the model down the line. After structuring and dropping all the records we are still left with over 500 thousand records, which should be more than enough to train our model.

2.3.2 Lowercase

Since capital letters and lowercase letters have different values, it’s important to baseline them. We don’t want the model reading different values for the words ‘problem’ and Problem. This inconsistency could limit the performance of the model if left unchecked. So we are going to run a script to make sure all the characters in the model are lowercase.

2.3.3 Remove unwanted elements

We don't want to use elements such as punctuation and HTML tags since they can get in the way of the model's performance, so we are going to remove those as well. A simple script can be run to perform this task.

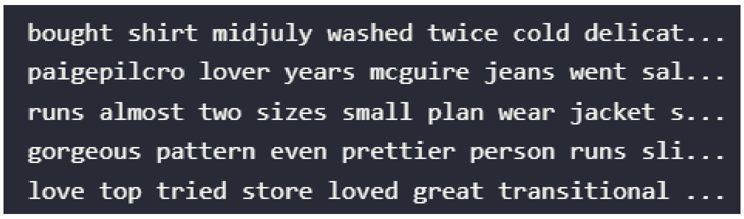
2.3.4 Remove stop words

Stop words are words that hold no value to the model. They are words such as 'I', 'this', and 'it's' are examples of stop words. These hold no weight for the model so we will remove them.

2.3.5 Stemming

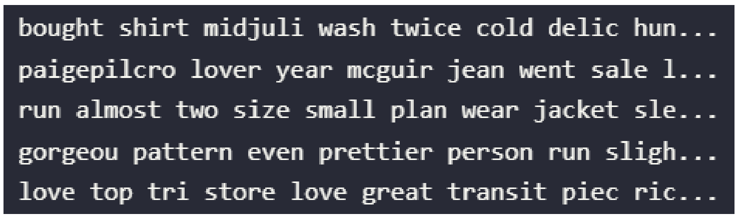
Stemming is the process of reducing all the words with the same root, down to their common form. This is done by removing the derivational and inflectional suffixes from the words (Lovins, 1968).

If we take a record, we can see this action in progress.



```
bought shirt midjuly washed twice cold delicat...
paigepilcro lover years mcguire jeans went sal...
runs almost two sizes small plan wear jacket s...
gorgeous pattern even prettier person runs sli...
love top tried store loved great transitional ...
```

Figure 10 Before stemming



```
bought shirt midjuli wash twice cold delic hun...
paigepilcro lover year mcguir jean went sale l...
run almost two size small plan wear jacket sle...
gorgeou pattern even prettier person run sligh...
love top tri store love great transit piec ric...
```

Figure 11 After stemming

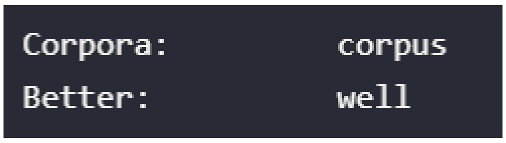
We see that certain words have very drastic changes, such as July which becomes Juli. This process will reduce the number of words that the model needs to learn by getting rid of unnecessary variations in words, such as plural.

Stemming will be applied to the dataset.

2.3.6 Lemmatization

Lemmatization takes a more realistic approach than stemming, instead of breaking the word down into its root form, it breaks it down into its normalized version. For example, stemming would take the words 'computes', 'computing', and 'computed' and break them down to be 'comput'. While stemming will take the normalized version of the word which is 'compute' (Plisson, et al., 2004).

It looks at the dictionary meaning of words before breaking them down.



```
Corpora:        corpus
Better:         well
```

Figure 12 Before and after lemmatization

The word corpora get converted to corpus while better gets converted to well. This method is often preferred over stemming as it considers the concept of the word as well. further reducing the number of words, the algorithm must learn which makes it a lot more reliable in production when it's likely people will use words the algorithm has never seen before.

However, in practice, this is not always the case. Sometimes reducing the word too far can have negative effects on the outcome of the data. Especially when it comes to ambiguity. The word 'better' might be more frequently used when referring to positive sentiment: "It does everything better", while the word well might be used for negative sentiments: "It doesn't work very well". These patterns might not make sense to humans, but the computer might find them. In this case, there is room for stemming to potentially be better.

2.3.7 Tokenisation

The neural network cannot work with raw text. It works best when fed a numeric variable. We can use the Tokenizer method from Keras to convert the string data into numeric Numpy Arrays. For more information on this, you can refer to the ipynb file provided.

2.4 Data enriching

To enrich the data, we will be utilizing SMOTE. This is a technique used when there are not enough records of a certain target. In our case, we have 350 thousand positive reviews and only 160 thousand negative reviews. This imbalance might severely limit the model's potential and could give us inaccurate information. An example of why it's important to keep an eye on the accuracy when the data is imbalanced: If we have 80% positive records and 20% negative records, and the model only guesses positive, it will have an 80% accuracy. Which is useless in production. So, we apply SMOTE to generate new data points for the minority target to compensate for their lack of them.

2.5 Validating the data

Since this section does not specifically deal with the model, model validation will not be discussed. However, it is important to make sure that the data is in the correct format and doesn't have any mistakes. This can be done with scripts such as the .info() command which will show you what type of data each column is stored in. Other probing methods such as the .describe() method or simply reading through the data quickly can point out any obvious mistakes. We can also visually see if the data is wrong by bringing up certain charts and seeing if they match what we expect.

2.6 Publishing

Once we are happy that the data is in the correct format we can look to publish it. This is the step where we save the data for the model's use. Before this step is completed the data must be in the correct format so that the preferred model can utilize it. In the instance of this task np.save() was used to save the cleaned NumPy arrays to a specified folder.

3 Building the Model

This section will look at how the model was built and evaluated as well as pay attention to why certain decisions were made and how we got around issues such as overfitting and underfitting. It will also pay attention to any improvements which can be made to the model.

3.1 Recurrent Neural Networks (RNNs)

This section pays attention to what a recurrent neural network is, why it's appropriate for text analysis, and how we get around its flaws.

3.1.1 What are RNNs

Recurrent Neural networks work similarly to other neural networks in the context that they have an input layer, hidden layers, and an output layer. Usually, a feedforward network will have a different weighting across each node in the network, but a recurrent neural network has the same weighting across each node. It gets its weighting from backpropagation and gradient descent (Baktha & Tripathy, 2017).

To understand how recurrent neural networks work it's important to understand how feedforward neural networks work.

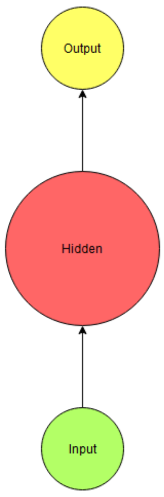


Figure 13 Feedforward Network

For a standard neural network, data is sent to the input layer, it then gets weightings in the hidden layer, and based on those weightings an answer gets outputted in the output layer.

Recurrent neural networks tend to add an extra step here. They add a looping mechanic that allows the network to loop through the hidden layer until it is happy.

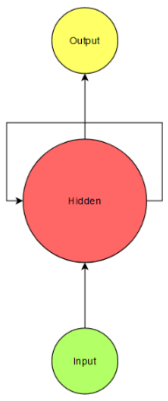


Figure 14 Recurrent neural network

Since the data can loop through the hidden layer it allows it to process an unknown number of inputs sequentially, which makes it fantastic at working with natural language. Recurrent neural networks also make use of a concept known as gradient descent which involves going through the model backwards and getting information based on the gradient of the data. Since the data is dependent on the previous gradient, we get situations like this:

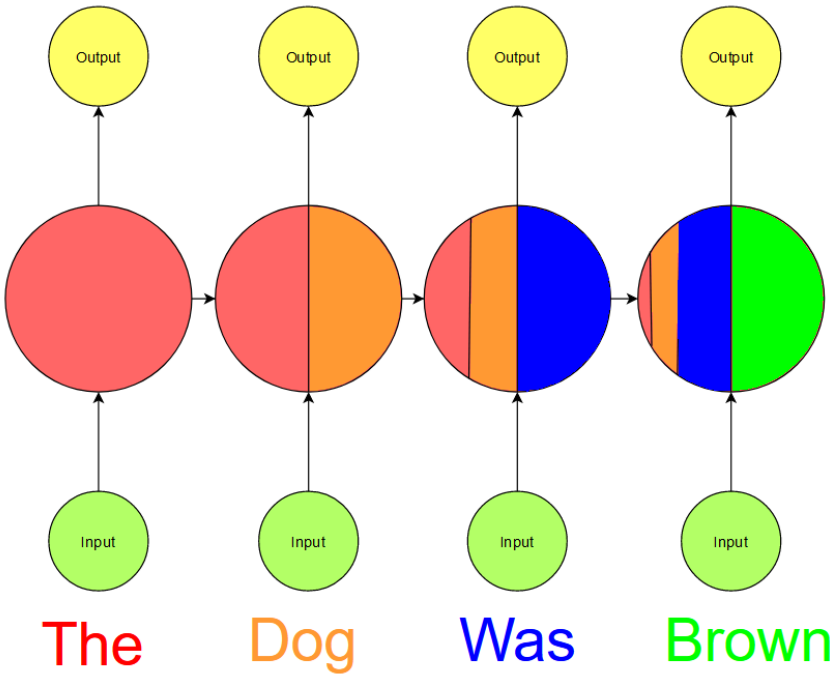


Figure 15 Effects of gradient descent

We see that the words are kept in order, but the weightings of the words at the start get so small due to the gradient of them being dependent on the previous node. This is a common problem with the Recurrent neural network, it's the fact that it has low memory as the early word weightings become so small to the point of being insignificant. However, this can be fixed with the use of the Long Short-Term Memory (LSTM) model.

3.1.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) models function almost identically to standard RNNs, however, there is a key difference in the fact that LSTM models introduce a logic gate that allows the model to know which words to remember and which words to discard. There are three gates within the model to control which information goes through, and which information gets discarded. There is the input gate which looks at the current input and decides what should go through and what should not, there is the forget gate which determines how much of the previous state should be allowed to pass, and finally the output gate which decides how much this current node should affect the entire external network (Baktha & Tripathy, 2017).

By doing this the model can remember data for long or short periods, hence the name. It also means that important words will not be drowned out by gradient descent and will maintain their weighting appropriately.

This will be the chosen model for our model due to this reason.

3.1.3 RNNs and Sentiment Analysis

Recurrent Neural Networks are very widely used for sentiment analysis and natural language processing in general as they can model temporal sequences with variable lengths (Baktha & Tripathy, 2017).

To put it simply the ability of a recurrent neural network to take the order in which words are given into account makes it good at natural language processing. When a different model gets the sentence “I hate how much I love this”, it might get confused between the word “hate” being negative and the word “love” being positive. The weightings of these two words might bring the model to a standstill, forcing it to guess based on how often one of those words previously appeared in a specific context. However, a recurrent neural network can take the order of words into account, and with enough data, it might realize that this specific order of characters is likely positive.

This is why Recurrent Neural Networks are so dominating with sentiment analysis as they are capable of recognizing sentences and not only words.

3.2 Prepping the data

Now that we have decided on our model, we can start looking at what needs to be done to the data to train the model. **The bag of words and tf-idf methods will not work as the sequence of the data is lost with them.** But we still need a way to get the data into a numeric form, which is where the Keras tokenizer method comes in.

3.2.1 Keras Tokenizer

The Keras tokenizer takes each word and assigns a number to it, and then that number will stay consistent throughout the entire model. This means if the word ‘hello’ is assigned the number 57, and the word ‘world’ is assigned the number 83, then the words will appear in the array as [57,83]. This allows us to keep the sequence of the model consistent which works well for the model.

The Keras tokenizer will be used to convert all the words into numeric form for the model to read.

3.3 Building the Model

The first thing we do is start creating a neural network with the following code.

```
model = Sequential()
```

The next thing we want to add is an embedding layer. This is the layer that looks at all the input and then codes it into something of defined length for the model to read. It acts as an input layer of sorts and is necessary.

The Max features option will take only the top X words, the embedded dimension is the number of dimensions in which you want to represent your word vectors and the input length is the length of the input sequences.

```
model.add(Embedding(max_fatures, embed_dim,input_length = X_train.shape[1])) #
```

A spatial dropout layer is a form of regularization, which drops out entire 1D feature maps instead of individual elements (Lang, et al., 2019). This helps with independence between features.

```
model.add(SpatialDropout1D(0.4))
```

Lastly, we add the actual LSTM layer and the Dense Layer. The LSTM layer is the actual layer that handles most of the weighting which was discussed above, it will loop through the nodes constantly dropping needless inputs and deciding which one gets through. If you don't specify the activation layer manually, it will be set to tahn and utilize the CuDNN method.

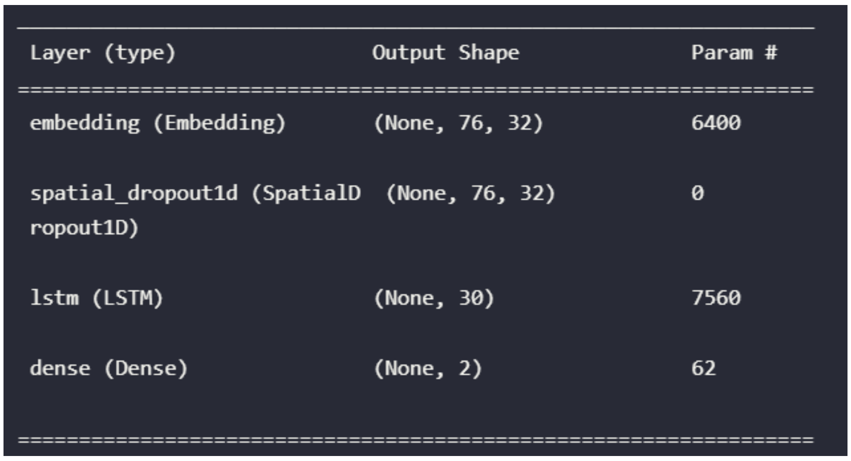
The Dense layer is the output layer that takes all the weightings and figures out if it's a 1 or 0. We set the activation of the dense layer to sigmoid since we want a binary output.

```
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0))
model.add(Dense(2,activation='sigmoid'))
```

Finally, when we compile the model, we want to use the adam optimizer as it is well known to be a good all-round optimizer which also helps fight against overfitting.

```
opt = tf.keras.optimizers.Adam(lr=1e-3, decay=1e-5)
model.compile(loss = 'sparse_categorical_crossentropy', optimizer=opt,metrics
= ['accuracy'])
```

Now that we have compiled the model, we can look at the schema for it.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 76, 32)            6400

 spatial_dropout1d (SpatialD  (None, 76, 32)           0
 ropout1D)

 lstm (LSTM)                 (None, 30)                7560

 dense (Dense)               (None, 2)                 62

=================================================================
```

Figure 16 Model Schema

We allow 76 input sequences into the embedding layer, the spatial layer accepts each of those 76, and it gets whittled down to just 2 output layers, a layer for 0 and a layer for 1.

Let's use these variables

```
max_fatures = 200 | embed_dim = 32 | lstm_out = 30 | batch_size = 128
```

3.4 Testing the Model

This section looks at how well the models performed with the different types of data they were given.

Stemming with SMOTE			
	Precision	Recall	F1-Score
0	0,89	0,77	0,88
1	0,8	0,91	0,84

Accuracy	0,84

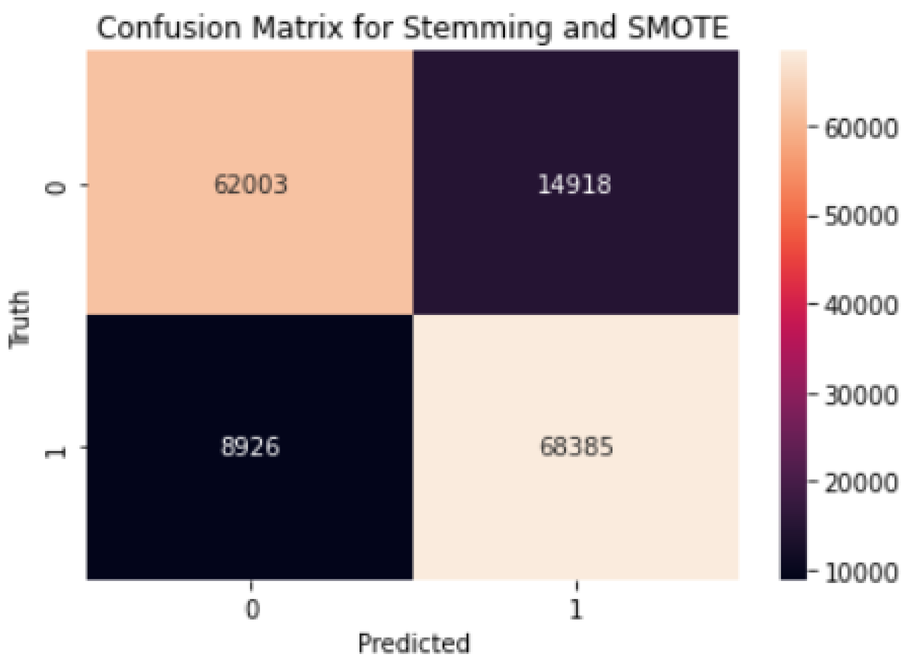


Figure 17 CM for Stemming and Smote

Lemmatization with SMOTE			
	Precision	Recall	F1-Score
0	0,8	0,81	0,8
1	0,8	0,8	0,8

Accuracy	0,8

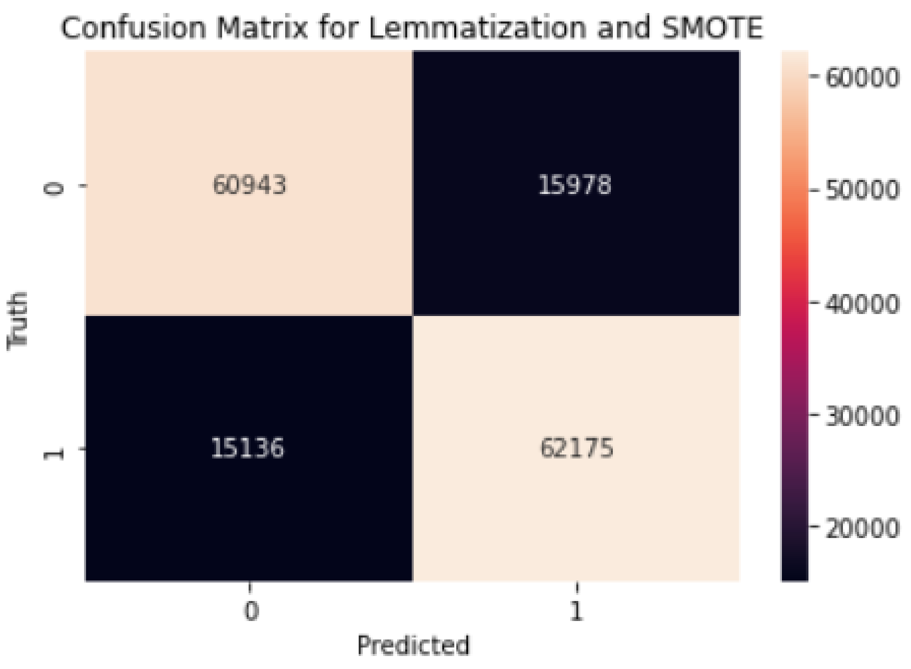


Figure 18 Lemmatization and Smote

Stemming and No SMOTE			
	Precision	Recall	F1-Score
0	0,74	0,64	0,69
1	0,85	0,9	0,88

Accuracy	0,82

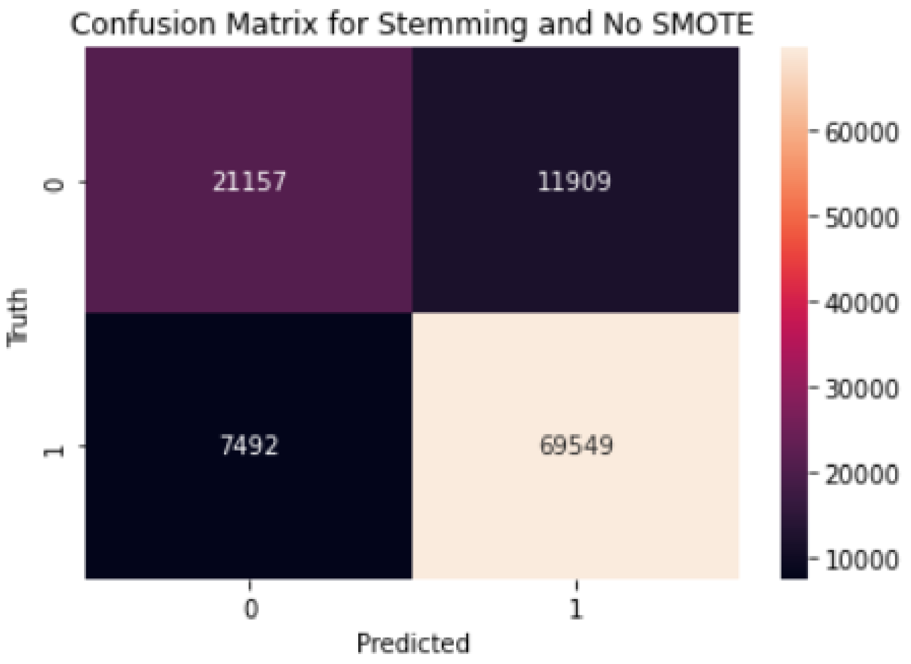
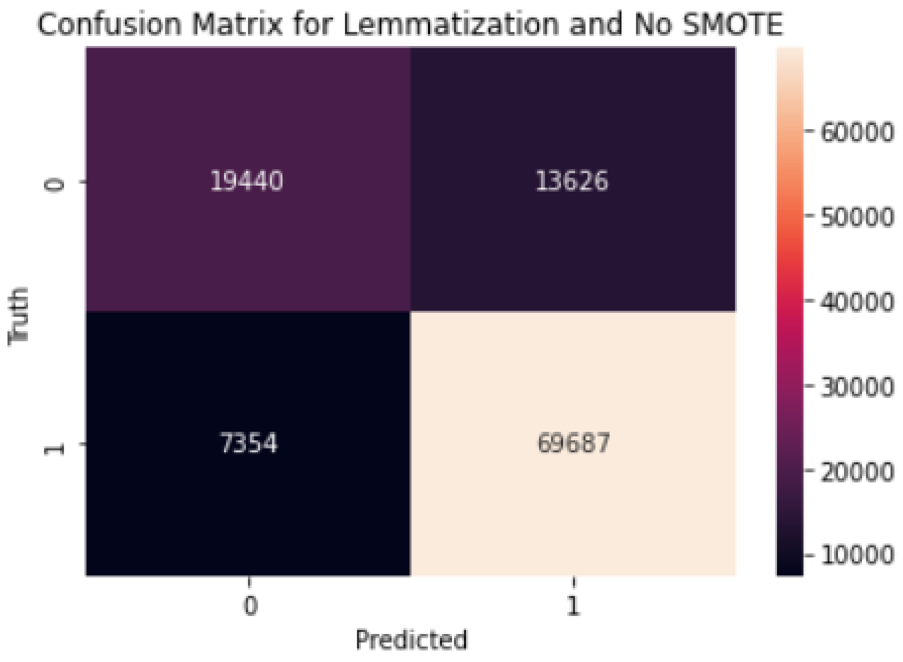


Figure 19 CM Stemming and no Smote

Lemmatization and No SMOTE			
	Precision	Recall	F1-Score
0	0,8	0,79	0,8
1	0,8	0,8	0,8

Accuracy	0,8



3.5 Interpreting the results

It seems like the stemming with SMOTE performed the best so let's focus on that. In both cases with and without SMOTE, stemming outperformed lemmatization. This is likely because lemmatization might change words that have more value not being changed. Refer to section 3.3.6 for more information on this. SMOTE also seems to have given a bit of a boost since the additional negative reviews likely helped the model predict them, we see this since the model has almost a 90% precision rate with negative reviews. But the precision can be a bit deceiving since now the model is struggling to predict positive values.

So only around 9k records were predicted as negative when they were positive. But almost 15k were predicted as positive when they were negative. So, the model is very good at predicting negative records but not very good at predicting positive records.

This is a big problem since the vast majority of records are likely going to be positive. So, if the model isn't very good at predicting positive records, then it's going to flop hard in production. This is a downside of using SMOTE. If we look at the Stemming without SMOTE, we see that the precision on positive records is much higher. So, it would likely be wiser to go with this method. It isn't fantastic though since now the negative records are struggling. So we need to look into making the model more accurate.

3.6 Hyper Parameters

The first thing to note is that the max features for the input layer are likely too low at 200. This means that only the first 200 records are going to be processed. Let’s change this to 2000 to match the max features of the tokenizer. This means that the tokenizer will take the most important 2000 features and then pass all of those features to the input layer. This means that more words can be recognized. Before it was possible that a word that is positive or negative simply wasn’t in the top 200 records when the model was being trained.

Next, we will increase the embedded dimensions to 128 and will allow for 200 LSTM units. Lastly, we will increase the epochs to 15 to allow for further training. After a lot of tuning these were the best parameters found.

```
max_fatures = 2000 # we will only take the top 2000 words
embed_dim = 128 # we will embed the words into 128 dimensions
lstm_out = 200 # we will use 200 LSTM units

model = Sequential() # create the model
model.add(Embedding(max_fatures, embed_dim,input_length = X_train.shape[1])) #
add the embedding layer
model.add(SpatialDropout1D(0.4)) # add the spatial dropout layer
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0)) # add the LSTM
layer
model.add(Dense(2,activation='sigmoid')) # add the output layer
opt = tf.keras.optimizers.Adam(lr=1e-2, decay=1e-2)# create the optimizer
model.compile(loss = 'sparse_categorical_crossentropy',
optimizer='adam',metrics = ['accuracy']) # compile the model
print(model.summary())

batch_size = 128
model.fit(X_train, y_train, epochs = 15, batch_size=batch_size, verbose = 1)
```

This model resulted in the following accuracy scores.

Stemming with No Smote and Hyper Params			
	Precision	Recall	F1-Score
0	0,83	0,77	0,8
1	0,9	0,93	0,92

Accuracy	0,88

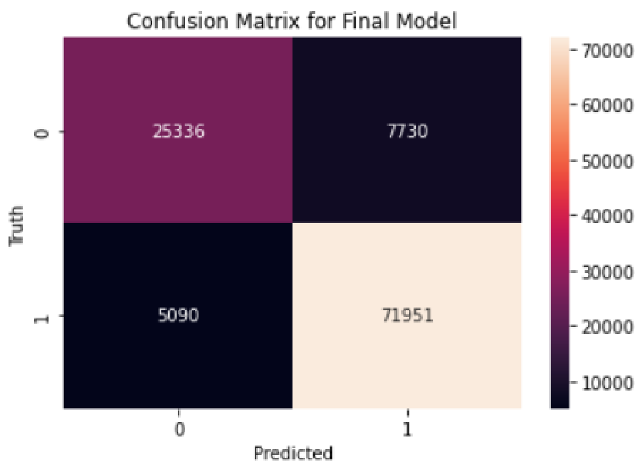
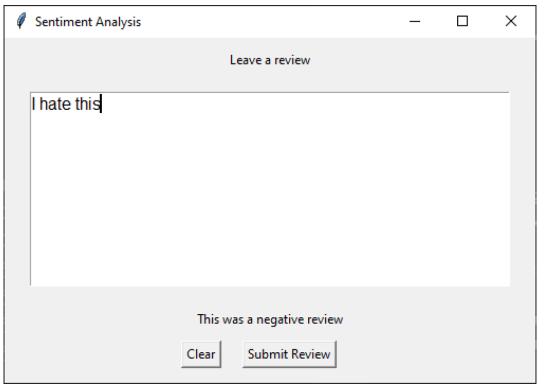


Figure 20 CM for final model

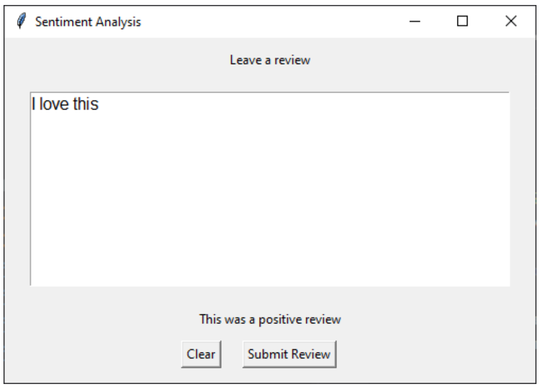
We can immediately see a massive difference in the scores, the matrix is much better at predicting positive reviews while maintaining the ability to still accurately spot negative ones. Let's play around with it and see how well it is performing with unseen data.



I hate this

This was a negative review

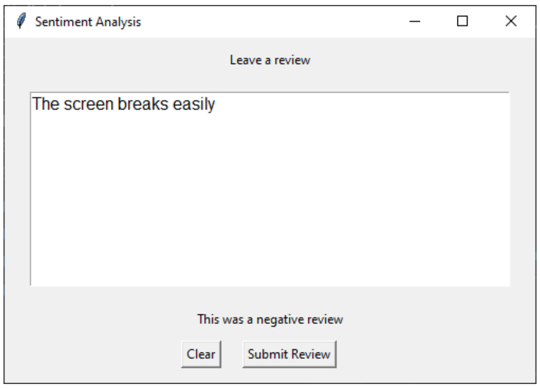
Figure 22 Negative test 1



I love this

This was a positive review

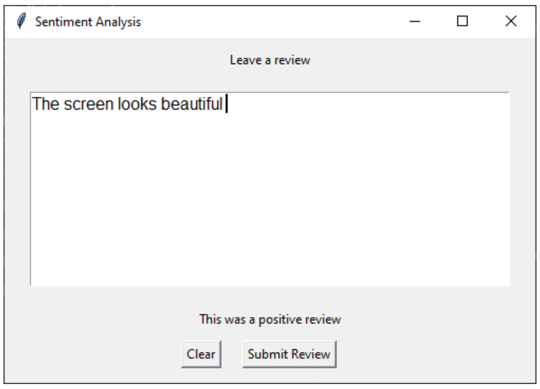
Figure 21 Positive test 1



The screen breaks easily

This was a negative review

Figure 24 Negative test 2



The screen looks beautiful

This was a positive review

Figure 23 Positive test 2

The model seems capable of correctly guessing the sentiment of records.

3.7 Overfitting and underfitting considerations

Overfitting often occurs when a model is too closely fit to the training data. This will likely prevent the model from predicting accurately as it is affected by the noise in the data set. A underfit model struggles to make accurate predictions as it cannot recognize the trend of the data (Muller & Guido, 2017).

3.7.1 Overfitting

There aren't many easy ways to deal with overfitting. Just make sure that the data is realistic and not generated to avoid it failing in production. The LSTM model can make use of the 'adam' optimizer which is pretty good at reducing overfitting.

3.7.2 Underfitting

The biggest contributor to underfitting is having bad data from the beginning. If there is just no link between the sentiment and what people are typing, then there isn't much you can do. Luckily the data seems to not struggle with this problem at all as it would be unlikely to have this happen in a review scenario. People who leave low ratings aren't going to speak positively about the data. The biggest change that could happen here with the dataset I used, in particular, is that the point between 0 and 10 to assign 0 and 1 could be adjusted. Currently, people who leave a review lower than 7 are considered negative reviews, if this hadn't worked then the bar could be lower to reviews lower than 6.

If there isn't enough data, then the model will likely struggle to learn. If this is the case, then you need to get more to work with. Luckily the records we were using allowed for over 500 thousand records with lengthy reviews to be analysed, so there was no problem there. If there wasn't enough of a specific record, then SMOTE could be attempted to generate more.

If the data is unclean, then you may also find that the model struggles a bit. If there are too many null values or if not all the records are in the same language, then the model could be completely underfitted as a result. So, it's important to make sure that the data is **consistent**.

Bad parameters can also be a cause of underfitting, as seen above. If the parameters aren't correct, then the model will try its best but simply might not be able to figure anything out with the tools it was given.

3.7.3 Testing for overfitting and underfitting

It's easy to see if the data is overfit or underfit, the trick is to break your data into training data and testing data. Your model gets built on the training data and you use your testing data, which the model has never seen, to simulate a real-life scenario. If the accuracy of these tests come back looking good, then it's safe to say you are in the clear.

3.8 Validation

Recurrent Neural Nets are quite easy to validate since they are constantly doing so when they train themselves, constantly getting accuracies and using it to retrain itself over and over again. We can see this in progress if we set verbose to 1 when we fit the model, which will allow us to get a live update on how the data is performing. We can also increase the size of the epochs which adds even more layers of validation as it makes the model train itself further.

Alternatively, or additively, we can use confusion matrixes with data the model has never seen before, which will simulate unseen data to see if it's working well.

3.9 Alternative Models

The LSTM model is not the only possible model that can be used for sentiment analysis. Two other notable alternatives can be used.

3.9.1 Naïve Bayes Classifier

The Naïve Bayes classifier is pretty good at sentiment analysis for a standard model. In my previous report, the Naïve Bayes outperformed logistic regression, random forest, and Support vector machines. It has the obvious advantage of being able to utilize the term frequency-inverse documentation method as it doesn't look at the sequence in the text. The obvious disadvantage is that it doesn't look at the sequence of the text. In testing this model the algorithm didn't perform too badly with an accuracy of 80%, with the right cleaning and tuning the model could outperform the LSTM model.

3.10 Gated Recurrent Unit

GRU functions very similarly to the LSTM model, except it, only has 2 gates within each node. A reset gate looks at how to combine new and old information and an update gate which determines how much of the information should pass (Baktha & Tripathy, 2017).

In a test run by (Baktha & Tripathy, 2017) it was discovered that the GRU method outperformed the LSTM method.

In internal testing, the GRU method was capable of reaching an accuracy score of 88 within 3 epochs compared to the LSTM which settled on 88 after 15 epochs. So, it's fair to say that the GRU with the right parameters could outperform the LSTM method here.

4 Conclusion

We started with a large dataset of over 1.4 million records and whittled it down to 500 thousand useful ones. Descriptive analysis was performed on the dataset, and it was found that positive reviews are quite a bit more common than negative reviews, especially on the Samsung Galaxy S8. It was also found that Italy and certain other countries have very high average ratings while the United States has a very low average rating score.

Sentiment analysis was conducted using the LSTM method from the Keras library. Lots of records were cleaned, tokenized, and fed into the model as training data, completely separated from the testing data which was used to verify the results. The results after hyper-tuning showed that stemming without SMOTE was the ideal choice for the model as SMOTE negatively impacted the model's ability to sense positive sentiment. Since the model will be used in an environment with predominantly positive reviews, evident from the descriptive analysis, it's wiser not to utilize it.

It was also found that GRU could be an even better method than LSTM as it was able to replicate its score in a fifth of the epochs.

The model ultimately performed well, being able to sense both positive and negative reviews with very high accuracy, which can be used to make better business decisions.

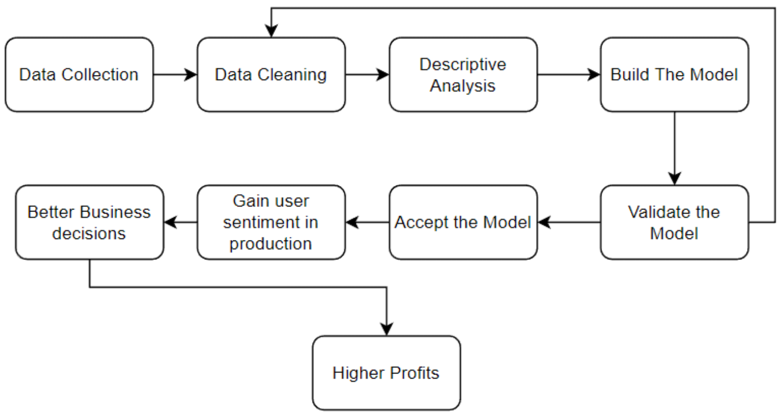


Figure 25 Sentiment Analysis Process

5 Recommendations

From the descriptive analysis side, there is quite a lot we can get from it. There are also quite a few interesting business decisions that would be made off of larger amounts of similar data. For instance, Italy had quite a lot of reviews yet maintained a high average review score. It would be in a phone company's best interest to be more aggressive when trying to get a review from Italians. One thing they could do is go to Uber and say "Listen, we will release our next phones in Italy with Uber eats pre-installed, if you allow us to give every Italian user a 10-euro coupon on their next order when they leave a review for this phone.". Uber says "sure" and now the company has the means to offer an incentive that isn't too pushy or obvious. Other incentives might come across as annoying to the public, such as locking the newest software update behind a review. But a coupon is perfect. After that, it's all about timing. The descriptive analysis showed that a lot of the negative reviews were about the screen breaking, so it's important to prompt for a review late enough that they have had enough time to enjoy the phone, but not too late that they have time to find its flaws. Around 2 to 3 days after the user buys the phone it should prompt them with the offer. The logic can be flipped where you might not want to push for user reviews in countries with low percentage review scores, but it's important not to hide the feature altogether because the people who want to leave a negative review will be more determined to find the option to do so than the people who wanted to leave a positive review.

With the ability to sense positive and negative responses a company suddenly has access to a large amount of potential data it might not have previously been able to acquire. For instance, the model could be used to search through tweets with the hashtag #SamsungGalaxyS8 and find out if people are being positive or negative about it. With Twitter API you can also potentially gain geographic information on these tweets as well, which could give a company a massive amount of insight into what's going on. If Canada hates the phone, then don't market as much there, but if India can't get enough of it then market it more in that region.

It can allow the company to make fast and real-time decisions. If Apple has a conference and they announce that they want to remove the charging port from the new iPhone, and the vast amount of social media attention is negative, then it's likely not a great idea to go through with it. The advantage of the model here is that the information can be processed so quickly that they might be able to realize their mistake and change it before it's too late.

6 References

Baktha, K. & Tripathy, B., 2017. Investigation of recurrent neural networks in the field of sentiment analysis. *International Conference on Communication and Signal Processing (ICCSP),* pp. 2047-2050.

Cote, C., 2021. *Predictive analytics.* [Online] Available at: https://online.hbs.edu/blog/post/predictive-analytics [Accessed 22 April 2022].

Lang, C., Steinborn, F., Steffens, O. & Lang, E., 2019. Applying a 1D-CNN network to electricity load forecasting. *International Conference on Time Series and Forecasting,* pp. 205-218.

Lovins, J., 1968. Development of a stemming algorithm.. *Mech. Transl. Comput. Linguistics,* 11(2), pp. 22-31.

Medhat, W., Hassan, A. & Korashy, H., 2014. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal,* 5(4), pp. 1093-1113.

Muller, A. & Guido, S., 2017. *Introduction to Machine Learning with Python.* 1st ed. United States of America: O'Reilly.

Plisson, J., Lavrac, N. & Mladenic, D., 2004. A rule based approach to word lemmatization. *Proceedings of IS,* Volume 3, pp. 83-86.

The Independent Institute of Education, 2022. *Data Analytics Module Manual ,* s.l.: s.n.