Webcrawler and NLP System – Capstone Report

Does the rating of accommodation reflect in their reviews?

A reflecting on Tripadvisor reviews

Overview (488 words)

The popular growth of social media and increased reliance on word-of-mouth recommendations has provided a treasure trove of personal opinions based on personalized experiences. These experiences whether real or not real become supportive truth for those wishing to share experiences. Therefore, reviews can make or break a business depending on the weight the review carries and the influence the reviewer has upon the intended audience.

The hypothesis for this report is that there is no relationship between wording in a review and the actual rating that is given by the reviewer.

To assist in the process of proving the hypothesis to be true this report will, through a corpus of data scraped from Tripadviosor.com.au and processed using natural language processing (NLP) and machine learning (ML), predict the reviewer rating of a reviewer from the words used in their review. The methodology is to divide the corpus into 2 sets:

- training data representing the review and the rating; and
- test data representing a portion of data that will be used to predict the review.

The test data will be stripped of its reviewer rating then processed through the ML model to ascertain a prediction of what rating would be expected and then compared to the actual rating given. Variances between the 2 is classified as the accuracy of the model. The models accuracy will also be assessed using root squared deviation to determine the standard deviation of errors of the model (how spread out the errors are).

Using a similar process Chiny et.al[1] identified that determinants of satisfaction was crucial to their study of short-term rental accommodation. Using NLP and ML their outcomes were that the customers (who were categorized) were not equally interested in satisfaction metrics (precision, cleanliness, check-in, communication, location, and value). It was also found that disparities were noted for the same indicator depending on the category to which the customers were classified. The outcome of this study was suggested improvements to the rating system adopted by Airbnb to align categories to clients.

Suzuki et.al[2] reviewed the process of natural language processing in reviewing guests' sensitivity to service and product in reflection of prices paid. Whilst the original concept of this report was to justify rates of hotels when compared to reviews to ascertain whether customers viewed, they received values for money the scope changed when the corpus was collected. Consequently, this referenced body became redundant to the study.

Alternatively, Chang et.al[3] study in using deep learning and visual analytics to explore hotel reviews and responses redefined the scope to the outcome – using deep learning and NLP to investigate guest reviews and determine from these reviews' sentiment of the reviewer; profiling the reviewer and their rating information and finally assisting management in prioritizing responses and develop response strategies.

This study assisted in the framework of this report and narrowed the scope to the final research body: sentiment analysis, rating analysis, sentence construction and word usage analysis.

Webcrawler (977 words)

The corpus of the study was extracted from Tripadvisor. Originally Booking.com and exedia.com.au were to be used to cross reference review data however both sites utilise a feed approach to a review which attempts to categories review responses which filters what would otherwise be an uncensored response.

Additionally, the concept was to extract from only hotels with 500 reviews or more, yet the study took a more structure approach. The study focused solely on hotels and resorts within the Cairns city (figure 1) which totaled 14.

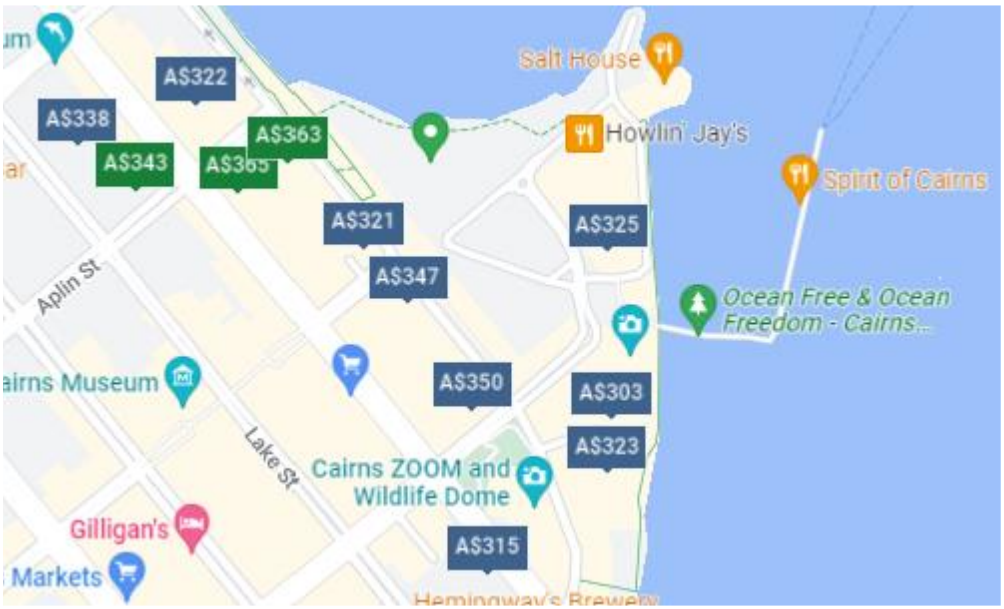


Figure 1: Cairns City – flags locating 12 of the 14 hotels/resorts under study (source: Google Maps)

Table 1 lists the hotel and associated information from TripAdvisor:

code	filename	urlid	Reviews	Pages	Rack_Rate
pc	pullmancairns.csv	d257801-Reviews-Pullman_Cairns_International	2656	266	$331
prc	reefcasino.csv	d256432-Reviews-Pullman_Reef_Hotel_Casino	2357	236	$399
sl	shangrila.csv	d256449-Reviews-Shangri_La_The_Marina_Cairns	3021	303	$360
hil	hilton.csv	d255653-Reviews-Hilton_Cairns	2740	274	$350
me	mantraesp.csv	d256436-Reviews-Mantra_Esplanade_Cairns	2347	235	$329
cb	bailey.csv	d18937319-Reviews-Crystalbrook_Bailey	175	18	$345
cf	flynn.csv	d19937778-Reviews-Crystalbrook_Flynn	132	14	$365
cr	riley.csv	d14450704-Reviews-Crystalbrook_Riley	698	70	$425
mt	mantratrilogy.csv	d489119-Reviews-Mantra_Trilogy_Cairns	2763	277	$325
nco	novotel.csv	d256455-Reviews-Novotel_Cairns_Oasis_Resort	279	28	$401
hildt	doubletree.csv	d255381-Reviews-DoubleTree_by_Hilton_Hotel_Cairns	2541	255	$340
ben	benson.csv	d23470282-Reviews-The_Benson_Hotel	78	8	$310
royal	royalharbour.csv	d256453-Reviews-BreakFree_Royal_Harbour_Cairns	1044	105	$419
oaks	oaks.csv	d21093788-Reviews-Oaks_Cairns_Hotel	157	16	$359

Table 1: Hotels information (rack rate was as of day of extraction)

The code given to each hotel was used as a category in the analysis dataset to isolate reviews by hotel, the filename is the actual csv download from the web scrap which the urlid is the identify for each of the properties within TripAdvisor's website. The last 3 columns were a construct gleaned from the webpages of each of the hotels within the study.

Tripadvisor has a paid API which is offered to partners which was not used. The alternative was found using selenium. Selenium is a python library which accommodates web scraping with the proviso that the user can navigate forward through pages if needed. In this instance reviews were a container (sub section) of the title page of a hotel. There were 10 reviews listed per review container and to get to the next page there is a next page button at the base of the container. It was this element that was critical to isolate and control programmatically.

The hierarchy of the compilation of the web scraper started with a csv which contained the contents of table 1, a list of elements that could be extracted from the webpage when scraped and a process of cleaning the scraped data to then be filed for NLP processing.

URL's of Tripadvisor has the following naming convention:

URL prefix			Hotel Identifier	URL suffix		
https://www.tripadvisor.com.au	Hotel_Review	-g255069-	d256432-Reviews-Pullman_Reef_Hotel_Casino	_Cairns	_Cairns_Region_Queensland.html	#REVIEWS

Table 2: TripAdvisor's url naming convention

Using the structure stated in table 2 the scraper could cycle through the hotels listed by changing a single hotel identifier rather than listing all the url's in long hand.

```
# setup defaults for scraper
urlid = df['urlid'].iloc[selected_hotel]
file = df['filename'].iloc[selected_hotel]
num_page = df['Pages'].iloc[selected_hotel]-1
path_to_file = os.path.join(dir_name,file)
num_page, path_to_file
```

Table 3 Hotel defaults for web scraper

Additionally, the number of pages to cycle through by the scraper and the path and filename to file the output all originates from this default section.

Focusing in on TripAdvisor's webpage, it was decided that each element of the review section was important. If not for modelling at least as a source of future analysis. Inspection of the html of the page found the following keys phrases were required for extraction purposes:

- review header information: class='sCZGP which included elements:
  - class 'CRVSd' = rating
  - class 'MziKN' = reviewer details
- Review detail information: class = "reviewid" which included elements:
  - class 'review-title' = title of review
  - class 'QewHA H4 _a = review
  - class 'ui_bubble_rating bubble_' = reviewer details
  - class 'teHYY _R Me S4 H3' = stay date

The web crawler was designed to cycle through each page of the hotel selected and extract the above contents, store it in an array called result and then at the completion of each page append the array to a list tilted reviewDetails simultaneously moving to the next page to repeat the process until the last page (number shown in pages field of table1). A timer and number of pages being processed is displayed for the user to monitor the crawler's progress.

The fields reviewer, reviewerdetails, and staydate all had data which needed to be extracted. In the case of staydate it was a simple clean of the identifier for the field while reviewer helded the review date in it and reviewer details had the number of contributions and votes the reviewer has.

Using Regex, the fields were split out during the wrangling stage to produce the following schema:

Pre wrangling	Rating	Title	Review	Reviewer		Reviewdetails				Staydate
Post wrangling	Rating	Title	Review	Reviewer	Date_reviewed	Reviewerdetails	Contributions	votes	Hotel	Staydate

Aside from the hotel list there were no other requirements for external data to be processed that would have enhanced the data set. The following diagram illustrates the web crawler's methodology:

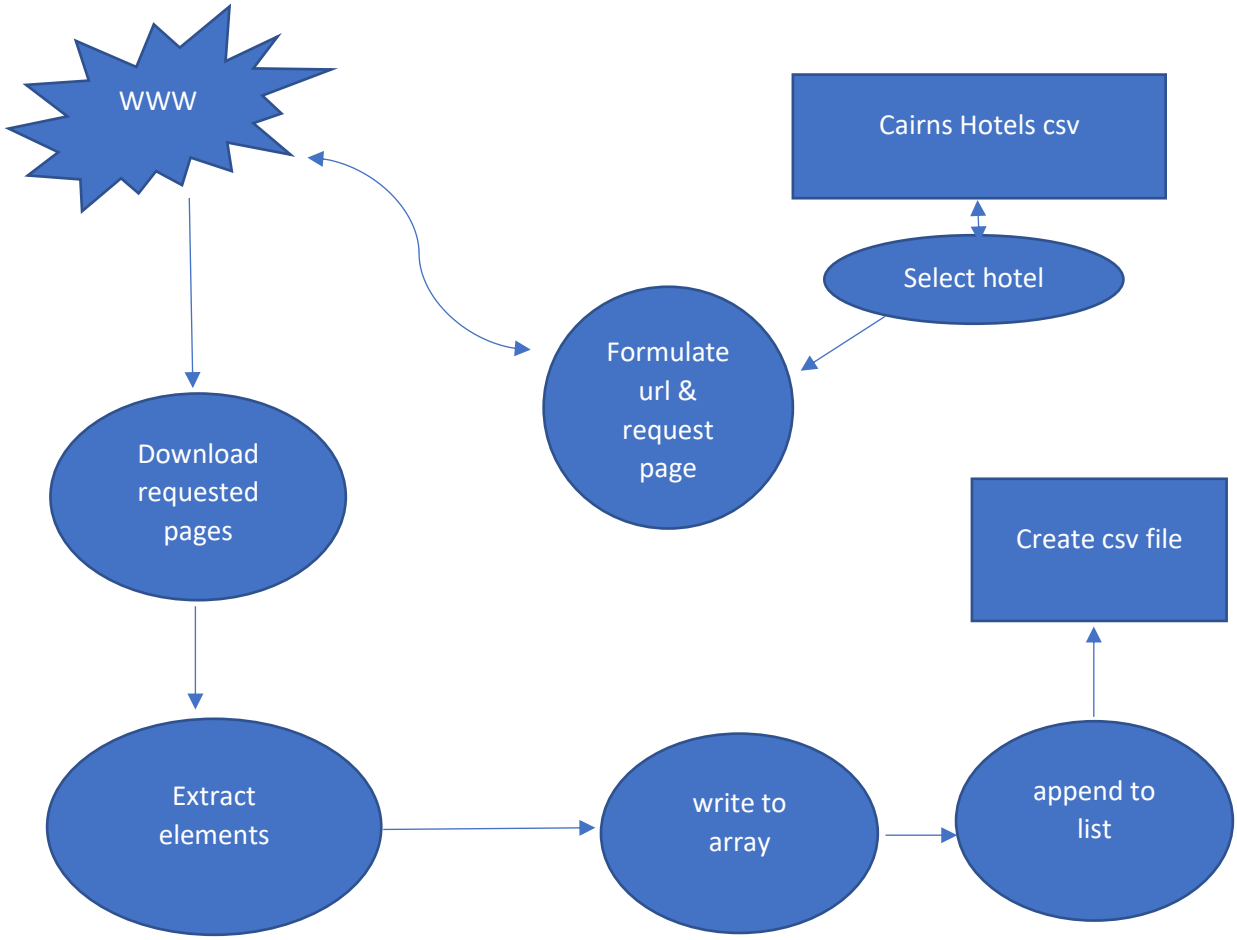


Figure 2: Scraper methodology

One aspect of the scrape that was an obstacle was when an element was empty. When this event happened, the crawler would stop and process no more reviews even if there were more to scrap. To overcome this an except clause was put into each elements extract phrase. The crawler instead of assuming the element was always populated it would first try the element and if the element had contents would populate the array and if not, the crawler would call a *NoSuchElementExemption* which would then skip the element and proceed to the next step of the processor.

Each page took 0.36 seconds so a run of 2000 reviews took approximately 11 minutes using a relatively old laptop and suburban internet connection.

The only limitations to the web crawler were focused on setup time and familiarity to the software - in other words user based. With more experience with the coding and focusing on programming rather than troubleshooting the design of the crawler would have accommodated

a progress bar for the processing and drop-down selector for hotels as opposed to entering an integer in a non-discrete section of the script. Regarding the TripAdvisor's site time was spent inspecting which tag did what and this is now a known known. It is apparent that tags were coded to restrict the everyday person in scraping which is a credit to the wed designers.

The site itself was not quite like the sites offered as examples in the course yet the challenge to investigate the machinations of the contents proved to be a valuable exercise. Therefore, there were no limitations that weren't overcome with perseverance.

Regarding data extraction and copyright there is no specific exemption for data mining under the Copyright Act[6]. As the data is not reproduced for commercial reasons and is only a base for research outcomes it would be argued that under these circumstances shelter would be found under the fair dealing exemptions (fair use).

Data Wrangling (2042 words)

The data wrangling process starts the second phase of the programming done. Whilst the scraper is a standalone program it is to be used in conjunction with the Tripadvisor NLP and DL program.

Starting with 14 separate files generated by the scraper the first step was to consolidate the files into one repository – a panda dataframe. This process starts with a listing of all files within the scraped directory ('reviews') for the user to check whether all files are present.

```
# check reviews directory for listing of Hotel reviews
all_files = glob.glob(dir_namer+"/*.csv")

# display files
all_files
```

Table 4: Reviews directory display

The next step is to import the contents from each file into a dataframe using a loop that reads each line of the file and appends the contents to the hotels dataframe. To avoid double processing the dataframe is cleared at the commencement of the process as is the counter for files processed. Relating this back to fault finding if there is not the number of reviews that should be the sum of all file's reviews in the directory then all the user must do is repeat this step.

```
# clear arrays to receive input from review scrapings
df=[]
li=[]

for filename in all_files:

    df = pd.read_csv(filename,index_col=None,header=0)
    li.append(df)
```

```
li
```

Table 5: Reviews loader

```
# append all individual datafiles to one dataframe

df = pd.concat(li,axis=0,ignore_index=True)
df
```

Table 6: Create master dataframe

The product of these 2 steps is a consolidation of all reviews for all hotels with an identifier of the hotel if at any time further queries need to be done on the data.

As discussed in the previous section new columns needed to be formulated to accommodate related data which did not flow directly from the metadata into the appropriate columns. The need to copy three of the data columns (table 7) prior to this process was the first step then using regex superfluous data from the new fields (table 8).

```
# copy date review column to split out name of reviewer
# result will be a date reviewed and reviewer
df['date_review'] = df['reviewer']

# copy reviewer details to new column = contributions
df['contributions'] = df['reviewerdetails']

# copy Contributions to votes
df['votes'] = df['contributions']
```

Table 7: Create new columns

```
# return rating as a 1-5 rating
df['rating'] = df['rating'].div(10).round(0)

# clean out everything except for date in date reviewed
df['staydate'] = df['staydate'].str.replace('Date of stay:','', regex=True)
df['staydate'] = df['staydate'].str.replace('^.','', regex=True)

# clean out everything in date review except date
df['date_review'] = df['date_review'].str.replace(r'.*review.','', regex=True)

# clean reviewer of surfluous characters
df['reviewer'] = df['reviewer'].str.replace(r'.wrote.*','', regex=True)
df['reviewer'] = df['reviewer'].str.strip()
df['reviewer'] = df['reviewer'].fillna('unknown')

# retain ony vote numbers
df['votes'] = df['votes'].str.replace(r'.*contributions','', regex=True)
df['votes'] = df['votes'].str.replace(r'.*contribution','', regex=True) # single row
items without cirty details
df['votes'] = df['votes'].str.replace(r'(\D+)','', regex=True)
df['votes'] = df['votes'].str.strip()
```

```
# strip string portion of contributions to the right of contribution
df['contributions'] = df['contributions'].str.replace(r'contrib.*','', regex=True)

# copy above result to reviwer details (reviewer details has suffix of numerics, cont
ributions has legacy characters)
df['reviewerdetails'] = df['contributions']

# remove numercis from review details
df['reviewerdetails'] = df['reviewerdetails'].str.replace(r'(\d+)','', regex=True)
df['reviewerdetails'] = df['reviewerdetails'].str.strip()

# remove characters from contributions
df['contributions'] = df['contributions'].str.replace(r'(\D+)','', regex=True)
df['contributions'] = df['contributions'].str.strip()
df.drop('Unnamed: 0', axis = 1, inplace = True)
```

Table 8: Clean data using regex

Upon the completion of this stage a snapshot file is created titled wrangled data in case the model needs to be split. If this is the case, then prior to normalizing the data the wrangling data could be imported not needing to re-wrangle the data again.

Aside from this issue my local computer had issues with removing whitespace entirely from empty values within the dataframe so saving the wrangling data to file and re importing it overcame this issue.

The first step in normalisation is to remove all null values out of the data frame and replace with a suitable replacement (or in some cases omit the data from analysis at this point).

```
### remove null values from reviewerdetails, contributors and votes
df['reviewerdetails'] = df['reviewerdetails'].fillna("unknown")
df['date_review'] = df['date_review'].fillna(df['staydate'])
df['contributions'] = df['contributions'].fillna(0)
df['votes'] = df['votes'].fillna(0)
df
```

Table 9: Remove null values

Once the data set is free of null values the features and unique values can be examined for any irregularities. As we had already cleansed the data there should be no issues. A re-check for null values provides a listing of no missing values – table 10.

```
Missing values in dataset:   0
Out[13]:
rating             0
title              0
review             0
reviewer           0
reviewerdetails    0
staydate           0
hotel              0
```

```
date_review        0
contributions      0
votes              0
dtype: int64
```

Table 10: Null values

The next stage is to set up a column which will be used to extract the key words for word and sentence analysis later. The new column is a concatenate of title and review as there could be relevant information in both to the outcome of sentiment of the reviewer.

```
# Moving along the lines of bag of words output
df["reviews"] = df["title"]+", "+df["review"]
df['reviews'] = df['reviews'].str.replace('[^\w\s]','', regex=True)
df['reviews'] = df['reviews'].str.lower() # lower case for presentation and conformit
y purposes (visuals) prior to analysis
df
```

Table 11: Bag of words

The next phase is pre-processing which is where we become familiar with the data.

The first process is to determine the sentiment of the reviewer based on their reviews (and title of the review). To do so there needs to be a framework in place to assess sentiment and a score that could represent the level of sentiment. In this case the score will be between 0.5 (positive) and less than or equal to -0.5 (negative). The range in between will be classified as neutral (see table 12).

```
## Creating sentimental polarity (detemine whether reviews are positive, negative or
neutral)
analyzer = SentimentIntensityAnalyzer()

# Compound score
def compound_score(txt):
    return analyzer.polarity_scores(txt)["compound"]

## Sentiments
def sentiment(score):
    emotion = ""
    if score >= 0.5:
        emotion = "Positive"
    elif score <= -0.5:
        emotion = "Negative"
    else:
        emotion = "Neutral"
    return emotion
```

Table 12. Sentiment polarity and sentiment score

Once this is applied to the data set the results are added to two new columns: sentiment and sentiment score.

	rating	title	review	reviewer	reviewerdetails	staydate	hotel	date_review	contributions	votes	reviews	sentiment_score	sentiment
0	3.0	Looks Different	Check In was seamless and Matthew handled all ...	iggyAustralia	Sydney, Australia	August 2022	cb	11 Aug	127.0	48.0	looks different check in was seamless and matt...	0.9387	Positive
1	5.0	Best hotel stay ever!	This hotel was everything we expected and more...	Letch_K	Brisbane, Australia	May 2021	cb	June 2021	29.0	7.0	best hotel stay ever this hotel was everything...	0.9823	Positive
2	2.0	Disappointing Stay & Response to Situation	We booked a 2 night stay in a residence for 7 ...	Rob P	Colchester, United Kingdom	August 2022	cb	2 Aug	8.0	4.0	disappointing stay response to situation we b...	-0.7261	Negative
3	2.0	Mould in the aircon?	I stayed here for a business trip and the room...	Courtney J	unknown	June 2022	cb	July 2022	2.0	0.0	mould in the aircon i stayed here for a busine...	0.9118	Positive

Table 13: Sentiment and sentiment score added.

From the data we can now see the profile of sentiment of the reviews:

```
Positive    18129
Neutral      1537
Negative     1251
Name: sentiment, dtype: int64
```

The result is both pleasing in the sense that 86.7% of reviewer's reviews were positive but for the purpose of modelling the data is positively skewed. This will affect the outcomes from the deep learning model although this is a non-censored capture of data in the public domain so we will refrain from influencing the data through deleting or filtering results.

Progressing, we will now look at the data – visually.

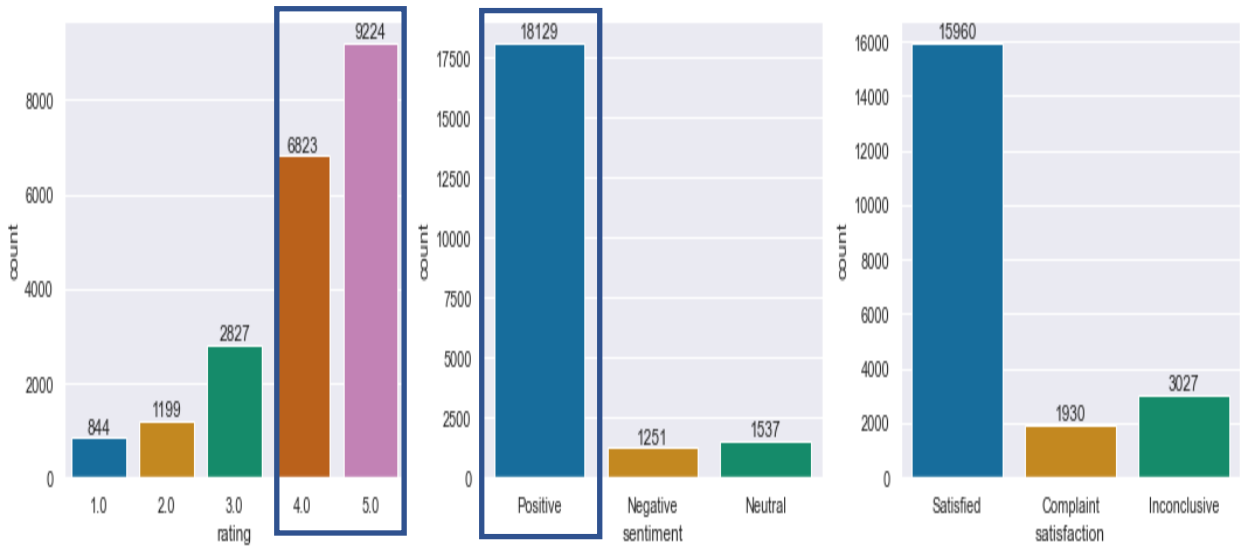
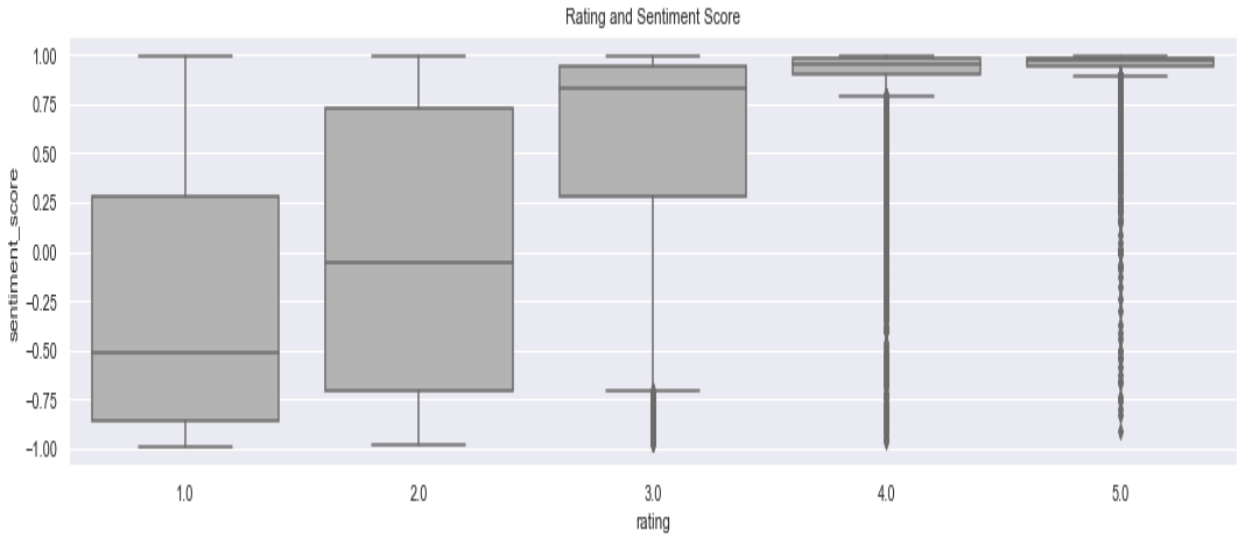


Figure 3: Sensitivity and sensitivity per rating

Figure 3 depicts that not only is sensitivity bias, but ratings are also. With 44% of the ratings being 5 bubbles and only 23.2% of the population being represented in ratings 1-3 there is a significant favorable skew towards high ratings. However, with 18,129 positive reviews and only 16,047 reviews being over 4 bubbles rated there appears to be a mixture between sentiment and rating.

Diving deeper into the data using box plot charts we can see that there are long tails on both 4 and 5 bubble rating reviews while 1 and 2 bubble reviews are more spread over the sentiment scale. In fact, the lower fence for rating 4 is 0.79 and 0.89 for rating 5 very close to the max scores of 0.99 for both.



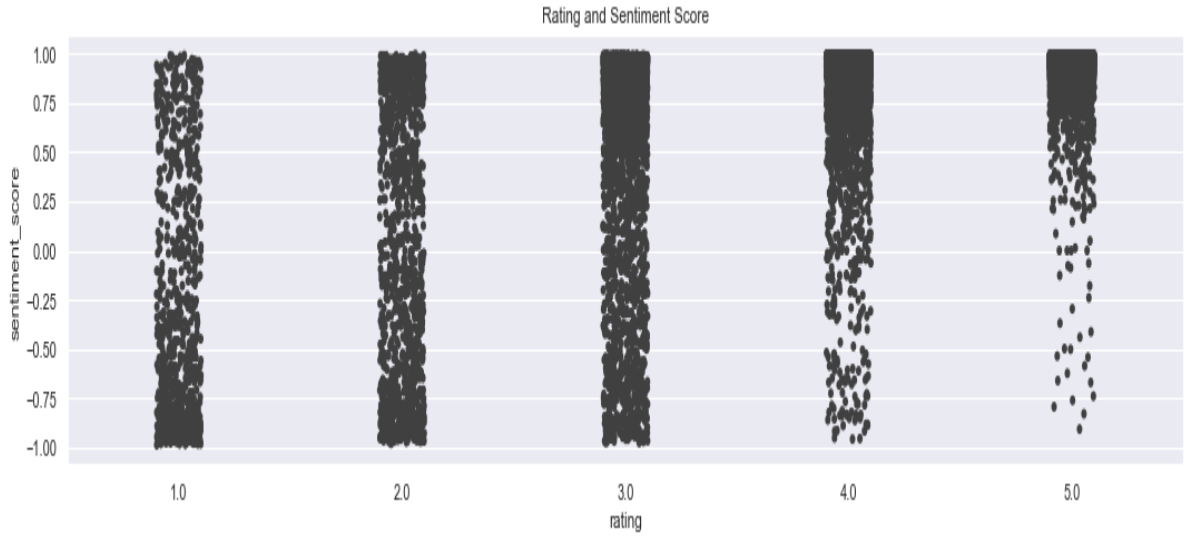


Figure 4: Box plot of rating and sentiment score

Redefining the parameters of sentiment and projecting forward with a new category satisfaction with 3 variables: complaints, satisfied and inconclusive. Satisfaction is an adhoc measure formulated while reviewing seaborne boxplot graphs which do not print nor generate graphic files, yet the graphs can be viewed within the model.

The basis of satisfaction is:

- complaints = sentiment_score<=0 # 1930 reviews
- satisfied = sentiment_score>=0.85 # 15960 reviews
- inconclusive = sentiment_score<0.85 & sentiment_score>0 #3027 reviews

Complaints were evenly spread between ratings 1 – 3 yet a small proportion appeared in 4 and 5. Conversely, satisfied ratings appeared in rating groups 1-3. The below table is a span shot of the results

Rating	Complaints	inconclusive	Satisfied	Total
1	573	219	52	844
2	625	373	201	1199
3	561	932	1334	2827
4	144	976	5703	6823
5	27	527	8670	9224
Total	**1930**	**3027**	**15960**	**20917**

Table 14: Satisfaction verses sentiment score

Expanding the analysis to include sentiment as measured by words:

	Ratings					
	1	***2***	***3***	***4***	***5***	Total
Complaints	**573**	**625**	**561**	**144**	**27**	**1930**
Negative	427	418	323	71	12	1251
Neutral	146	207	238	73	15	679

Inconclusive	**219**	**373**	**932**	**976**	**527**	**3027**
Neutral	114	169	319	194	62	858
Positive	105	204	613	782	465	2169
Satisfied	**52**	**201**	**1334**	**5703**	**8670**	**15960**
Positive	52	201	1334	5703	8670	15960
Total	**844**	**1199**	**2827**	**6823**	**9224**	20917

Table 15: Satisfaction verses sentiment

Table 15 highlights that the bubble ratings associated with the review maybe flawed. The fact that 52 reviews could be classified as positive yet be given a 1 bubble rating fails logic as does 12 complaints originating from 5 bubble rated reviews. Yet reviewing the actual reviews in the later class and one can does not link the satisfaction score to the words of the review. The following review is and example: *capturing cairns on the cairns foreshore a myriad of activities await you at pullmans casino hotel adults can use the casino children it adults savor the australiana zoo while experiencing a first class hotel (*which scored -0.6708).

Alternatively a score of 0.9661 was given to this review: *rude management i regularly stay at shangri la hotels in asia kl shangri la malaysia penang shangri la malaysia and riverfront shangri la bangkok thailand this was my first experience of shangri la in australiaman what a contrast shangri la hotels are renowned for their friendly hospitality and excellent service in asia they regularly win awards over hotels like the mandarin oriental there i found the amenities in this hotel to be fine for this standard of hotel i found the staff to be inexperienced and the management to be overbearing and rather rude experienced 5 star hotel staff in my opinion are friendly courtious and helpful inexperienced staff tend to think they are superior to the guests and have yet to learn the above mentioned qualities.*

As a conclusion from sentiment analysis there is no accounting for sarcasm, sentences that have alternate points structured within (example 2: is a complaint but highlights the chains strength in other locations) and in the case of the first example questionable scoring.

Taking this into account and moving forward we focus on the words alone and their influences to mapping to a rating. Here we are looking for the use of words and their repetition within categories. Figure 5 highlights those words that are frequently used in the categories of sentiment. The larger the word the higher the frequency. Common across all categories is room, hotel, stay all of which do not highlight specific features which would lead to classification. The words great, excellent, good are descriptors which help categorize and affirm the reviews classification. However, from a management point of view Staff, location, service, bed, breakfast are features which are controllable and as such this analysis could support the strength, weakness portion of a SWOT analysis for the hotels.

Figure 5: Word frequency in review sentiment

Figure 6 lists those words that frequently occur in each of the rating categories. Again room, hotel, stay feature prominently yet more words are filtering through such as:

- 5 bubble rating: - helpful, friendly, pool, service, restaurant, clean
- 4 bubble rating: - nice, restaurant, great
- 3 bubble rating: - service, location, need, bed, bathroom, breakfast
- 2 bubble rating: - location, good, bed, old, need
- 1 bubble rating: - bathroom, booking, dirty, told, even, need, old, good, check

Figure 6: Word frequency in review ratings

It appears word frequency is the source of defining categories – yet not conclusively. A further interjection at this point into the analysis manually would clarify some of the issues of words populating categories that they were wrongly allocated to.

Following the frequency analysis, we load the pre-processed data to perform key word analysis. This starts with removing punctuation from the reviews the define the key words using the following script:

```
# to extract key words from reviews to a list

reviewsDetails['key_words'] = ''
r = Rake()   # use Rake to discard stop words (based on english stopwords from NLTK)

for index, row in reviewsDetails.iterrows():
    r.extract_keywords_from_text(row['reviews'])   # to extract key words from review
s
```

```
    key_words_dict_scores = r.get_word_degrees()    # obtains dictionary with key wor
ds and their scores
    row['key_words'] = list(key_words_dict_scores.keys())   # to assign list of key w
ords to new column

reviewsDetails
```

Table 16: Defining keywords

The result is a list embedded within the data set which has parsed the sentence making up the review into the component words. Consequently, we can know view the top twenty words used throughout the corpus (table 17) and view the top 20 words graphically (figure 7).

		keyword	count			keyword	count
1	18	room	12301	6	50	great	7083
2	7	hotel	11414	7	8	cairns	6809
3	91	staff	10800	8	208	rooms	6076
4	249	location	9032	9	75	stayed	5828
5	70	stay	7857	10	369	good	5642
		keyword	count			keyword	count
11	12	friendly	5353	16	904	view	4792
12	62	clean	5236	17	63	breakfast	4553
13	139	would	5218	18	96	helpful	4386
14	256	service	5215	19	250	well	3934
15	232	pool	4895	20	713	nice	3927

Table 17: Top 20 words in corpus

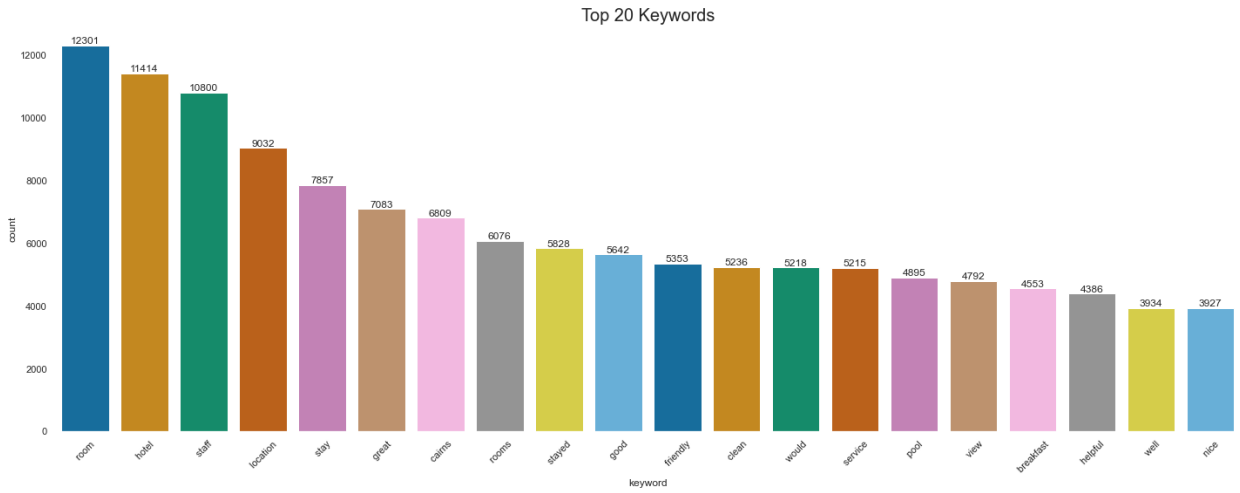


Figure 7: Top 20 words

The word analysis has supported the analysis previously discussed with the words room, hotel, staff, location all featuring prominently in the sentiment analysis and are the top 5 words listed above.

The next step is to remove stop words from the reviews. Stop words are those words that are commonly used but hard to definitively define. The nltk package has a dictionary of predefined list of words which it extracts. This list is available at https://gist.github.com/sebleier/554280 .

There is an argument that stop words should be included in text analysis as it they are more than just words. As John Mueller Trends Analyst of Google states ""*To be or not to be*" is a collection of stop words but stop words alone don't do it any justice"[6].

Once stop words have been removed the process of Lemmatization takes place. Lemmatizing converts a word to its base form.

```
# Define the sentence to be lemmatized
sentence = "The striped bats are hanging on their feet for best"

# Tokenize: Split the sentence into words
word_list = nltk.word_tokenize(sentence)
print(word_list)
#> ['The', 'striped', 'bats', 'are', 'hanging', 'on', 'their', 'feet', 'for', 'best']

# Lemmatize list of words and join
lemmatized_output = ' '.join([lemmatizer.lemmatize(w) for w in word_list])
print(lemmatized_output)
#> The striped bat are hanging on their foot for best
Including
Detailing:
a. Cleaning, normalisation, feature extraction of the sourced data. Normalisation may
include applying a word embedding algorithm
b. Summary and visualisation of the harvested data. Preliminary EDA is
acceptable in this section as well.
```

Table 18: Lemmatize and tokenize example[7]

The next step is to join the lemmatized words into a string separated by commas to be prepared for tokenization which is the conversion of the lemmatized words into a vector that has a coefficient for each token in the from of binary values. This then creates by default a vocabulary index based on word frequency. Every word gets a unique number with zero being reserved for padding. In the case of this corpus there were 26,078 words with maximum word count per sentence being 101 words.

The following histogram illustrates the profile of the corpus:

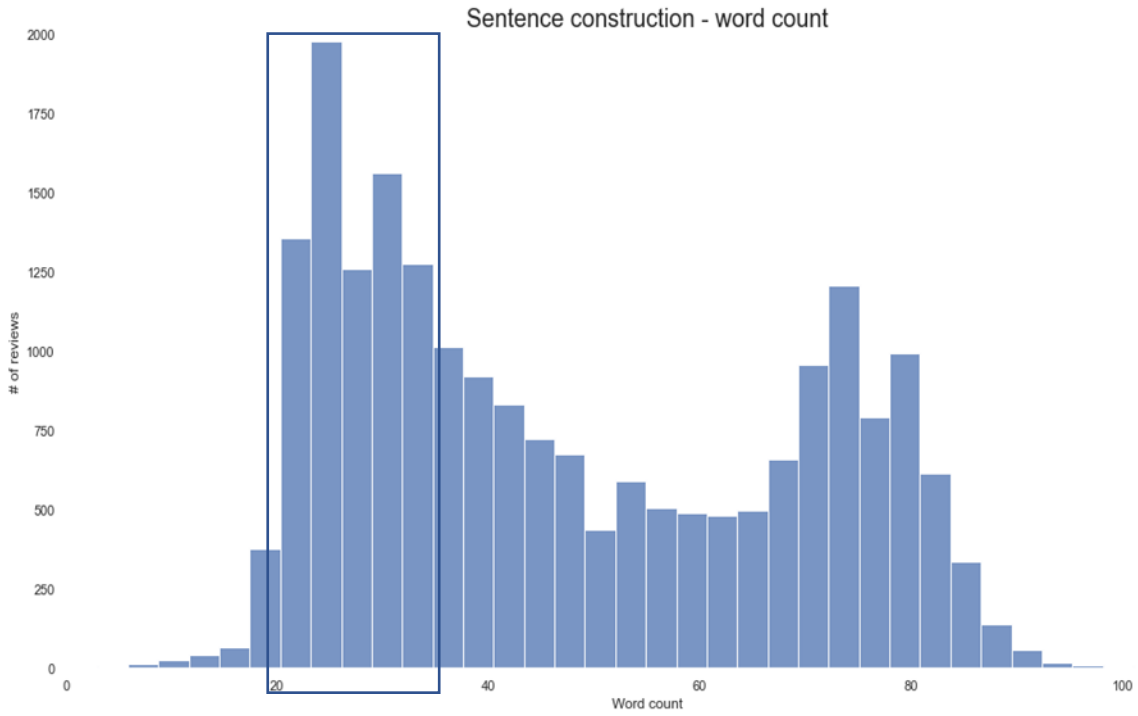


Figure 8 Sentence construction - word count

The main volume of the corpus is between 25 and 30 words which is very different to other Tripadvisor review analysis which were in heavily skewed in the range between 20 words and 100 words with some review extending to over 500 words[4].

There is a train of thought that the longer the review the more likely the review will be negative in nature, yet the longest review was deemed to be neutral in nature yet a rating of 2. A review of the matrix generated after the tokenization leads one to conclude the train of thought is not far from the truth. Certainly, a good place to start if poor reviews were of principle interest.

The final steps to data wrangling involve refinement of what will be the two key inputs in the deep learning program – the X and Y classifiers.

The tokenized matrix is represented by X. This matrix requires all its values to be populated which post tokenization is not the case. By determining the longest sentence is 101 words and there are 20,917 reviews the matrix profile should be 20917; 101 with each row having words represented in integer form yet if the sentence ends prior to the 101st column there will be blank values. These values need to be populated (padded) which in simple terms means filling the blanks with 0. Now each row has 1011 columns of integers (table 18).

```
# fill blank elements of array with 0 (pad) to complete indicies matrix

# Padding the reviews [Pads sequences to the same length.]
X = pad_sequences(X, padding='post', maxlen=101)
```

Table 19: Padding tokenized matrix

The predictor for the model is the rating for the review and this requires the rating column to be converted firstly from a string to a float and then due to the packages supplied to model the index will start at zero. That requires that each of the ratings need to be converted for [“1”,”2”,”3”,”4”,”5”] to [0,1,2,3,4,5]. This is done through the utilization of a map (table 20).

```
# used y2 due to spending hours watching cryptic error messages return from ML models
output.
# the only way I was assured of converting 1,2,3,4,5 from a string to a float and the
n convert the list to stay in the parameters of 0-4 for the model

y = reviewsDetails['rating']
y2 = np.frompyfunc(lambda x: x.replace(',',''),1,1)(y).astype(float)
y2

y2 = y2.map({1: 0,
             2: 1,
             3: 2,
             4: 3,
             5: 4
            })
y2
```

Table 20: Mapping Y axis with float values

Deep Learning (1519 words)

The process of deep learning starts with the construction of a model that will test the data which has previously been wrangled.

The model chosen is based on the Keras Sequential model as it fits the criteria of this study.

Keras[8] states that a sequential model is ***not*** appropriate when:

- Your model has multiple inputs or multiple outputs
- Any of your layers has multiple inputs or multiple outputs
- You need to do layer sharing
- You want non-linear topology (e.g., a residual connection, a multi-branch model)

In this case we can say no to each of the above statements and conclude on a double negative that the sequential model is appropriate in assessing a model with single inputs and outputs.

The principle behind a sequential model is that it consists of a series of layers where each layer has one input tensor and one output tensor .

The model starts with an input layer using a long-short term memory (LSTM) modelling to drive the inner layers to process the input tensors and finally produce the tensor output (in our case a rating).

In the case of the keras model being constructed the layers are represented by the term dense and what happens at the layer level at each node is governed by the term activation. In this

case the activation is driven by the coding of 'relu' for inner layers and 'softmax' for the outer layer where:

- Relu is the formula (f(x) = max(0,x)); and
- Softmax converts a vector of K real numbers into a probability distribution of K possible outcomes. Softmax is used where there is multi-classification problems where class membership is required on more than two class labels.[10]

A problem faced with modeling data using neural networks is overfitting (figure 9a). Overfitting is where a neural network (ANN) performs well on the training data yet fails when faced with a new data set. To overcome this issue Keras allows the addition of dropout layers which regularizes[11] the ANN preventing overfitting to occur. In this case a dropout layer is included between each layer of the model thus proving end points for some of the tensors (figure 9b).

The dropout rate ranges between 0-1 where 0 is no dropout rate and Brownlee (2018)[12] suggests that most modeling is done with rates between 0.5 and 0.8 but only where the NN is complex, and overfitting is more present. In this case the NN is simple and 0.2 has been settled on after numerous tests. Table 21 shows the coding for the sequential model.

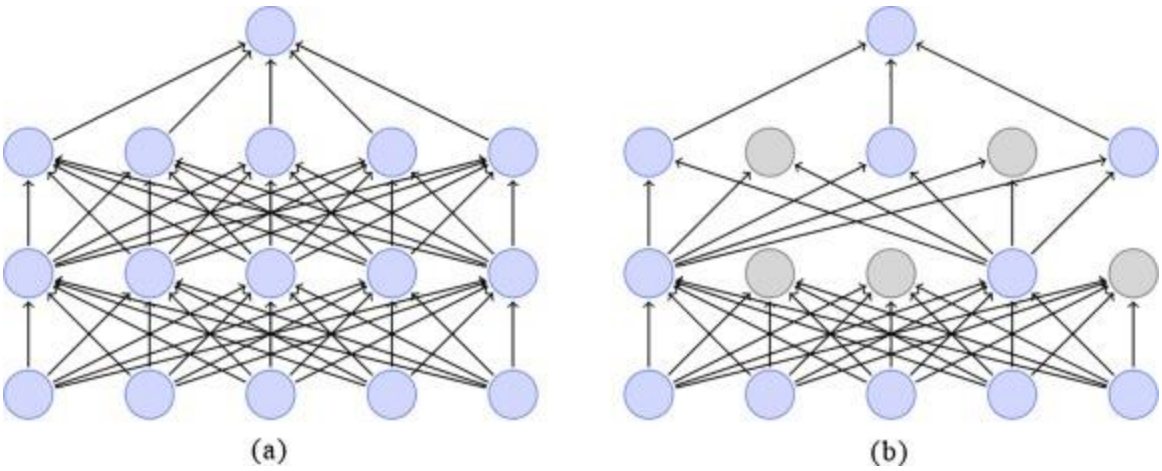


Figure 9: (a) ANN without dropout (b) ANN with dropout (source Science Direct[12])

```
# Setup layers and input output tensors, weights (dense),

#https://keras.io/guides/sequential_model/
model = tf.keras.Sequential([
    L.Embedding(vocab_size, int(embedding_dim), input_length=X.shape[1]),
    L.Bidirectional(L.LSTM(int(units),return_sequences=True)),
    L.Conv1D(64,3), # the values here were chosen over numerous test
    L.MaxPool1D(),
    L.Flatten(),
    L.Dropout(0.2),
    L.Dense(128, activation="relu"),
    L.Dropout(0.2),
```

```
    L.Dense(64, activation="relu"),
    L.Dropout(0.2),
    L.Dense(5, activation="softmax") # predicted rating should be a value between 0 a
nd 5
])
```

Table 21: Neural Network Model code

Once the model has been created the next step is to compile the model. The model compiler defines the loss function, optimizer and metrics. The loss function finds the error or deviation in the learning process. Since we are training data for categories and there is more than 2 classes, we used Sparse Categorical Crossentropy as recommended by TensorFlow[13].

```
# Using Sparse Categorical Crossentropy, Accuracy metric and ADAM optimizer for train
ing.

# https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossen
tropy (sparse_categorical_crossentropy is advisable when there are 2 or more lables (
5 in this case))

model.compile(loss=loss_function, optimizer=optimizer,metrics=['accuracy'])
```

Table 22: ANN Model Complier

The optimizer function adam was chosen as it requires little memory to operate and is well suited for problems that have large data/parameters while the metrics is accuracy as we are looking for the outputs that will determine whether the model is effective or not.

With the model's parameters set in place the models variables or hyperparameters. These hyperparameters are the values that control the learning process. With numerous hours of research, I have not found a document that states an algorithm to use to populate these variables (with the exemption of epochs). In practice I have had to run numerous tests across the data to find a set of values that when populated into the model being used the optimum result is presented. In this case the following values were effectively the providers of the best accuracy from the model on my local computers:

- embedding_dim = 20
- units = 64
- val_split = 0.12
- batch_size = 150
- epochs = 2

The embedding_dim is used in shaping the parameters of the model. Testing of this value ranged between 10 and 25 with 20 providing a consistent accuracy rate (over 63%).

The unit parameter is a trade off between depth and breath. The higher the number the more features are included while a lower number will provide richer features. Since we are dealing with words and not images the setting that appeared to work constantly was 64.

The val_split variable relates to the validation dataset within the training section of the model. This figure basically states that 12% of the data from the training set will be used to validate the test data for accuracy.

Test data 10%	Training data (78%)	Validation data (12%)

In summary the configuration for the model is:

```
# Model configuration
batch_size = 150
embedding_dim = 25
epochs = 2
loss_function = SparseCategoricalCrossentropy()
no_classes = 100
optimizer = 'Adam'
units = 64
verbosity = 1
val_split = 0.1
```

Table 23: Model configuration

```
history = model.fit(X_train, y2_train, epochs=int(epochs), validation_split=float(val
_split), batch_size=int(batch_size), verbose=verbosity)
```

Table 24: Training for 2 epochs

As previously discussed, the epochs can be chosen by a simple run with the epochs number chosen outside the normal range for the model (in this case 10). Plotting the history of accuracy over each of the epochs will present the following graph:

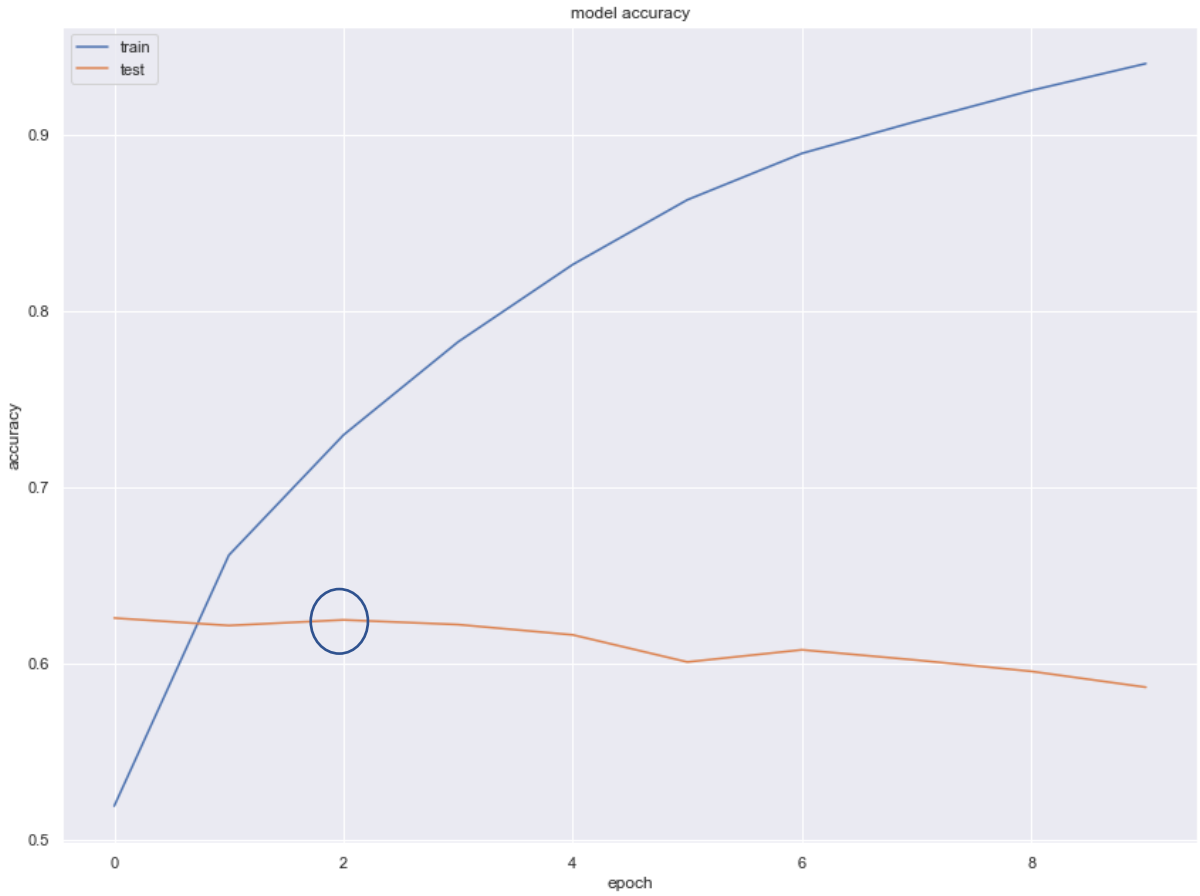


Figure 10: History of accuracy per epoch

Figure 10 shows that the test data starts to fall in accuracy after epoch 2 (circled) which is why this number was chosen as the optimum value for the model's configuration.

Table 25 lists the results of 6 test done within the past hour. The setting for the train/test data split is listed in table 24.

```
X_train, X_test, y2_train, y2_test = train_test_split( X, y2, test_size=0.2, random_s
tate=2)
```

Table 24: Training -test data set split

Variable	1	2	3	4	5	6
Embedding_dim	20	20	20	20	20	20
units	50	50	50	60	50	64
Epochs	2	2	2	2	2	2
Val_split	12%	12%	12%	12%	20%	12%
Batch_size	150	150	300	150	150	150
Training_set	80%	90%	80%	80%	80%	80%
Accuracy	64.01%	63.72%	62.14%	62.95%	62.14	64.1%

Table 25 Test results of modelling

The above table shows that test 9 was the most successful outcome for the model with and accuracy of 65.06%. These tests were completed at and all run on a single machine without changing the random variable of the split training/test data set. Results may differ on other machines, yet the range of results are within the 60.0% – 64.1% range.

Running the NN model produces the following outputs:

- Model accuracy (test vrs train);
- Model loss (test vrs train);
- Prediction – which predicts the review rating from the train set;
- Accuracy score;
- Root Mean Square Error (RMSE) of the model;
- Confusion Matrix Heatmap;
- Classification report.

The model accuracy and loss graphs depict the accuracy of both the train and test data resulting from the models

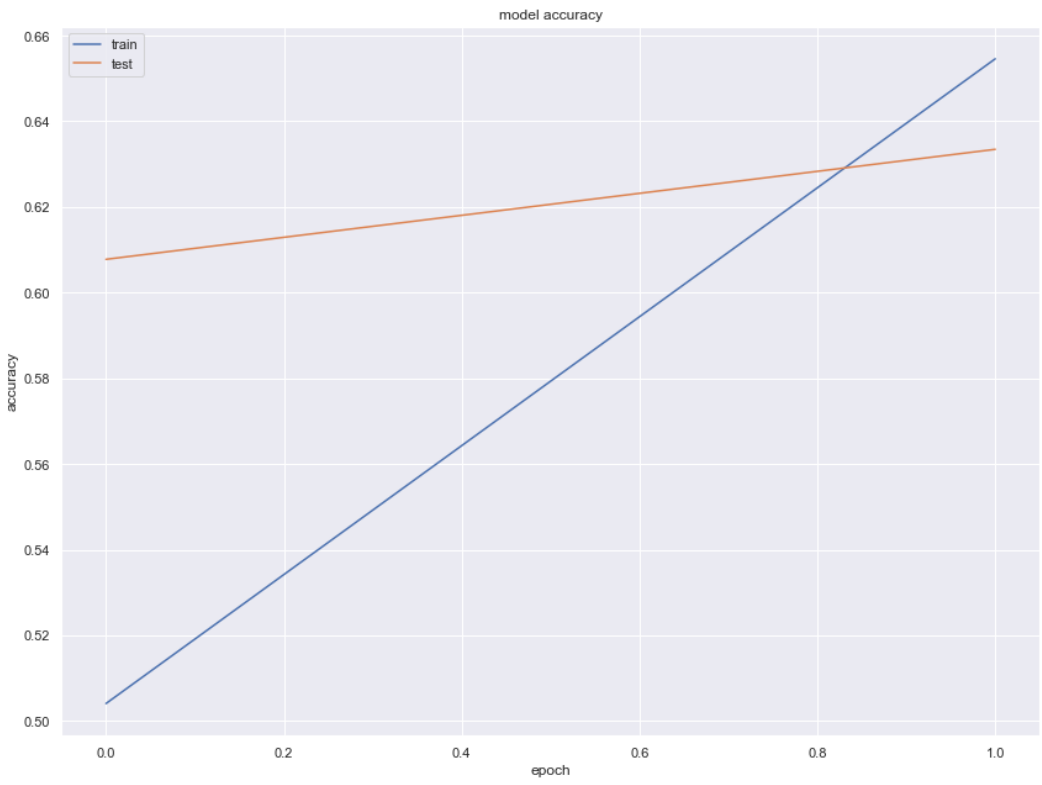


Figure11 Model accuracy

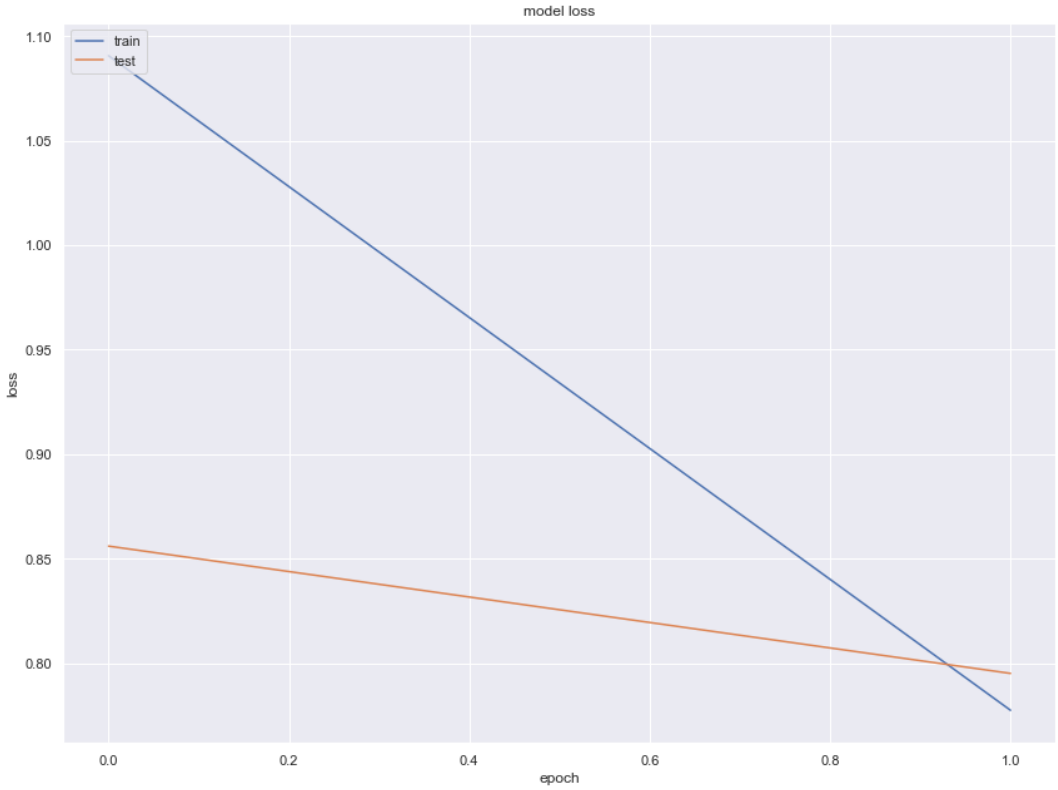


Figure12 Model loss

Figure 11 shows that the accuracy steadily improves from a beginning of 0.6078 at the first epoch to 0.6335 at the second epoch while val_loss dropped from 0.856 to 0.7951 over the same time during the models training process.

Post prediction and the score of 64.1% for accuracy (TP+TN)/ (TP+FN+TN+FP) with a RMSE of 0.72 is recorded. As an indicator of whether the model is a good predictor or not a RMSE score closer to zero is a better fitting model than those that are near to 1. Consequently, this model is not as good at predicting as it should be.

Investigating this point, we can view the predictive data in a heatmap to isolate the strengths and weakness of the model (figure 13).

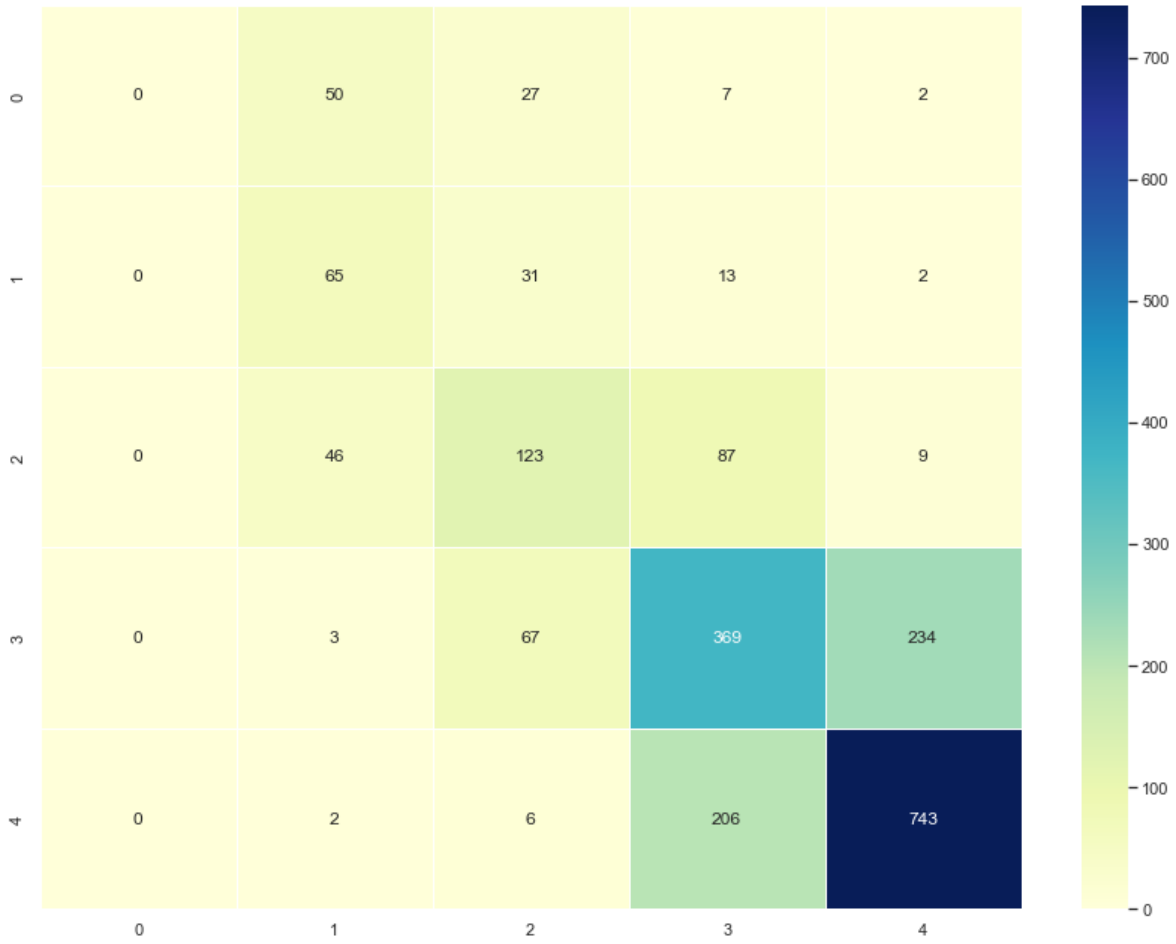


Figure 13: Confusion Matrix Heatmap of prediction results

The above confusion matrix heatmap clearly identifies the model correctly classified:

- 743 5 bubble ratings; 369 4 bubble ratings; 123 3 bubble ratings; 65 2 bubble ratings and failed to predict a 1 bubble rated review.

Clearly the mix of data has influenced the predictions. As the dataset is strongly skewed towards the bottom 4 squares (74.1% of the data predicted) even small changes to this area would increase/decrease the accuracy considerably. As an example, perhaps better defining the words may change some of those predictions that are 4 but should be 5 and move those predicted as 5 to 4.

Finally, the classification report lists the precision (positive predictions measured against the positive class – TP/ (TP+FP)), recall (correct predictions made out of all the positive examples in the dataset- TP(TP+FN)) and the F measure which is the harmonic mean of both precision and recall (2 *(precision * recall)/(precision + recall). Additionally, the classification report lists accuracy which is slightly different to the accuracy calculated previously. The accuracy on the classification is the product of (TP)/(TP+FN+TN+FP). This where TP = true positive, FP = false positive, FN = false negative, TN = true negative.

Classification report :

	precision	recall	f1-score	support
0	0.00	0.00	0.00	86
1	0.39	0.59	0.47	111
2	0.48	0.46	0.47	265
3	0.54	0.55	0.54	673
4	0.75	0.78	0.76	957
accuracy			0.62	2092
macro avg	0.43	0.47	0.45	2092
weighted avg	0.60	0.62	0.61	2092

Figure 14: Classification report

In conclusion the model formulated achieved a performance accuracy of between 60 and 65% depending on the hyperparameters selected with a RMSE value of 0.72 which is not fantastic but reasonable at predicting review ratings from the text of the reviews supplied. Contributing factors of the less than impressive performance is the skewness of the data sourced and in turn the nature of the process of determining sentiment from the written word using mathematical algorithms.

Issues in collating the data were few yet copyright infringement is a consideration, yet the project was done considering fair use and educational purposes. Until the legislation has been clearly defined this area will remain a grey area with a conservative stance of requesting data which is already in the public domain with no firewalls in place to restrict access.

Other issues surrounding the data is the imbedded sarcasm in some reviews while others used a positive background when delivering a negative review.

Issues regarding the modelling of the data focused on overfitting of the model yet the results clearly state that this has not happened.

NLP and Deep Learning is a complex and interesting topic with considerable impact in daily life in future years. It is, and always will be, an area of requiring continuous improvement processes part of which will be the development of the model created here.

Bibliography

1. Chiny M; Bencharef, O, et al (2021), A Client-Centric Evaluation System to Evaluate Guest's Satisfaction on Airbnb Using Machine Learning and NLP. https://www.hindawi.com/journals/acisc/2021/6675790/
2. Suzuki, T; Gemba K; Aoyama A, (2013), Hotel classification visualization using natural language processing of user reviews. https://ieeexplore.ieee.org/document/6962540
3. Chang Yung-Chun, Ku Chih-Hao, Chen Chien-Hung (2020), Using deep learning and visual analytics to explore hotel reviews and responses. https://www.sciencedirect.com/science/article/pii/S0261517720300558#sec1
4. Awan, A, Trip Advisor Data Analysis/ML https://deepnote.com/@abid/Trip-Advisor-Data-AnalysisML-f6060b39-d76c-4579-9648-a54bc8b5ffb5
5. Australian National University, Text Data Mining – Copyright https://libguides.anu.edu.au/c.php?g=792730&p=5829437
6. Wikipedia, Stop word https://en.wikipedia.org/wiki/Stop_word
7. Prabhakaran S, (2018) Lemmatization Approaches with Examples in Python https://www.machinelearningplus.com/nlp/lemmatization-examples-python/
8. Keras (2020), The Sequential Model https://keras.io/guides/sequential_model/#when-to-use-a-sequential-model
9. CS231N Convolutional Neural Networks for Visual Recognition https://cs231n.github.io/neural-networks-1/
10. Brownlee, J (2020) Softmax Activation Finction with Python https://machinelearningmastery.com/softmax-activation-function-with-python/#:~:text=This%20can%20be%20achieved%20by%20calculating%20the%20exponent%20of%20each,sum%20of%20the%20exponent%20values.&text=For%20example%2C%20we%20can%20turn,3)%20%2B%20exp(2))
11. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors Comput. Sci., 3 (2012), pp. 212-223
12. J. Brown (2019), A Gentle Introduction to Dropout for Regularizing Deep Neural Networks https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
13. TensorFlow tf.keras.losses.SparseCategoricalCrossentropy

https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

Tables

Figures

12. Model loss
13. Confusion Matrix Heatmap of prediction results
14. Classification report.

List of Support Files

1. Tripadvisorscraper.ipynb – Tripadvisor web scraper file
2. TripadvisorNLDP.ipynp – Tripadvisor wrangling, NLP and Deep Learning processor
3. CairnsAccom.csv – list of Cairns Hotels details need for Tripadvisor scraper