CZ4045 Natural Language Processing - Assignment 1 (G06)

Asuri Simhakutty Kamakshi
KAMAKSHI001@e.ntu.edu.sg
Nanyang Technological University
Singapore

Unnikrishnan Malavika
MALAVIKA002@e.ntu.edu.sg
Nanyang Technological University
Singapore

Musthiri Sajna
SAJNA001@e.ntu.edu.sg
Nanyang Technological University
Singapore

Das Atrik
ATRIK001@e.ntu.edu.sg
Nanyang Technological University
Singapore

Padhi Abhinandan
ABHINAND001@e.ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

The purpose of this report is to present the inferences obtained from the various Natural Language Processing (NLP) tasks performed on a collection of user reviews posted on Yelp. This project follows through the steps of analysis, NLP tasks and evaluates the obtained results following which a simple application is created using NLP concepts on the same dataset.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Information extraction**; *Lexical semantics*; **Phonology / morphology**.

KEYWORDS

tokenization, stemming, POS tagging, sentiment analysis

ACM Reference Format:
Asuri Simhakutty Kamakshi, Unnikrishnan Malavika, Musthiri Sajna, Das Atrik, and Padhi Abhinandan. 2021. CZ4045 Natural Language Processing - Assignment 1 (G06). In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This project aims to perform the various steps involved in an end-to-end Natural Language Processing (NLP) application. The main components implemented are

(1) Dataset analysis
(2) Extraction of Indicative Adjective Phrases
(3) Simple sentiment analysis application to analyse user sentiments from the Yelp reviews.

The NLP task were implemented using functions from NLTK and SpaCy libraries.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Dataset Description

The dataset used in this project is a collection of user reviews posted on Yelp [1]. It contains 15,300 reviews. Each data case has the following information as attributes - review ID, user ID, business ID, stars, date, text, useful, funny and cool. The columns business ID, text and stars to carry out the given nlp tasks.

2 DATASET ANALYSIS

We use the Pandas dataframe to carry out dataset analysis given the extensive built-in functions that the dataframe allowing easy manipulation of data.

2.1 Tokenization and Stemming

Tokenization is the process of breaking down a sentence, phrase, or text document into smaller parts called tokens, such as words or keywords. In this project, we utilize tokenization to extract all of the reviews from two random businesses and compare the data sets' word frequency distributions and top ten frequent words before and after stemming.

A small sample of tokens that are generated after performing tokenisation on the selected dataset is as indicated in : The tokenizer used on this data set is NLTK's word_tokenize. But before performing tokenization, we must remove stop words from the sentences in the dataset. Removing stop words from the tokens before the frequency distribution is important for avoiding clouding of frequency graphs. In order to remove stop words, a function called rem_sw() is used.

```
'Relative', 'to', 'the', 'quality',
 'of', 'the', 'ingredients', 'that',
 'Ed', "'s", 'uses', '(',
```

Figure 1: Some tokens generated

```
nltk.download('stopwords')
def rem_sw(path):
    nltk_tokens_without_sw = [word for word in
        path if not word in stopwords.words()]
    return nltk_tokens_without_sw
```

It is observed that the samples of the frequency distributions, as shown in figures, contain symbols such as '!', '$', '.' as tokens. However, it is incorrect for such special characters to be considered as valid tokens. This is one limitation of the tokenization method that has been used. In addition to tokenization, stemming is used to get the root of the word which then helps to identify the most frequent tokens easily.

Figures 2-5 show the frequency distribution of tokens across two businesses before and after stemming. Table 1 shows a compilation of the most frequent tokens for each Business, before and after stemming.

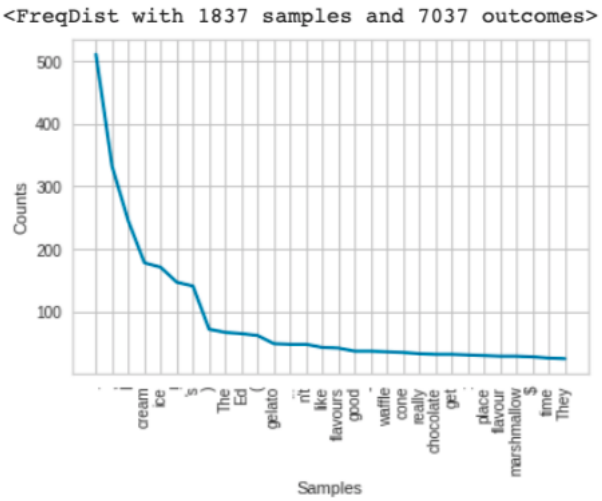

Figure 2: Frequency distribution for Business 1 before stemming

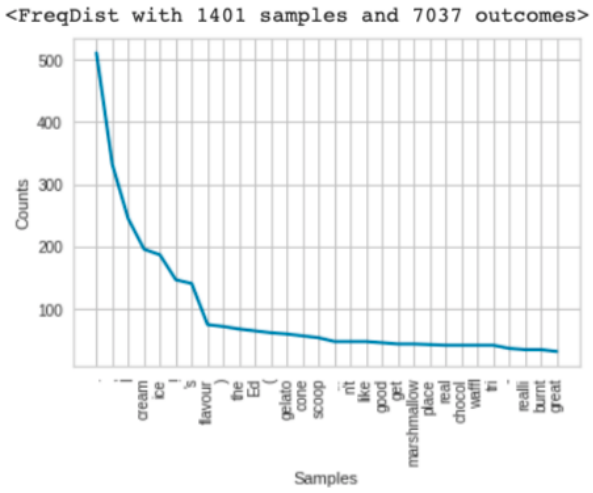

Figure 3: Frequency distribution for Business 1 after stemming

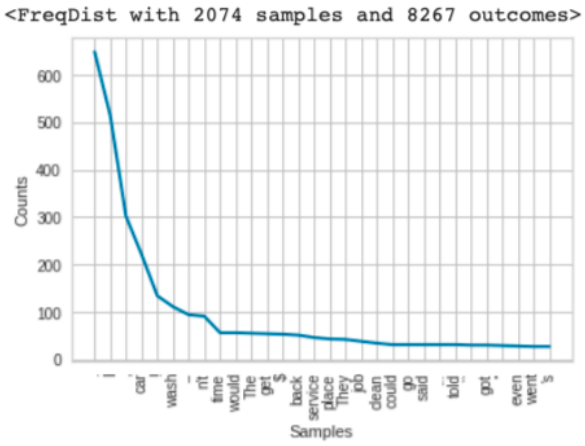

Figure 4: Frequency distribution for Business 2 before stemming

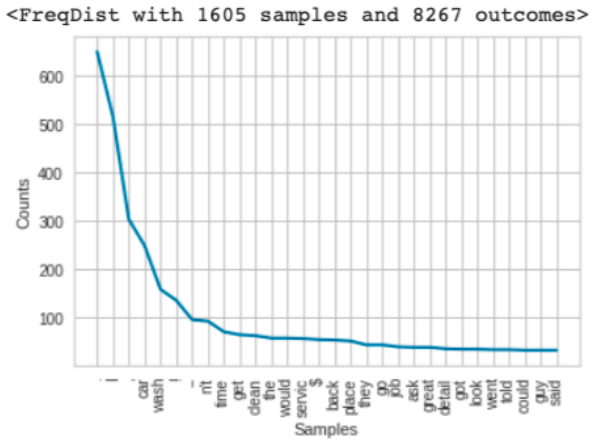

Figure 5: Frequency distribution for Business 2 after stemming

2.2 POS Tagging

Part-of-speech tagging is the process of categorizing words in a text according to the part of speech, it's definition and context. On our datasets we applied pos tagging methods from both NLTK and SpaCy libraries. (see POS_tagging_nltk and POS_tagging_spacy functions)

```
def POS_tagging_nltk(path):
  global count
  print("Sentence " + str(count) + ":")
  sentence = word_tokenize(path)
  print("Tokenization: " + str(sentence))
  tagged_sentence = nltk.pos_tag(sentence)
  print("POS Tagging using NLTK: " +
  str(tagged_sentence))
```

	Business 1	Business 2
Before Stemming	[('.', 511), (',', 332), ('I', 245), ('cream', 178), ('ice',171), ('!', 147), ("'s", 141), (')', 72), ('The', 67), ('Ed', 65)]	[('.', 649), ('I', 514), (',', 302), ('car', 221), ('!', 134), ('wash', 111), ('–', 94), ("n't", 91), ('time', 56), ('would', 56)]
After Stemming	[('.', 511), (',', 332), ('I', 245), ('cream', 196), ('ice', 187), ('!', 147), ("'s", 141), ('flavour', 75), (')', 72), ('the', 68)]	[('.', 649), ('I', 514), (',', 302), ('car', 247), ('wash', 157), ('!', 134), ('–', 94), ("n't", 91), ('time', 69), ('get', 63)]

Table 1: 10 Most Frequent Tokens Distribution

NLP Task	Better Performance
Word Tokenization	SpaCy
Sentence Tokenisation	NLTK
POS Tagging	SpaCy

Table 2: Performance Analysis

```
    print()
    count += 1

def POS_tagging_spacy(path):
    global count
    print("Sentence " + str(count) + ":")
    print()
    sentence = nlp(path)
    print(f"{'Text':{10}} {'POS':{8}}
      {'TAG':{8}} {'Dep':{8}} {'POS explained':
          {25}}{'Tag explained'} ")
    for token in sentence:
      print(f'{token.text:{10}} {token.pos_:{8}}
      {token.tag_:{8}} {token.dep_:{8}}
      {spacy.explain(token.pos_):{25}}
      {spacy.explain(token.tag_)}')
    print()
    count += 1
```

2 sets of tagging results performed on 5 sentences are obtained and analysed. NLTK Library We use POS tags defined in the Penn Treebank project and the POS tags for each word in the 5 sentences is printed out in the Jupyter notebook. SpaCy We use the predefined POS tags in the SpaCy library and similar to the NLTK one the POS tags for each word is printed out.

The main difference between SpaCy and NLTK is that NLTK is a string processing library. It takes strings as input and returns strings or lists of strings as output. Whereas, spaCy uses an object-oriented approach. When we parse a text, spaCy returns a document object whose words and sentences are objects themselves. spaCy has support for word vectors whereas NLTK doesn't. NLTK splits text -> sentences -> words whereas spaCy constructs a syntactic tree for each sentence. In word tokenization and POS tagging spaCy shows better performance compared to NLTK but NLTK is better for sentence tokenization.

2.3 Writing style

To analyse the writing styles of different articles, the following functions are performed -

Calculate the number of words and sentences - using sent_tokenize and word_tokenize The length of an article is an important indicator because content published on the internet is subject to search engine optimization (SEO) among many other algorithms that run through millions of articles to provide the best search results. According to Copypress[1], the average word count of top ranked searches is 2416 words, and the varying length of the article also affects its visibility on the internet. So the articles chosen for this analysis are also sent through the following two functions to determine the number of sentences and words.

```
def article_sentlength(content):
    s_tokens = sent_tokenize(content
    return len(s_tokens)
def article_wordlength(content):
    w_tokens = word_tokenize(content)
    return len(w_tokens)
```

The tokenisation step is key to perform any further analysis because this is what helps to break down the text into smaller pieces - sentences and then words, allowing for a more comprehensive assessment of the article.

Average number of words in each sentence - This value is calculated using the values obtained from the two previous functions. There is no such specific standard for average number of words in a sentence, but usually sentences have anywhere between 15-30 words.

Average word length in each sentence - This function is used to calculate the average length of words in each sentence using sent_tokenize and then the split() function to extract each word.

```
def avg_sent_word(content):
    s_tokens = sent_tokenize(content)
    for index in range(len(s_tokens)):
    char_ct = 0
    for word in s_tokens[index].split():
        char_ct += len(word)
    print('Sentence ', index + 1, ': ',
        char_ct/len(s_tokens[index].split()))
```

One observation from the results of this function and the next is that, some sentences where the average word length is small could contain more stop words (see HardwareZone article #1 for illustration)

Number of stopwords in each sentence - Stopwords are a really important aspect of search engine optimization (SEO). Stop words are the most prevalent words that many search engines avoid in order to save space and time when crawling or indexing massive amounts of material. This allows search engines to reduce the amount of

data stored in their databases. In our function, a list of stopwords is obtained from the corpus made available in the NLTK library and the stop words from each sentence are extracted.

Count and value of any numerical data in the text - this is performed using regex, and is implemented with a predefined function from the re library. This is more effective than python's inbuilt isdigit() function because with the former, we can we even pick out numbers from words that contain both symbols/alphabets, for example - 'US$59'

Words completely in uppercase/ just capitalised - One use of this function could be to find if there are any abbreviations or acronyms in the article. Apart from this, the function can also filter out all words whose first letter has been capitalised, thus making it easy for any proof-reading, to ensure that all proper nouns and beginning of sentences have been capitalised. Another indication is that completely uppercase words could also be subheadings (see CNA articles for example) The writing style analysis functions were implemented on articles from three different websites -

(1) Hardware Zone
(2) Channel News Asia
(3) StackOverflow

HardwareZone is a Singapore-based technology blog that mainly posts articles and reviews about the latest technology. The two articles chosen to perform analysis are titled -

(1) Zephyr Pro gaming mouse review: Built-in fan keeps you cool (https://www.hardwarezone.com.sg/review-zephyr-pro-gaming-mouse)
(2) Apple's third-generation AirPods features spatial audio, longer battery life, and reimagined design (Updated)(https://www.hardwar news-apple-third-generation-airpods-3-2021-pricing-availability-singapore-pre-order)

Preprocessing - The articles were analysed after scraping using the beautifulSoup package and removing all the unwanted HTML tags.text is ready to be analysed.

```
#script.decompose testing
for script in soup(["script", "style"]):
    script.decompose()
print(' '.join(soup.stripped_strings))
```

Then the main content of the article is picked out by selecting all the paragraph tags, 'p'. Now we can find all other classes/ tags that need to be ignored from the text, such as img tags. After appending the tags to be ignored into a list, those are removed from the text and now the content is ready to be analysed.

```
article = soup.find_all("p")
for div in article:
    print(div)

#identified invalid tags and replace tags
#with their content in each item of soup
invalid_tags = ['img', 'a', 'strong', 'em']
for tag in invalid_tags:
    for match in soup.findAll(tag):
        match.replaceWithChildren()
```

As shown in table 1, Both articles have an average word length that falls in the usual range. The maximum number of stop words found in a sentence are 17 and 13 for the two articles. A count is also performed to calculate the total number of stopwords used in the article. Upon calculating the ratio of stop words to total number of words, it is found that stop words make up 36.4% and 25.88% of article 1 and 2 respectively.

Avg word length Vs. Number of stop words - From the graphs Figure 1 and Figure 2 plotted below, it can be observed that when the number of stopwords is really high, the average word length is relatively lower compared to other sentences.(look at red line in figure 1) Similarly, when the number of stop words are fewer, the average word length in the sentence is also higher. (see black line in graph)

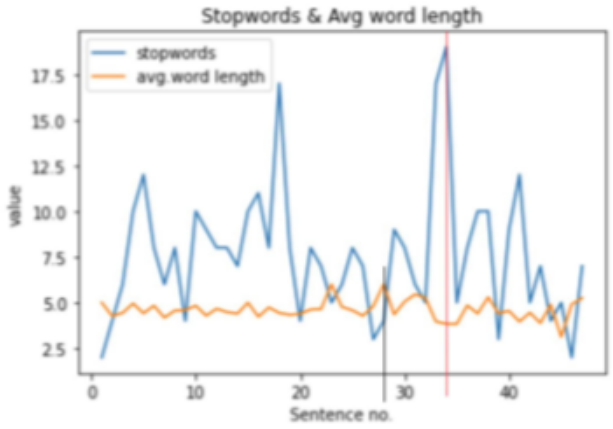

Figure 6: Avg word length in a sentence Vs. Number of stop words in a sentence for Hardware Zone Article 1

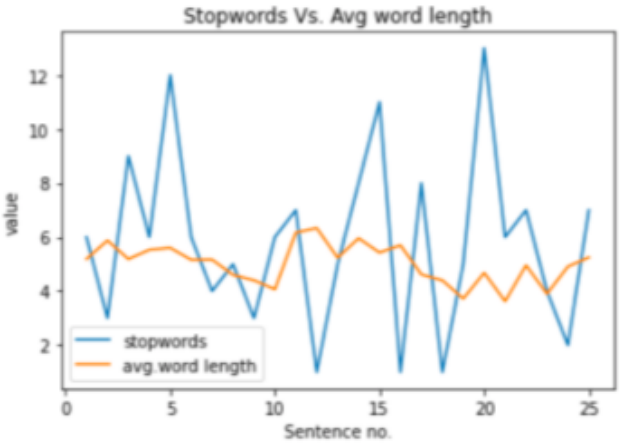

Figure 7: Avg word length in a sentence Vs. Number of stop words in a sentence for Hardware Zone Article 2

After further analysis and verification, this could be very useful when search engines like Google have to crawl through millions of

Function	Article 1	Article 2
Total no. of words	986	561
Total no. of sentences	47	25
Avg. no. of words in a sentence	20.98	22.44
Total number of stopwords	359	146
Max. no of stopwords in a sentence	Sentence 18: 17 words	Sentence 20: 13 words

Table 3: Hardware Zone Analysis

websites, to optimise search timing and results.

Capitalisation Uppercase Words - In the first article, the words with the first letter capitalised consist of those words at the beginning of each sentence as well as proper nouns. Words that are completely in uppercase are either acronyms/ abbreviations or important text that they do not want the reader to miss out on, e.g. 'HARDWAREZONE' is a voucher code specially for the readers of the blog.

The second article has very few words that are completely in uppercase and mostly just consists of words with the first letter capitalized - as they are either the first word of the sentence or proper nouns.

Channel News Asia was established in March 1999 by Mediacorp and is an English language Asian news network[2]. The two articles from CNA that were analysed were titled -

(1) Soaring prices dampen consumer mood in India's festival season (https://www.channelnewsasia.com/business/india-rising-prices-deepavali-diwali-festival-2258601)
(2) Only fully vaccinated employees can return to workplaces from Jan 1; others must test negative (https://www.channelnewsasia fully-vaccinated-employees-can-return-workplace-unvaccinated-test-negative-2263807)

Processing - CNA's articles could not be scraped directly because the site blocks any attempts to read the content using the urlib.open() function. So for the purpose of analysis, the main content has been taken manually. Due to this, no additional preprocessing such as removal of unwanted tags had to be done.

Both the articles have 500-800 words spread over just 15 sentences. The CNA articles are more brief, concise and contain to-the-point information. However, according to our analysis it was found that both articles have a large number of stop words - 39 in one sentence for the first article and 49 in one sentence for the second. Then, upon calculating the ratio of stop words to total number of words, it is found that stop words make up 32.87% and 34.6% of article 1 and 2 respectively. On an average, the percentage of stopwords in the CNA articles is higher when compared to that of the Hardwarezone articles

Avg word length Vs. Number of stop words - Similar to the previous Hardwarezone articles, sentences with a higher number of stopwords tend to have a smaller average word length. (Figure 3 & Figure 4)

Capitalisation Uppercase Words - In the first CNA article, the words that have their first letter capitalised are mostly proper nouns - as the article is about the festival spirit in India, and therefore many proper nouns, pertaining to places and people, have been

Figure 8: Avg word length in a sentence Vs. Number of stop words in a sentence for CNA Article 1

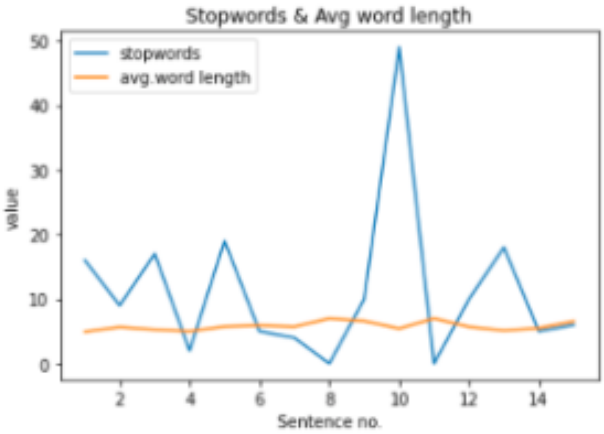

Figure 9: Avg word length in a sentence Vs. Number of stop words in a sentence for CNA Article 2

mentioned. Unlike the Hardwarezone articles, there are no calls-to-action or any offer codes in the CNA articles. Most uppercase words are acronyms such as COVID-19 or proper nouns such as the location of reporting.

Stack Overflow's public platform is used by nearly everyone who codes to learn, collaborate and share their technical knowledge. The two articles being analysed are titled -

Function	Article 1	Article 2
Total no. of words	794	491
Total no. of sentences	15	15
Avg. no. of words in a sentence	52.93	32.73
Total number of stopwords	261	170
Max. no of stopwords in a sentence	Sentence 14: 39 words	Sentence 10: 49 words

Table 4: CNA Analysis

(1) I'm getting Key error in Python (https://stackoverflow.com/questions/10116518/im-getting-key-error-in-python)
(2) python - beautifulsoup - TypeError: sequence item 0: expected string, Tag found (https://stackoverflow.com/questions/14264102/python-beautifulsoup-typeerror-sequence- item-0-expected-string-tag-found)

Processing - Both the articles were extracted from the url using the beautifulSoup package and all the cleaning functions, just as for the Hardwarezone articles are performed. One additional preprocessing step was to separate the code tags from the main article.

The issue with Stack Overflow's articles is that - since most of their posts are actually questions which have an answering thread, it is not one cohesive article. Further, since it is a technical website, there is a lot of code embedded into the text. Sometimes, code is a major portion of the entire article. Thus performing tokenization and then any form of POS tagging or stemming wouldn't be easy to do on the article as a whole. It might be necessary to split the article into each individual thread in order to analyse it. Furthermore, we cannot use the same tools for analysing both text and code. So this is another point that needs to be considered before trying to understand the writing style of such articles.

In this concluding table, we observe the percentage of stopwords occurring in each article analysed. The distribution of the stopwords is not equal in each sentence, nor is there any pattern seen in articles from the same website. However, it is interesting to note that all the articles have the stopwords percentage ranging from 29-37%. Which, upon further analysis with many more articles could be an important judgement metric for identifying the amount of important useful content in articles.

2.4 Most Frequent Noun-Adjective Pairs

After organising the data by splitting into dataframes based on rating and randomised business IDs, the datasets are passed into the method noun_adj_pairs() that return a list of noun-adjective pairs.

```
nlp = spacy.load('en')
def noun_adj(path):
 noun_adj_pairs = []
 doc = nlp(path)
 for i,token in enumerate(doc):
     if token.pos_ not in ('NOUN','PROPN'):
         continue
     for j in range(i+1,len(doc)):
         if doc[j].pos_ == 'ADJ':
             noun_adj_pairs.append((token,doc[j]))
             break
 return noun_adj_pairs
```

Using the list obtained from the method, we use the Counter() method available in the collections package and obtain the 10 most common noun-adj pairs available in the list. Table 7 shows the outputs we obtained for each rating dataset.

Fig 10 is an example of a challenge that we face, where the POS is tagged wrongly to certain words. For instance, 'ENTIRE' is an adjective but is identified as a noun. Similarly 'mine' is interpreted as a noun whereas, contextually, 'mine' is a possessive pronoun. 'am' is a verb, but is tagged as a noun. This is a major limitation of using spaCy's POS tagging approach to identify noun-adjective pairs.

```
(opinion, original),(availability, original),
(ENTIRE, original),(DAY, original),
(company, original),(Yelp, original),(part, original),
(review, first),(visit, young),(Aaron, young),(pools, young),
(man, bad),(attitude, new),(pool, new),(property, new),
(motor, new),(Pros-, new),(tech, new),(time, new),
(Fri, new),(am, new),(tech, new),
(problem, new),(Cons-, new),(parts, new),(store, new),
(week, new),(job, new),(mine, new),(office, new),
(explanation, new),(Saturday, new)
```

Figure 10: Sample noun-adjective outputs

3 EXTRACTION OF INDICATIVE ADJECTIVE PHRASES

In this section, we were tasked to extract the most indicative adjective phrases from the reviews for a randomly selected business. After an initial understanding of adjective phrases, we formulated a table (Table 8) of grammar rules that a phrase must follow in order to qualify as an adjective phase.

In order to extract adjective phrases from the reviews, the reviews are first extracted into a list (reviews_txt). This list of reviews (reviews_txt) is passed into a method that returns a list of tokens (result) for the texts. The nlp()method that spaCy offers was used to segment words from reviews.

```
def tok_pos(reviews_txt: list):
   processed_txt = []
   x = nlp(str(reviews_txt))
   for token in x:
       process = []
       process.append(token.text)
```

Function	Article 1	Article 2
Total no. of words (excluding code)	555	382
Total no. of sentences	25	14
Avg. no. of words in a sentence	22.2	25.86
Total number of stopwords	164	106
Max. no of stopwords in a sentence	Sentence 20: 32 words	Sentence 6: 17 words

Table 5: Stack Overflow Analysis

Article	Total Number of Words	Avg. words per sentence	Total Number of stopwords	% Stopwords in article
HardwareZone 1	986	20.98	359	36.41
HardwareZone 2	561	22.44	146	25.88
CNA 1	794	52.93	261	32.87
CNA 2	491	32.73	170	34.62
Stack Overflow 1	555	22.2	164	29.55
Stack Overflow 2	362	25.86	106	29.28

Table 6: Overall %stopword Analysis

Rating	Sample Size	Top 10 Most Frequent Noun-Adjective Pairs
1	50	[(('food', 'Korean'), 8), (('place', 'such'), 6), (('abomination', 'such'), 6), (('service', 'good'), 6), (('meal', 'good'), 6), (('tonight', 'good'), 6), (('friend', 'avid'), 6), (('birthday', 'avid'), 6), (('meal', 'avid'), 6), (('BBQ', 'worst'), 6)]
2	20	[(('cup', 'awesome-'), 6), (('water', 'awesome-'), 6), (('SW', 'awesome-'), 6), (('bagel', 'awesome-'), 6), (('barista', 'seasonal'), 6), (('city', 'representative'), 4), (('break', 'best'), 4), (('items', 'other'), 4), (('syrups', 'delicious'), 4), (('sourdough', 'multi'), 4)]
3	20	[(('water', 'warm'), 9), (('people', 'quick'), 6), (('people', 'amazing'), 6), (('menu', 'amazing'), 6), (('beef', 'middle'), 6), (('chicken', 'middle'), 6), (('veggies', 'better'), 6), (('Humus', 'ok'), 6), (('food', 'warm'), 6), (('store', 'last'), 6)]
4	20	[(('Denny', 'big'), 4), (('cream', 'good'), 3), (('pancakess', 'fluffy'), 3), (('light', 'fluffy'), 3), (('Hubby', 'french'), 3), (('strawberry', 'french'), 3), (('toast', 'delicious'), 3), (('Georgia', 'positive'), 3), (('recommend','positive'), 3), (('thisnplace', 'positive'), 3)]
5	20	[(('food', 'good'), 4), (('food', 'amazing'), 3), (('service', 'little'), 3), (('while', 'worth'), 3), (('food', 'worth'), 3), (('wait', 'big'), 3), (('tacos', 'big'), 3), (('meat', 'amazing'), 3), (('burgers', 'amazing'), 3), (('BREAKFAST', 'free'), 2)]

Table 7: 10 Most Common Noun-Adjective Pairs for each rating

	Grammar Rules
Bigram adjective phrases	ADV - ADJ ADJ - ADV ADJ-ADJ
Trigram adjective phrases	ADJ-ADP-NOUN ADV-ADV-ADJ
4-gram adjective phrases	ADJ-ADP-ADP-NOUN

Table 8: Adjective Phrase Grammar Rules

```
        process.append(token.pos_)
        processed_txt.append(process)
    return processed_txt,x
processed_txt,result = tok_pos(reviews_txt)
```

The doc object (result) obtained from the above method is now used as input for the method adj_phrases_extraction(), used to extract adjective phrases. Below is the pseudocode of this method. The method returns 3 lists (bigram adjective phrases list, trigram adjective phrases list and 4-gram adjective phrases list) and 3 integer values (count of bigram, trigram and 4-gram adjective phrases).

```
Function: adj_phrases_extraction(token_list):
Input: doc object
Output: 3 lists and 3 integer values

adj_bigram, adj_trigram, adj_ngram <- 0
count_bi, count_tri, count_n <- 0
for idx = 0 -> length(token_list)-2 do
        prev <- token_list[idx-1]
        current <- token_list[idx]
```

Adjective Phrases	Count
Bigram adjective phrases	279
Trigram adjective phrases	18
4-gram adjective phrases	0

Table 9: Adjective Phrases Distribution

```
        next <- token_list[idx+1]
        next_next <- token_list[idx+2]
        if (POS tag of prev and current
          satisfy bigram adjective
             phrase structure) then
                adj_bigram.append((prev, current))
                count_bi <- count_bi + 1
        end if
        if (POS tag of prev, current
          and next satisfy trigram
            adjective phrase structure) then
                adj_ngram.append((prev,
                            current, next))
                count_tri <- count_tri + 1
        end if
        if (POS tag of prev, current, next and
            next_next satisfy 4-gram adjective
                phrase structure) then
                adj_trigram.append((prev,
                    current, next, next_next))
                count_n <- count_n + 1
        end if
end for
return adj_bigram, count_bi, adj_trigram,
    count_tri, adj_ngram, count_n
end procedure
```

Table 9 below shows a frequency distribution of bigram, trigram and 4-gram adjective phrases. Figure 11 shows a sample output containing trigram adjective phrases from a corpus of text reviews extracted from the randomly selected business - YgWGWza3sXzI0brcOogR5A.

In order to obtain the most indicative adjective phrases, we introduce a parameter, "Indicativeness".

$$Indicativeness = \frac{Frequency of given adjective phrase}{Total number of reviews}$$

```
def adj_ind (adj_phrase: list):
    freq_dist = nltk.FreqDist(adj_phrase)
    freq_dist_top20 = freq_dist.most_common(20)
    freq_dict = {}
    for i in freq_dist.most_common(20):
      freq = i[1]/len(reviews_txt)
      freq_dict[i[0]] = freq
    return freq_dict
```

The above code snippet shown defines a method adj_ind() that calculates the indicativeness of the 20 most frequent adjective phrases and returns a dictionary containing adjective phrases as keys and their respective indicativeness as values. We pass the adjective phrases extracted through adj_phrases_extraction() into this method and then concatenate the results and extract the 10 most indicative adjective phrases used in reviews for the chosen business.

Figure 12 below shows a dictionary containing the top 10 most indicative adjective phrases used in the reviews along with their respective indicativeness values for business YgWGWza3sXzI0brcOogR5A.

In order to understand the uniqueness of the adjective phrases selected in business YgWGWza3sXzI0brcOogR5A. In order to see the relationship, we manually repeat the above steps for each of the businesses in the dataset. Appendix A is a sample of most indicative adjective phrases extracted from two other businesses, suJ-eo5Gkvcg0iXVQw4RyA and 84DjKzaR26vphu9fNI9nKg. We can see the presence of some generic adjective phrases such as ('very', 'nice') and ('very', 'friendly') present in the most indicative adjective phrases dictionary of both these businesses.

When comparing the frequency of the chosen indicative adjective phrases across reviews from all the businesses, we can see a certain trend. Whilst the generic adjective phrases such as ('very', 'nice') that score high on indicativeness within the selected business itself, we also can observe that it is highly indicative in other businesses also, hence does not attribute to the uniqueness of the business. However, some other phrases such as ('very', 'thorough'),('extremely', 'clean') and ('most', 'modern') are almost absent in the most indicative adjective phrases list for other businesses indicating that they are unique to the chosen business.

4 APPLICATION

For this assignment, we developed a simple NLP application that carries out sentiment analysis of the reviews. Sentiment analysis, one of the most popular applications of Natural Language Processing, refers to the process of identifying, extracting, and quantifying subjective information in textual data. For this project, sentiment analysis is used to determine whether a review is positive, negative or neutral.

The accuracy of a sentiment analysis application primarily depends on the data pre-processing and type of neural network architecture used. Thus, for our project, we pre-processed the reviews dataset, split the dataset into train, test, and validation sets, and trained a model based on the Long Short-Term Memory (LSTM) network architecture. We discuss the working of the application in more detail in the following paragraphs.

DataPre – processing - First, in order to create a ground-truth column for the sentiments expressed in the given reviews dataset, we decided to use the review ratings (number of stars) column. The star ratings are integer values in the range [1, 5], and we assigned ratings of 1, 2, and 3 stars a "negative" label, with ratings of 4 and 5 stars being assigned a "positive" label. The ratings were binary-coded using list comprehension to process the text reviews, and 1 was assigned for positive reviews, and 0 for negative reviews.

```
[('nice', 'at', 'times'), ('Very', 'well', 'pleased'), ('so', 'very', 'nice'),
('nervous', 'about', 'dentists'), ('So', 'incredibly', 'happy'), ('open', 'for',
'hours'), ('no', 'longer', 'afraid'), ('so', 'terribly', 'rude'), ('just',
'really', 'nice'), ('even', 'more', 'amazing'), ('very', 'very', 'nice'),
('advanced', 'in', 'technology'), ('very', 'very', 'picky'), ('kind', 'of',
'traumatized'), ('still', 'super', 'clean'), ('all', 'too', 'familiar'),
('familiar', 'with', 'dentists'), ('comfortable', 'with', 'beverages')]
```

Figure 11: Sample output of trigram adjective phrases

```
{('extremely', 'clean'): 0.02,
 ('most', 'modern'): 0.02,
 ('so', 'gentle'): 0.04,
 ('so', 'glad'): 0.05,
 ('so', 'nice'): 0.06,
 ('so', 'sweet'): 0.03,
 ('very', 'gentle'): 0.03,
 ('very', 'nice'): 0.07,
 ('very', 'professional'): 0.02,
 ('very', 'thorough'): 0.03}
```

Figure 12: Top 10 most indicative adjective phrases

It must be noted that there is no label for "neutral" reviews as we deemed that the information gained by considering neutral reviews was insignificant; instead, we decided to develop a binary sentiment analysis application.

Next, we processed the text in each review to make it suitable for use with our LSTM model. The text in each review was integer encoded and tokenized, including converting the text to lowercase for increased consistency. Each unique token was then mapped to a unique integer. The data was then split into train, test, and validation sets, following a 70:20:10 split for the respective sets.

LongShort − TermMemory(LSTM)Classifier - Long Short Term Memory networks (LSTMs) are a variant of Recurrent Neural Networks (RNNs) that have the ability to learn long-term dependencies. The following is an overview of what RNNs are and how they work.

Unlike normal neural networks that are linear (i.e. the data is fed forward and then back-propagated to optimize the weights and biases of each layer), an RNN is actually recursive in nature. An RNN contains multiple loops in each layer so that information can persist between consecutive layers - which is why they are called "recurrent" networks. An RNN tries to create a Language Model (LM) by reading a text corpus from left to right and remembering the context of each word in relation to its position in the sentence. However, as sentences get long and complicated, conventional RNNs have difficulties remembering words seen in the earlier parts of the input sentence. Thus, the LSTM architecture was developed to fix this issue of so-called "long-term dependencies".

Each layer of an LSTM network has a sigmoid layer coupled with a pointwise multiplicator that outputs a floating point value between 1 and 0. This layer acts as a "gate" that controls the information flow to each layer, effectively allowing the network to add or remove information according to what it deems as relevant. LSTMs allow data to be passed through the layers multiple times until the LM can be constructed. Then, an LM can analyze a text corpus and try to predict (with a higher accuracy relative to traditional RNNs) the next word in a sentence based on statistical inferences on the probability of the occurrence of a specific word sequence in a sentence.

For our NLP application, we developed the LSTM-based model using TensorFlow 2 (Keras). Our model uses TensorFlow's Sequential API, and consists of an Embedding layer, LSTM layer, and a Dense (i.e. Fully Connected) output layer. The embeddings that are input to the LSTM refers to the representation of text as numerical feature vectors of a fixed length. The embedding size was set based on the vocabulary size and maximum length of input sequences. For our LSTM layer, we set the number of units to 200, used a tanh activation function, and applied kernel (weight) and bias L2 regularization to address overfitting to the training set. For the Dense output layer, the number of neurons was 1 and we used a sigmoid activation function. Lastly, the loss function used is binary cross-entropy loss, and the Adaptive Moment estimation (Adam) optimizer was used for gradient descent.

When trained for 50 epochs, our model was able to achieve a high training accuracy of 95%, validation accuracy of 85%, and test accuracy of 82%.

To assess the weaknesses of our model, we further tested the model on the full reviews dataset, and checked for incorrect predictions. The data was passed to the trained model and a new column corresponding to the predicted sentiment with the column name "Sentiment" was added to the Pandas dataframe storing the data.

For example, for the review with a 5-star rating, the model has incorrectly predicted it as a negative statement. Nevertheless, the ratio of misclassified sentiments to the correctly classified sentiments is negligible, which indicates that the model is adequately reliable for the application.

Task	Teammate
Tokenization and Stemming	Das Atrik
POS Tagging	Das Atrik
Writing Style	Unnikrishnan Malavika
Most frequent Noun-Adj Phrases	Padhi Adbhinandan
Extraction of Indicative Adjective Phrases	Asuri Simhakutty Kamakshi
Application	Musthiri Sajna Padhi Abhinandan

Table 10: Contributions

Thus, although there are few inaccuracies, we think the model still serves its purpose of analysing the sentiment expressed in the reviews in the given dataset.

REFERENCES

[1]"Application of RNN for customer review sentiment analysis", Medium, 2021. [Online]. Available: https://towardsdatascience.com/application-of-rnn-for-customer-review-sentiment-analysis-178fa82e9aaf. [Accessed: 25- Oct- 2021]

[2]"Sentiment Analysis using Python [with source code]". [Online]. Available: https://techvidvan.com/tutorials/python-sentiment-analysis/. [Accessed: 25- Oct- 2021]

[3]A. Dodge, "4 Statistics Every Blogger Should Know About Content Word Count", Content Marketing Agency | Content Marketing Services by CopyPress. [Online]. Available: https://www.copypress.com/blog/4-statistics-every-blogger -should-know-about-content-word-count/. [Accessed: 19- Oct- 2021].

[4]"About Us", CNA. [Online]. Available: https://www.channelnewsasia.com/about-us. [Accessed: 22- Oct- 2021].

A APPENDIX

A.1 Appendix A

suJ-eo5Gkvcg0iXVQw4RyA
('very', 'nice'): 0.09, ('very', 'friendly'): 0.06, ('really', 'nice'): 0.05, ('very', 'good'): 0.05, ('much', 'better'): 0.05, ('very', 'clean'): 0.04, ('super', 'friendly'): 0.03, ('super', 'nice'): 0.03, ('African', 'American'): 0.02, ('extremely', 'rude'): 0.02, ('how', 'disappointed'): 0.02, ('several', 'other'): 0.02, ('pleasantly', 'surprised'): 0.02, ('very', 'helpful'): 0.02, ('so', 'tired'): 0.02, ('free', 'happy'): 0.02, ('very', 'impressed'): 0.02, ('extremely', 'disappointed'): 0.02, ('how', 'nice'): 0.02, ('completely', 'unprofessional'): 0.02, ('of', 'too', 'big'): 0.01, ('same', 'for', 'soap'): 0.01, ('free', 'of', 'charge'): 0.01, ('now', 'so', 'modern'): 0.01, ('separate', 'from', 'bath'): 0.01, ('tasty', 'with', 'lots'): 0.01, ('convenient', 'to', 'downtown'): 0.01, ('full', 'of', 'kids'): 0.01, ('more', 'then', 'other'): 0.01, ('cold', 'with', 'cheese'): 0.01, ('Really', 'really', 'poor'): 0.01, ('good', 'to', 'curve'): 0.01, ('just', 'awfully', 'bad'): 0.01, ('great', 'with', 'tons'): 0.01, ('right', 'at', 'home'): 0.01, ('dirty', 'with', 'rust'): 0.01, ('concerned', 'about', 'guests'): 0.01, ('also', 'very', 'good'): 0.01, ('FULL', 'OF', 'SOMEONES'): 0.01

84DjKzaR26vphu9fNI9nKg
('pretty', 'good'): 0.1, ('very', 'good'): 0.05, ('very', 'friendly'): 0.05, ('very', 'attentive'): 0.03, ('too', 'bad'): 0.03, ('pretty', 'decent'): 0.03, ('more', 'upscale'): 0.02, ('very', 'nice'): 0.02, ('Very', 'disappointed'): 0.02, ('too', 'busy'): 0.02, ('so', 'much'): 0.02, ('good', 'as'): 0.02, ('too', 'much'): 0.02, ('as', 'good'): 0.02, ('little', 'better'): 0.02, ('pretty', 'dead'): 0.02, ('so', 'awesome'): 0.01, ('pretty', 'consistent'): 0.01, ('how', 'bad'): 0.01, ('near', 'perfect'): 0.01, ('best', 'in', 'town'): 0.02, ('far', 'more', 'courteous'): 0.01, ('equally', 'as', 'attentive'): 0.01, ('good', 'from', 'time'): 0.01, ('sort', 'of', 'friendly'): 0.01, ('amazing', 'on', 'tap'): 0.01, ('dead', 'during', 'football'): 0.01, ('Good', 'for', 'loners'): 0.01, ('always', 'very', 'friendly'): 0.01, ('little', 'more', 'upscale'): 0.01, ('very', 'very', 'clean'): 0.01, ('outrageous', 'for', 'bar'): 0.01, ('also', 'pretty', 'cool'): 0.01, ('decent', 'for', 'bar'): 0.01, ('much', 'less', 'crowded'): 0.01, ('little', 'under', 'par'): 0.01, ('sort', 'of', 'gross'): 0.01, ('actually', 'very', 'good'): 0.01, ('very', 'very', 'good'): 0.01

A.2 Appendix B

Analysis across all businesses

```
For the business:  YgWGWza3sXzI0brcOogR5A
For the phrase:  ('so', 'glad')

 Other businesses
Business: CQxhyLM833WF45yrS9yeAA Probability:  0.04
Business: mSqR24h_nKXyMhwtWSih3Q Probability:  0.03
Business: hXzoNgpkC86K_Jfg_zMHvA Probability:  0.02
Significant difference in indicativeness of the given adjective phrase

Business: x3LoVFuzVIPK_J-kDZORdg Probability:  0.02
Significant difference in indicativeness of the given adjective phrase

Business: aDHD7nASfqiQBB6YXy2aGA Probability:  0.01
Significant difference in indicativeness of the given adjective phrase

Business: IT8amJTTW64XAre4ilKcjg Probability:  0.05
Business: QaFO4S6HFUu2NIbeu4OwCg Probability:  0.01
Significant difference in indicativeness of the given adjective phrase

Business: GwKq3kjkFXhbBMAwe4H3rg Probability:  0.03
Business: CSGJxWQwD_QfOKxp569grQ Probability:  0.04
Business: OjnRf8yDGEBCoUDdchSViw Probability:  0.02
Significant difference in indicativeness of the given adjective phrase

Business: grvkjaJB9aorAMIjGD6XSg Probability:  0.02
Significant difference in indicativeness of the given adjective phrase

Business: shIPnFoXrL3dFo5HLH1_HA Probability:  0.04
Business: YgWGWza3sXzI0brcOogR5A Probability:  0.05
Business: ApeTrSttf8f4KUrBVtcKgw Probability:  0.03
Business: AT5NZHqoLeP4rhWMM2OWBA Probability:  0.02
Significant difference in indicativeness of the given adjective phrase
```

Figure 13: Appendix B

```
For the business:  YgWGWza3sXzI0brcOogR5A
For the phrase:  ('very', 'professional')

 Other businesses
Business: qMyYY8zpHusOXUd6wmqX2w Probability:  0.04
Business: 7e3PZzUpG5FYOTGt3O3ePA Probability:  0.03
Business: 4tjKsIaBXCDUP9PL82Vu_A Probability:  0.02
Business: QaFO4S6HFUu2NIbeu4OwCg Probability:  0.07
Business: CSGJxWQwD_QfOKxp569grQ Probability:  0.04
Business: UuO28w986H0CO_HJluQmew Probability:  0.02
Business: eWmARaQb_9vxq-zy5PVvXA Probability:  0.05
Business: grvkjaJB9aorAMIjGD6XSg Probability:  0.04
Business: YgWGWza3sXzI0brcOogR5A Probability:  0.02
Business: 3kUqNxO1rkDDb89GAfyNgw Probability:  0.04

_____________________________________________________

For the business:  YgWGWza3sXzI0brcOogR5A
For the phrase:  ('most', 'modern')

 Other businesses
Business: YgWGWza3sXzI0brcOogR5A Probability:  0.02

_____________________________________________________

For the business:  YgWGWza3sXzI0brcOogR5A
For the phrase:  ('extremely', 'clean')

 Other businesses
Business: YgWGWza3sXzI0brcOogR5A Probability:  0.02

_____________________________________________________
```

Figure 14: Appendix B