# **CZ4045 Natural Language Processing Assignment**

Chan Wei Chang

U1622381K

wchan027@e.ntu.edu.sg

Yan Ran

U1320651F

yran001@e.ntu.edu.sg

# **1. INTRODUCTION**

The following report contains the results and analysis for Natural Language Processing (NLP) assignment. This report will be segmented into 5 main sections, the first 4 sections are similar to the sections given in the assignment, Dataset Analysis, Development of a Noun Phrase Summarizer, Sentiment Word Detection, and Application. The last section will be a conclusion for this assignment.

# **2. Dataset Analysis**

This section will cover all the tasks given within the Dataset Analysis section of the assignment. Given the size of the dataset, the data will be read in line by line from the file instead of the loading the entire dataset into memory as shown in Listing 1. Each line corresponds to a JSON object, which can be the index with a property field.



**Listing 1: An example of loading dataset line by line**

The programming language used for this assignment is Python and the Natural Language Processing library of choice is NLTK.

**2.1 Top-10 Product and reviewer**

The first task of dataset analysis is to identify the top-10 products and reviewers from the dataset. Each line being read is a JSON object, thus to get the asin (productid) and reviewerID of each review, the appropriate property field (e.g. asin, reviewerID) is used to index the JSON object.

A python dictionary data structure is used to keep track of the number of reviews related to each product and reviewer. The keys in the dictionary are the asin values and reviewerID, the value associated with each key is the number of times the keys are seen in the dataset. Each time a new asin value or reviewerID is seen, a new key is added to the dictionary with a value of 1. If a asin or reviewer has been registered as a key before, it will increment the value associated with it.

**Table 1. Top 10 Products table**

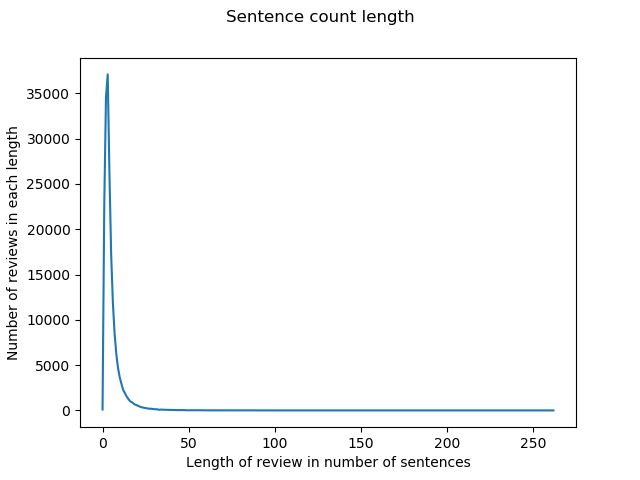
|  |  |
| --- | --- |
| Top 10 Product | |
| ASIN (ProductID) | Number of Reviews associated with |
| B005SUHPO6 | 836 |
| B0042FV2SI | 690 |
| B008OHNZI0 | 657 |
| B009RXU59C | 634 |
| B000S5Q9CA | 627 |
| B008DJIIG8 | 510 |
| B0090YGJ4I | 448 |
| B009A5204K | 434 |
| B00BT7RAPG | 431 |
| B0015RB39O | 424 |

**Table 2. Top 10 Reviewer table**

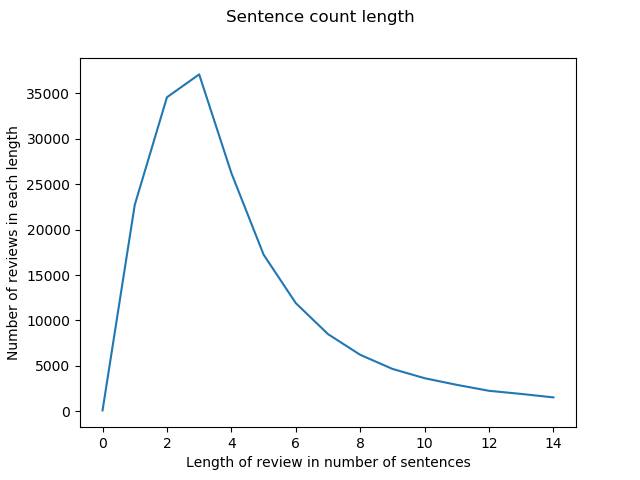
|  |  |
| --- | --- |
| Top 10 Reviewer | |
| ReviewerID | Number of Reviews associated with |
| A2NYK9KWFMJV4Y | 152 |
| A22CW0ZHY3NJH8 | 138 |
| A1EVV74UQYVKRY | 137 |
| A1ODOGXEYECQQ8 | 133 |
| A2NOW4U7W3F7RI | 132 |
| A36K2N527TXXJN | 124 |
| A1UQBFCERIP7VJ | 112 |
| A1E1LEVQ9VQNK | 109 |
| A18U49406IPPIJ | 109 |
| AYB4ELCS5AM8P | 107 |

## **2.2 Sentence Segmentation**

To perform segmentation, reviews will be retrieved from the JSON object by indexing it with ‘reviewText’. NLTK sentence segmentation method will then be applied to each review. Figure 1 shows the distribution of the data. The x-axis is the length a review in the number of sentences, and the y-axis is the number of reviews of each length. Figure 2 shows the same distribution but focused on the area where most of the reviews are.

****

**Figure 1. Sentence segmentation distribution**

****

**Figure 2. Sentence segmentation distribution**

**(Zoomed into peak)**

From Figure 1, it can be seen that most of the review length are of shorter length. Then from Figure 2, we can see that most reviews have 2 to 3 sentences. This shows that most reviewers tend to write their reviews with just a few sentences instead a long detailed review.

Five reviews are chosen at random from the dataset to verify whether the sentence segmentation of NLTK detects the sentence boundaries correctly. Two of the reviews are long reviews, which has more than five sentences. While the other three reviews are short reviews, which has less than 5 sentences. The random selection is done with the code shown on Listing 2, each sentence has a 5 percent chance of getting chosen. All the random selection throughout this section is done in a similar way.



**Listing 2: Code for random chance**

**Table 3. Example of sentence segmentation on long review**

|  |  |
| --- | --- |
| 1 | 'This is the first battery case I have had for my Galaxy S4.' |
| 2 | ‘The S4 fits very well, is slim and doesn't add much weight to the Galaxy S4.’ |
| 3 | 'It doubles the battery life.' |
| 4 | 'You can charge either the battery, the phone or both.' |
| 5 | 'There is a handy on-off switch with leds to indicate the level of charge.The battery case came on time and was packaged well.' |
| 6 | 'Well worth the price.' |

**Table 4. Original review**

|  |
| --- |
| "This is the first battery case I have had for my Galaxy S4. The S4 fits very well, is slim and doesn't add much weight to the Galaxy S4. It doubles the battery life. You can charge either the battery, the phone or both. There is a handy on-off switch with leds to indicate the level of charge.The battery case came on time and was packaged well. Well worth the price." |

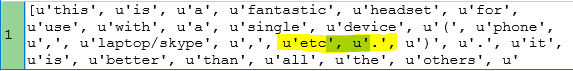
Table 3 shows the result of one the long reviews, each segmented sentence is placed on a different row with the sentence number on the first column. All the sentences appears to be segmented properly, other than the fifth sentence as it has two sentences in it. This is due to the fact that there is no space between the end of the first sentence and the start of the second sentence. This is also observed from the original review shown at Table 4, with the problematic part highlighted.

Typical in a paragraph, there will be a space between the end of one sentence and the start of the next sentence. NLTK sentence segmentation library seems to work under this assumption, which explains why it fails in this scenario while the other four sentence are being segmented properly without any issues.

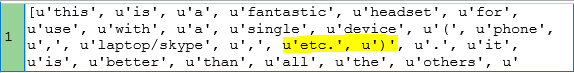
Therefore, there is a limitation to NLTK sentence segmentation. If the text to be segmented is not formatted properly, the NLTK sentence segmentation might not work. The given dataset are reviews where people might not follow the proper syntax, thus the result of NLTK sentence segmentation might not be very good.

## **2.3 Tokenization and Stemming**

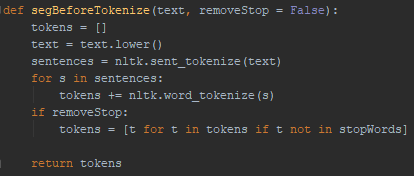
Tokenization is done with NLTK tokenizer, the tokenizer method takes in the text to be tokenize and returns an array, in which each element is a token. Tokenization can be done with or without sentence segmentation. However, by counting the total number of tokens, it shows that there is a difference. The total number of tokens if tokenization is done after sentence segmentation is 19931559. If tokenization is done without sentence segmentation, there will be 19922117 tokens. There is a difference of almost ten thousand, to find out the difference I printed out the result of the tokenization if the number of tokens from the two methods does not match. Figure 2 and 3 will show one such result.



**Figure 3. Tokenization after sentence segmentation**



**Figure 4. Tokenization without sentence segmentation**

****

**Listing 3: Code for tokenization after sentence segmentation**

Tokenization after sentence segmentation produces better result than without sentence segmentation. Note that the leading u before each token means that the string is Unicode, thus it can be ignored. This can be confirmed with the results shown in Figure 3 and 4. In Figure 3, the word ‘etc’ and ‘.’ is identified as two separate tokens instead of one like in Figure 4. This is because sentence segmentation is able to identify that a full stop followed by a closing bracket is an indication of a new sentence. Thus, sentence segmentation will separate the sentence ending with full stop from the sentence starting with the closing bracket. Therefore, all tokenization mentioned from here on will perform sentence segmentation before tokenization.

NLTK provides multiple stemming methods, in order to find out the best stemmer, first a corpus is needed to check whether the word exist in the corpus. To find the best suited corpus for this dataset, four different corpus are compared against each other (WordNet, Brown, Product Reviews, Word Lists). Tokenization will be performed on all reviews, followed by checking whether each token exist in the corpus and a counter will be incremented if it exist. Words in the corpus are converted to lower letters to prevent the letter case from interfering with the results.

**Table 5. Number of tokens found in each corpus**

|  |  |
| --- | --- |
| Corpus | # Tokens found in it |
| WordNet | 10846856 (10 million) |
| Brown | 18343518 (18.3 million) |
| Product Reviews | 18043427 (18 million) |
| Word Lists | 15463534 (15.4 million) |

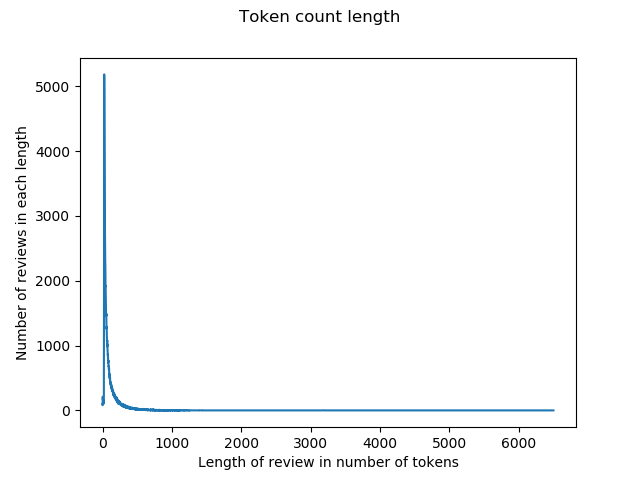
Based on the results shown on Table 5, Brown corpus is the best suited corpus for this dataset. Now, two different type of stemmer will be compared against each other, Porter stemmer and Snowball stemmer. Tokenization will again be performed first, followed by stemming and finally checking if the stemmed token exist in the Brown corpus.

**Table 6. Number of stemmed tokens found in Brown corpus**

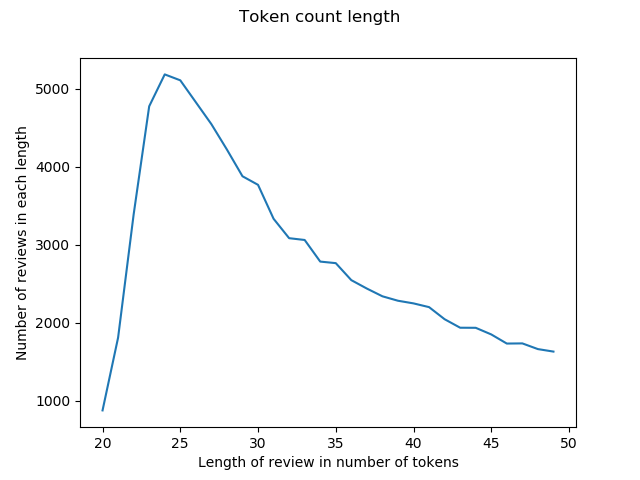
|  |  |
| --- | --- |
| Stemmer | # tokens found in Brown |
| Porter | 16121075 (16.1 million) |
| Snowball | 16602499 (16.6 million) |

Based on the results shown on Table 6, Snowball stemmer clearly out performs Porter stemmer, with 500 thousands more tokens being stemmed into words that exist. However, stemming has resulted in 1.7 million tokens being incorrectly stemmed when compared to the amount of unstemmed tokens found in brown corpus (18.3 million). Since Snowball stemmer performs better, the stemmer used for the rest of the report will be Snowball stemmer.

To find how much stemming affects the number of token types, we compare the number of tokens before and after stemming. Before stemming there are 209343 (209 thousand) distinct tokens, after stemming is performed there are 185785 (185 thousand) almost 24 thousand distinct tokens are removed after stemming.



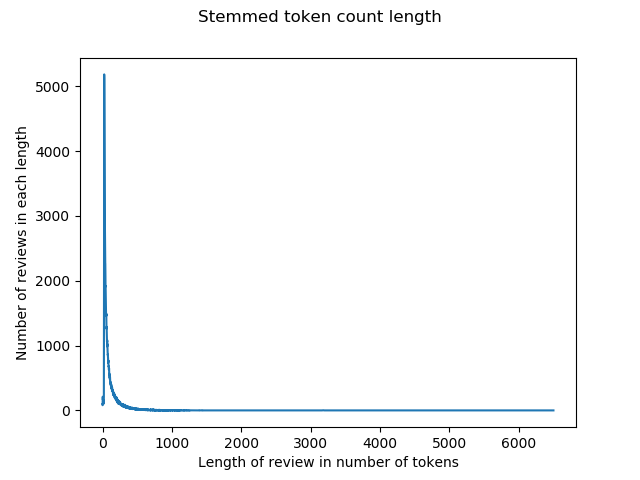
**Figure 5. Tokenization without stemming distribution**

****

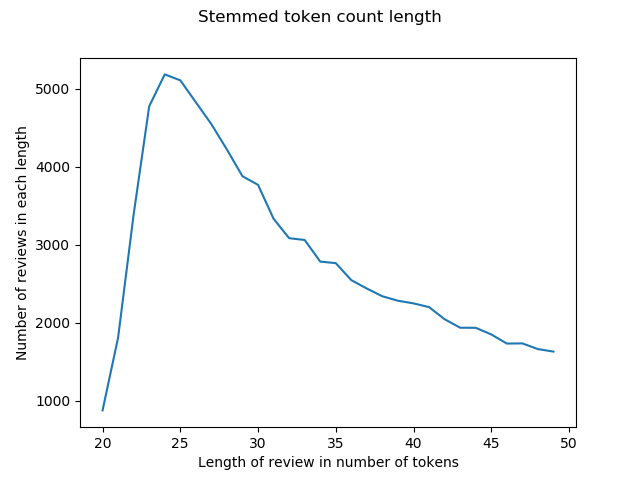
**Figure 6. Tokenization without stemming distribution (Zoomed into trending area)**

Figure 5 shows the tokenization without stemming distribution, it almost similar to the sentence segmentation distribution with most the reviews clustered within a range.

Figure 6 shows the zoomed in portion where most reviews are, it also has a similar looking graph as the zoomed in portion of sentence segmentation. However, the dipping of the graph appears to be more gradual compared to the sentence segmentation graph. Which indicates that the number of tokens does not increase much as the number of sentences increases. This might be due to reviewers writing expressive sentences such as “Great product!”, which contributes to the number of sentences but not so much to the number of tokens.

****

**Figure 7. Tokenization with stemming distribution**

****

**Figure 8. Tokenization with stemming distribution (Zoomed into trending area)**

Since stemming only changes the token to its root form, the number of tokens per review should remain the same as the amount before stemming. The only difference is the number of distinct tokens, which was discussed previously. Thus, the distribution for tokenization with stemming would be the same as the distribution for tokenization without stemming.

In order to find the top-20 most frequent words before and after stemming, we first have to come up with a list of stop words. We will be using the pool of stop words provided by NLTK and in additional to that, punctuations will also be added into the pool of stop words. As punctuations such as full stop commonly occurs through the dataset, and having it as one of the top-20 most frequent words does not carry much meaning.

**Table 7. Top-20 words before stemming**

|  |  |  |  |
| --- | --- | --- | --- |
| **Word** | **Word Count** | **Word** | **Word Count** |
| phone | 174348 | good | 57856 |
| case | 144660 | battery | 57136 |
| n't | 116215 | well | 49473 |
| 's | 97796 | iphone | 47733 |
| one | 85413 | get | 46324 |
| like | 71797 | charge | 44391 |
| great | 65972 | charger | 38170 |
| use | 60771 | really | 37971 |
| screen | 59489 | product | 37684 |
| would | 58738 | also | 36168 |

Given that the dataset are reviews on cell phones and accessories, it is to be expected that most words relate closely to these. Examples such words are phone, case, screen, battery, iPhone, charge and charger. In reviews, users tend to express their feelings towards the product, thus using expressive words such as like, great, good and well. Another form of expression will be through utterances such as “really great product!” or “I would also like to get one more if possible!”. Clitics are a commonly used in writing, thus it is not unusual for them to be in the top-20 words.

**Table 8. Top-20 words before stemming**

|  |  |  |  |
| --- | --- | --- | --- |
| **Word** | **Word Count** | **Word** | **Word Count** |
| phone | 189572 | batteri | 65089 |
| case | 163289 | get | 61106 |
| use | 116698 | screen | 61075 |
| n't | 116215 | would | 58738 |
| 's | 97796 | good | 58097 |
| charg | 91199 | look | 51814 |
| one | 90936 | fit | 49923 |
| like | 79632 | iphon | 49905 |
| work | 75522 | well | 49485 |
| great | 66375 | time | 46973 |

The top-20 frequent words show in Table 8 after stemming seems to get the same type of words as the ones in Table 7, before stemming with a slight difference in the order. For example charge and charger both appear in the top-20 words before stemming. However after stemming both will be stemmed to the same root, thus it becomes the top few frequent words.

The only word that does not relate to cell phones and accessories in Table 8 is ‘time’, although cell phone has time on them, it is not a feature that people would explicitly spell out given that it is a given feature in almost all cell phones. A possible case as to why ‘time’ is frequently used, might be a reference to the delivery time. Users might also include the delivery time of the product on the review, instead of just the product itself.

**Table 9. POS tagging results**

|  |
| --- |
| [(u'these', 'DT'), (u'stickers', 'NNS'), (u'work', 'VBP'), (u'like', 'IN'), (u'the', 'DT'), (u'review', 'NN'), (u'says', 'VBZ'), (u'they', 'PRP'), (u'do', 'VBP'), (u'.', '.'), (u'they', 'PRP'), (u'stick', 'VBP'), (u'on', 'IN'), (u'great', 'JJ  '), (u'and', 'CC'), (u'they', 'PRP'), (u'stay', 'VBP'), (u'on', 'IN'), (u'the', 'DT'), (u'phone', 'NN'), (u'.', '.'), (u'they', 'PRP'), (u'are', 'VBP'), (u'super', 'JJ'), (u'stylish', 'JJ'), (u'and', 'CC'), (u'i', 'NN'), (u'can', 'MD'), (u'share', 'NN'), (u'them', 'PRP'), (u'with', 'IN'), (u'my', 'PRP$'), (u'sister', 'NN'), (u'.', '.'), (u':', ':'), (u')', ')')] |
| [(u'great', 'JJ'), (u'charger', 'NN'), (u'for', 'IN'), (u'2', 'CD'), (u'devices', 'NNS'), (u'and', 'CC'), (u'i', 'NNS'), (u'have', 'VBP'), (u"n't", 'RB'), (u'had', 'VBD'), (u'any', 'DT'), (u'problems', 'NNS'), (u'so', 'RB'), (u'far', 'RB'), (  u'3yrs', 'CD'), (u'now', 'RB')] |
| [(u'performs', 'NNS'), (u'exactly', 'RB'), (u'as', 'IN'), (u'advertised', 'VBN'), (u'.', '.'), (u'it', 'PRP'), (u"'s", 'VBZ'), (u'very', 'RB'), (u'sturdily', 'RB'), (u'built', 'VBN'), (u',', ','), (u'and', 'CC'), (u'provides', 'VBZ'), (u'lots', 'NNS'), (u'of', 'IN'), (u'boost', 'NN'), (u'.', '.'), (u'it', 'PRP'), (u'does', 'VBZ'), (u'exactly', 'RB'), (u'what', 'WP'), (u'it', 'PRP'), (u"'s", 'VBZ'), (u'supposed', 'VBN'), (u'too', 'RB'), (u'.easy', 'JJ'), (u'to', 'TO'), (u'insert', 'VB'), (u'phone',  'NN'), (u'in', 'IN'), (u'and', 'CC'), (u'out', 'IN'), (u'.', '.'), (u'definitely', 'RB'), (u'a', 'DT'), (u'5', 'CD'), (u'star', 'NN'), (u'experience', 'NN'), (u'.', '.'), (u'do', 'VBP'), (u"n't", 'RB'), (u'know', 'VB'), (u'what', 'WP'), (u'i', 'NN'), (u'would',  'MD'), (u'do', 'VB'), (u'without', 'IN'), (u'this', 'DT'), (u'case', 'NN'), (u'love', 'VB'), (u'love', 'NN'), (u'love', 'IN'), (u'it', 'PRP'), (u'.', '.')] |
| [(u'the', 'DT'), (u'samsung', 'NN'), (u'car', 'NN'), (u'charger', 'NN'), (u'has', 'VBZ'), (u'stopped', 'VBN'), (u'working', 'VBG'), (u'within', 'IN'), (u'a', 'DT'), (u'6', 'CD'), (u'month', 'NN'), (u'period', 'NN'), (u'.', '.'), (u'i', 'VB'), (u'thought', 'VBD'), (u'a', 'DT'), (u'fuse', 'NN'), (u'was', 'VBD'), (u'blown', 'VBN'), (u'inside', 'IN'), (u'the', 'DT'), (u'unit', 'NN'), (u'but', 'CC'), (u'when', 'WRB'), (u'i', 'NN'), (u'attempted', 'VBD'), (u'to', 'TO'), (u'change', 'VB'), (u'it', 'PRP'), (u',', ','), (u'i', 'VBZ'), (u"'ve", 'VBP'), (u'noticed', 'VBN'), (u'that', 'IN'), (u'it', 'PRP'), (u'does', 'VBZ'), (u"n't", 'RB'), (u'come', 'VB'), (u'apart', 'RB'), (u'like', 'IN'), (u'most', 'JJS'), (u'vehicle', 'NN'), (u'chargers', 'NNS'), (u'.', '.')] |
| [(u'I', 'PRP'), (u'bought', 'VBD'), (u'this', 'DT'), (u'a', 'DT'), (u'little', 'JJ'), (u'skeptical', 'JJ'), (u'.', '.'), (u'After', 'IN'), (u'I', 'PRP'), (u'tried', 'VBD'), (u'it', 'PRP'), (u'I', 'PRP'), (u'bought', 'VBD'), (u'two', 'CD'), (u  'more', 'JJR'), (u'.', '.'), (u'It', 'PRP'), (u'works', 'VBZ'), (u'great', 'JJ'), (u'and', 'CC'), (u'so', 'RB'), (u'far', 'RB'), (u'it', 'PRP'), (u'has', 'VBZ'), (u'lasted', 'VBN'), (u'for', 'IN'), (u'about', 'RB'), (u'3', 'CD'), (u'months', 'NNS'), (u'.', '.')  , (u'If', 'IN'), (u'that', 'DT'), (u'changes', 'VBZ'), (u'I', 'PRP'), (u'will', 'MD'), (u'update', 'VB'), (u'this', 'DT'), (u'review', 'NN'), (u'.', '.')] |

Based on the POS tagging results shown in Table 9, the POS tagging is quite accurate. The most common error is that ‘i’ has being tagged with ‘NN’ instead of ‘PRP’. However, this is due to the token being transform to lowercase before tagging. This can be seen from the last example give, where the tokens are not transformed to lowercase. The token ‘I’ appears two times and both times the correct tag is given.

This shows that NLTK pos tagging takes into account the letter case of the tokens. If tokens are transformed to lower case before tagging, it might affect tokens such as Proper Nouns and Personal Pronoun to be tagged incorrectly.

Another issue would be the POS tagging is done on a word level instead of sentence level. As tokenization is required to be done before POS tagging for NLTK pos tagger, the semantic of the sentence will be lost.

# **3. Development of a Noun Phrase Summarizer**

**3.1. The top-20 most frequent noun phrases**

In this section a noun phrase detector is designed and implemented. It will analyze text taken from both "reviewText" and "summary" to extract noun phrases.

The regular expression used to split a string and get tokens is shown as below.

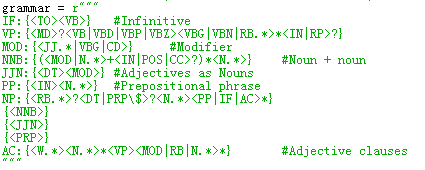
**Figure 9: Regular expression used to get tokens**

C:\Users\user\AppData\Roaming\Tencent\Users\380511977\QQ\WinTemp\RichOle\GQNSACTDSG4{{VAYYOYHXDV.png

As a noun phrase (NP) can contain:

* Exactly one determiner at the beginning of the NP
* An arbitrary number of adjectives before the noun
* An arbitrary number of preposition phrases (PP) after the noun
* One or more sentences at the end of the NP
* A personal pronoun

Figure 10 shows the grammar used for noun phrase chunking.



**Figure 10. Chunk grammar used by the detector**

Table 10 gives some examples that the grammar can match.

**Table 10. Example phrases**

|  |
| --- |
| the/DT great/JJ wall/NN  bank/NN of/IN china/NN  the/DT developing/VBG country/NN  a/DT 5-year-old/JJ boy/NN  the/DT police/NN officer/NN 's/POS dog/NN  (JJN the/DT old/JJ)  the/DT girl/NN in/IN the/DT front/JJ seat/NN  the/DT book/NN (IF to/TO read/VB)  the/DT child/NN (AC that/WDT looks/VBZ lost/VBN) |

Though pronouns matches in this grammar, as they do not provide any useful information for this assignment, in later statistics, they will be removed from the dictionary of all noun phrases, together with other individually-appearing Engligh stopwords. While they would still be used for computing the precision and recall value.

The top-20 most frequent noun phrases used by reviewers are listed below.

**Table 11: The top-20 most frequent noun phrases**

|  |  |  |
| --- | --- | --- |
| **Rank** | **Noun phrase** | **Frequency** |
| 1 | the phone | 0.833% |
| 2 | this case | 0.818% |
| 3 | the case | 0.667% |
| 4 | my phone | 0.461% |
| 5 | works | 0.353% |
| 6 | this product | 0.305% |
| 7 | this one | 0.303% |
| 8 | the screen | 0.205% |
| 9 | your phone | 0.198% |
| 10 | a little | 0.196% |
| 11 | the battery | 0.189% |
| 12 | a bit | 0.182% |
| 13 | the price | 0.172% |
| 14 | great product | 0.153% |
| 15 | this phone | 0.153% |
| 16 | great case | 0.144% |
| 17 | something | 0.133% |
| 18 | work | 0.132% |
| 19 | pros | 0.130% |
| 20 | my iphone | 0.127% |

The result shows that the most frequent noun phrase is “the phone”, certainly. The noun phrase with “case” also shows its popularity. Those frequent mentioned noun phrases indicate aspects user concerned, such as “the screen”,“the battery” and “the price”.

**3.2 Representative noun phrase**

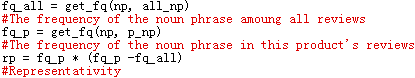
According to Table 1: Top 10 Products table in Section 2.1, the top 3 popular products with the largest number of reviews have been chosen as follows:

**Table 12: The asin for top 3 popular products**

|  |  |
| --- | --- |
| **Rank** | **asin** |
| 1 | B005SUHPO6 |
| 2 | B0042FV2SI |
| 3 | B008OHNZI0 |

A "representative noun phrase" for a product’s reviews is considered as a noun phrase that frequently appearing in this product’s reviews but not in others.

The formula is implemented to help measure the representivity of a noun phrase as below. The result does not have meaning, but it would be used for comparing and ranking.



**Figure 11: Computing of the representativity**

The results are shown in following three tables.

**Table 13: Representative noun phrase for Product B005SUHPO6**

|  |  |
| --- | --- |
| **Rank** | **Noun phrase** |
| 1 | this case |
| 2 | otterbox |
| 3 | the case |
| 4 | the phone |
| 5 | my phone |
| 6 | great case |
| 7 | your phone |
| 8 | great product |
| 9 | this product |
| 10 | the rubber |

**Table 14: Representative noun phrase for Product B0042FV2SI**

|  |  |
| --- | --- |
| **Rank** | **Noun phrase** |
| 1 | the screen |
| 2 | my phone |
| 3 | works |
| 4 | this product |
| 5 | these screen protectors |
| 6 | bubbles |
| 7 | this screen protector |
| 8 | perfect |
| 9 | the matte finish |
| 10 | the screen protector |

**Table 15: Representative noun phrase for Product B008OHNZI0**

|  |  |
| --- | --- |
| **Rank** | **Noun phrase** |
| 1 | the screen |
| 2 | the screen protector |
| 3 | tech armor |
| 4 | no bubbles |
| 5 | this product |
| 6 | great product |
| 7 | the protector |
| 8 | this screen protector |
| 9 | these screen protectors |
| 10 | the instructions |

**3.3 Precision and Recall**

Following are five sample reviews taken for evaluating precision and recall. The underline parts are noun phrase.

**Table 16: Sample reviews for evaluation**

|  |  |  |
| --- | --- | --- |
| **No** | **Sample review with manually annotated noun phrases** | **Detected by the noun phrase detector** |
| 1 | "It is what it is. A charging and sync cord. It charges the phone and helps transfer data from the phone to the computer." | "It is what it is. A charging and sync cord. It charges the phone and helps transfer data from the phone to the computer." |
| 2 | "I have not used this as of yet , however, my wife has without issue.I have the x50 headset and she has the voyage, it works on both really well." | "I have not used this as of yet , however, my wife has without issue.I have the x50 headset and she has the voyage, it works on both really well." |
| 3 | "It works great. It has a longer cord than the one I got with the phone. I would recommend this item." | "It works great. It has a longer cord than the one I got with the phone. I would recommend this item." |
| 4 | "just what you need, I am always having to charge my phone and then find I have another item to charge also." | "just what you need, I am always having to charge my phone and then find I have another item to charge also." |
| 5 | "I rate this 5 stars. Love this product! great price and quality. Love how it works with my iphone 4. And it comes with a lot of tools which is great, in case you misplace one of them." | "I rate this 5 stars. Love this product! great price and quality. Love how it works with my iphone 4. And it comes with a lot of tools which is great, in case you misplace one of them." |

**Table 17: Comparing results for sample reviews**

|  |  |  |
| --- | --- | --- |
| True positive | False positive | False negative |
| 30 | 6 | 4 |

Precision = tp÷(tp+fp) = 83.333%

Recall = tp ÷ (tp+fn) = 88.235%

By comparing above sample reviews annotation, it is found that the false positives usually come out when the detector fails matching the noun phrase when a participle phrase is performing as the modifier. While the false negatives come out when the word is given a wrong tag.

# **4. Sentiment Word Detection**

This section will cover all the tasks given within the Sentiment Word Detection section of the assignment. Each line in the dataset corresponds to a JSON object. For performance reasons, the entire dataset will be concatenated as one large JSON array string before being deserialized by the JSON decoder.

The programming language used for this assignment is Python and the Natural Language Processing library of choice is NLTK.

## **4.1 Methods**

Sentiment word detection is done by obtaining the argmax of the following evaluation function.

represents the evaluation function for a specific word x given emotion q.

P(x|q) represents the probability of word x occurring given emotion q.

x represents a word.

q represents an emotion (Positive or Negative).

The evaluation function can be understood as the weighted sum of two different features, where the weight for feature P(x|q) is 1 and weight for feature P(x|~q) is -1

.

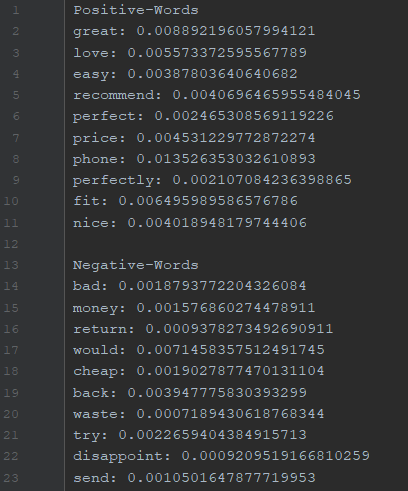
The intuition for the +1 weight for P(x|q) is such that we expect to find a word that occurs most frequently for a given emotion that we are interested in.

The intuition for the -1 weight for P(x|~q) is such that we expect to find a word that occurs less frequently for a given emotion that is in direct contradiction with the current emotion that we are interested in. This weight appears to be an important regularizer for us to eliminate words that occurs in relatively high frequency in both P(x|q) and P(x|~q).

In our assignment, we obtain P(x|q) through the following process:

1. Initialize score table with initial score of 5 to form a laplace smoothed estimate.
2. Tokenize each sentence in the dataset into words
3. Convert the words into lower case
4. Perform part of speech tagging for each word
5. Remove words with tag that is not listed in the tag whitelist (UH, VB\*, JJ\*, RB\*, RP)
6. Remove words that are listed in the STOPWORDS corpus in NLTK
7. Extract the lemma of the word by using WordNet lemmatizer
8. Remove multiple occurrences of the same lemma, x, in a sentence to reduce bias towards any particular sentence with high frequency of a specific lemma
9. Calculate the positive-score, , and negative-score, r + 1. Where r represents the overall rating for the sentence.
10. Add the positive-score to SCOREp(x) and SCOREp(NONE). Add the negative-score to SCOREn(x) and SCOREn(NONE).
11. Calculate the P(x|q) using the following formula

## **4.2 Results & Analysis**



**Figure 9. Showing top 10 positive & negative words and their corresponding P(X|Q) values.**

Figure 9 shows the results of the top 10 positive and negative words by applying our algorithm. Words like great, love and easy expresses strongly positive opinions while words like bad, money and return expresses strongly negative opinions. It is worthy to note that certain phrases that occurs in high frequencies for a certain emotion are shown from the results, for example “return back” and “send back”.

# **5. Application**

This section covers a simple NLP application based on the dataset. This simple tool aims at finding out the comments that are controversial or conflict with overall score. As the overall score is the most intuitive way that a potential buyer to know the product.

This case can happen in the following conditions:

1. The comment is updated after the reviewer left a comment, however the overall score is unchanged

For example,

1. The comments contains different aspects of the product. However, reviewer give the overall score

based on personal so much. For example, the product is overall pretty good, however some design makes the reviewer give very low marks.

Therefore the score can’t represent the real condition of the product.

1. The reviewer makes a mistake while giving the overall score.(This case is not common in this database as it’s already cleaned and selected. It is very meaningful especially when applied on raw data.

## **5. 1 Methods**

The nltk package is applied to do segmentation and sentimental analyse. SentimentIntensityAnalyzer() function

from nltk.sentiment.vader is applied to do sentimentale analyse on sentence from review text and summary.

SentimentIntensityAnalyzer() use n diagram model and can evaluate sentiment expressed by the sentence. polarity\_scores() methods can return a value range from (-1,1). The negative value means negative feeling and positive value means positive feeling. Therefore we can evaluate reviews’ comments on their review and summary.

Besides, summary and review text are given different weight contribution and mapped to a range from 1 to 5 to compare with the overall score.

## **5. 2 Results & Analysis**

There are overall 140 controversial reviews found by the application among all the 200000 reviews.

The following is a simple controversial review that is found out by the application

# **6. Conclusion**

In this assignment, we have analysed reviews from amazon regarding cell phone and cell phone related accessories.

The first step is to analyse the dataset to understand it and identify limitations of the tools used. Next, we design a noun phrase detector that will identify noun phrases in this dataset. Then we again analyse the dataset, but this time we try to get the sentiments of the reviews and identify words that are commonly associated with positive and negative emotions.

Lastly, we came up with an application where it would detect reviews that does not match the rating given. We experimented with different weight values in order to get the best result.

## **References**

[1] Pandas Community. *Python Data Analysis Library*. Available from <http://pandas.pydata.org/>.

[2] Team NLTK. *NLTK 3.3.* Available from <https://www.nltk.org/>

[3] Matplotlib Community. *Matplotlib 2.2.3.* Available from <https://matplotlib.org/2.2.3/index.html>

[4] NumPy Community. *NumPy 1.15.2*. Available from <http://www.numpy.org/>

# **Appendix**

List of stop words used (NLTK stop words library and punctuations)

|  |
| --- |
| ';', 'too', 're', "mustn't", 'haven', 'does', "shan't", "she's", "should've", 'am', 'having', 'mustn', 'they', 'other', 'at', 'only', 'it', "wouldn't", 'll', 'very’, below', 'after', 'and', '"', 'is', 'under', "weren't", 'him', 'here', "you'll", before', 'wouldn', 'further', '\\', 'them', 'while', "wasn't", '&', 'just', 'about', 'did', 'won', 'out', 'herself', 'hers', "aren't", 'weren', "won't", 'of', 'which', ourselves', ':', 'if', 'who', 'each', ')', 'the', 'should', 'have', 'itself', '>', 'not', 'because', 'hasn', 'in', '<', 'most', '{', 'than', 't', theirs', 'on', 'again', "you'd", '/', '^', 'he', 'were', 've', 'myself', '+', 'i', 'our', 'wasn', 'once', 'was', 'with', 'from', 'whom', 'me', ma', 'yourself', 'same', '-', 'both', 'so', 'are', "you're", 'then', 'yourselves' |