**Attn: Dr. Sun Aixin**

A picture containing text

Description automatically generated

**CE/CZ4045 Natural Language Processing**

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below. We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work. We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work.

|  |  |
| --- | --- |
| Name | Signature / Date |
| Chulpaibul Jiraporn | / 1 November 2020 |
| Leong Ko Rixie Tiffany | A picture containing shape  Description automatically generated / 1 November 2020 |
| Leong Kai Ling | /1 November 2020 |
| Liew Zhi Li | A picture containing text  Description automatically generated /1 November 2020 |

Important note:

Name must **EXACTLY MATCH** the one printed on your Matriculation Card. Any mismatch leads to **THREE (3)** marks deduction.

Domain Specific Text Data Analysis and Processing

Chulpaibul Jiraporn, Leong Ko Rixie Tiffany, Leong Kai Ling, Liew Zhi Li

School of Computer Science and Engineering

Nanyang Technological University

Singapore

{CHUL0004, RLEONG007, KLEONG020, LIEW0177}@e.ntu.edu.sg

ABSTRACT

In this report, we will summarize the findings and analysis from our hands-on experience using different NLP tools and techniques to solve NLP tasks. We will also present the effectiveness and challenges faced by each component and the solutions. The report consists of three parts: (i) Dataset Analysis, (ii) Development of Summarizer, (iii) Sentiment Analysis Application.

KEYWORDS

NLP, Tokenization, Stemming, Sentence Segmentation, POS tagging, Noun-Adjective Pair, Sentimental Analysis

1 Domain Specific Dataset Analysis

In this section, three domain specific datasets are selected for analysis as follows:

1. Stack Overflow question and answers about ReactJS
2. Research papers about peer tutoring
3. Patents for drugs-related inventions

Each dataset contains 20 documents.

1.1 Document Retrieval and Preprocessing

We extracted the data for the three domain specific datasets. Then, regular expressions were used to clean the text before analysis.

*1.1.1 Stack Overflow.* The Stack Exchange API was used to obtain the links to the most frequently asked questions about ReactJS. Beautiful Soup was used to extract the text from the HTML. One document refers to one question and its answers.

*1.1.2 Research Papers.* The research papers about peer tutoring were obtained from NTU OneSearch. Pdfminer was used to extract text from the PDF documents. One document refers to one research paper.

*1.1.3 Patents.* All drug-related patents are selected from <https://patents.google.com/>. Pdfminer was used to extract text from the PDF documents. One document refers to one patent.

1.2 Tokenization and Stemming

*1.2.1 Tokenization.*

nltk.word\_tokenize() was used to perform tokenization (Figure 1). We converted the tokens into lower case so that the tokens are case insensitive. After obtaining the tokens, we used python sets to obtain the unique tokens.

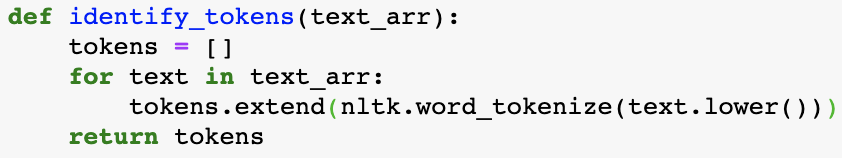


Figure 1: Code snippet of tokenization function.

*1.2.1.1 Stack Overflow.* Stack Overflow is a question-and-answer site for programmers. Hence, its documents often contain a lot of source code. Source code involves the use of many punctuation marks. Below is a list of some the correctly and incorrectly recognized tokens returned by the tokenizer.

Correctly recognized tokens:

‘react-router’, ‘redux’, ‘useeffect’, ‘❌’, ‘const’, ‘function’, ‘state’

Incorrectly recognized tokens:

|  |  |  |
| --- | --- | --- |
| **Input** | **Expected tokens** | **Tokenizer output** |
| <p>hello</p> | ‘<p’, ‘>’, ‘hello’, ‘</’, ‘p’, ‘>’ | ‘<’, ‘p’, ‘>’, ‘hello’, ‘<’, ‘/p’, ‘>’ |
| console.log(“data fetch error”) | ‘console’, ‘.’, ‘log’, ‘(’, ‘“data fetch error”’, ‘)’ | ‘console.log’, ‘(‘, ‘``’, ‘data’, ‘fetch’, ‘error’, ‘''’, ‘)’ |
| var React= require('react'); | ‘var’, ‘React’, ‘=’, ‘require’, ‘(’, ‘‘react’’, ‘)’, ‘;’ | ‘var’, ‘React=’, ‘require’, ‘(‘, ‘'react’, ‘'’, ‘)’, ‘;’ |

From the examples above, we can observe that the tokenizer was able to identify space separated domain specific tokens easily. However, the tokenizer often fails to correctly delimit lines of code that involve punctuation marks such as less than (<), equals to (=) and opening single quote (‘). Since the tokenizer is unable to distinguish between plaintext and source code, the tokenizer treats all text as plaintext. This results in source code often being incorrectly tokenized.

For example, the tokenizer fails to recognize that strings are delimited by double quotes (“”) or single quotes (‘’), and that a string should be treated as a single token.

The tokenizer performed slightly better with properly formatted code. For example, for the input “const React = require('react');”, the tokenizer was able to correctly recognize “React” and “=” as separate tokens, but it was not able to for the input “var React= require('react');” (refer to the third example for incorrectly recognized tokens).

To identify incorrect tokens through programs, we could use regular expressions to correctly identify tokens with punctuation marks such as strings and HTML tags. This will allow the tokenizer to correctly handle the punctuation marks.

To improve the tokenizer, we could use a separate tokenizer for plaintext and source code. The current tokenizer (nltk.word\_tokenize()) can be used to tokenize the plaintext. Another tokenizer specifically designed to tokenize source code can be used to tokenize the source code. This tokenizer can resemble the tokenizer used in compilers to tokenize code. A binary classifier could be used to determine whether a sentence is plaintext or source code, so the correct tokenizer can be used.

*1.2.1.2 Research Papers.* The research papers have a similar structure in terms of how the components included in the paper, such as citations, appendices and abstracts. A list of some the correctly and incorrectly recognized tokens returned by the tokenizer are shown below.

Correctly recognized tokens:

‘tutoring’, ‘i.e.’, ‘methodology’, ‘collaborative’, ‘learning’, ‘assessment’, ‘e.g.’

Incorrectly recognized tokens:

|  |  |
| --- | --- |
| **Expected token** | **Tokenizer output** |
| teriston@yamaguchi-u.ac.jp | ‘teriston’, ‘@’, ‘yamaguchi-u.ac.jp’ |
| https://doi.org/10.1016/j.nedt.2017.12.001 | ‘https’, ‘:’, ‘//doi.org/10.1016/j.nedt.2017.12.001’ |
| et al. | ‘et’, ‘al.’ |

In general, most of the tokens are space separated. So, the tokenizer was able to correctly recognize most of the tokens. It was also able to correctly recognize some tokens with periods (.), such as “i.e.” and “e.g.”.

However, the tokenizer was unable to handle other punctuation marks correctly, such as at sign (@) and colon (:). This resulted in the tokenizer incorrectly breaking up the tokens for email addresses and URLs. The tokenizer was also unable to handle research specific tokens such as “et al.”.

To identify incorrect tokens through programs, domain-specific tokens can be added to the dictionary of the tokenizer, such as “et al.”.

To improve the tokenizer, regular expressions may be used to help the tokenizer handle email addresses and URLs correctly.

*1.2.1.3 Patents.* In general, patents have similar formats such as patent number, dates, and figures. Moreover, all the patents selected were related to drugs, so chemical names frequently appeared. Some of the resulting tokens returned by the tokenizer are as shown:

Correctly recognized tokens:

‘carboxymethyl’, ‘polydimethylsiloxane’, ‘ethylene-co-vinyl', ‘polycarbonate-urethan’, ‘2008/0081064’, ‘1.00’, ‘7,8-didehydro-4,5-epoxy-17-methylmorphian-3,6-diol'

Incorrectly recognized tokens:

|  |  |
| --- | --- |
| **Expected token** | **Tokenizer output** |
| 6-oxo-3-((4-(pyridin-2-ylsulfamoyl)phenyl)hydrazinylidence cyclo-hexa-1,4-diene-1-carboxylic | '6-oxo-3-', '(', '(', '4-', '(', 'pyridin-2-ylsulfamoyl', ')', 'phenyl', ')', 'hydrazinylidence', 'cyclo-hexa-1,4-diene-1-carboxylic' |
| 2-hydroxyethyl methacrylate | '2-hydroxyethyl', 'methacrylate' |
| 95% polylactide | '95', '%', 'polylactide' |
| HPMC - E3 | 'hpmc', '-', 'E3' |
| FIG. | 'fig', '.' |

From the examples above, we can observe that the tokenizer generally does not use hyphen (-), comma (,) and forward slash (/) as delimiters. Thus, chemical compound and tokens with these punctuations are identified accurately regardless of the length of the token. On the other hand, the tokenizer uses parentheses, whitespace and percentage symbol % as delimiters. As such, tokens containing such symbols are mistaken by the tokenizer and are further decomposed into smaller tokens. However, in the case of a period (.), the tokenizer will depend on the symbol after the period. If the period is followed by a whitespace, then the period will be regarded as a single token. Otherwise, the period will be part of a token along with the neighboring non-whitespace symbol.

To identify incorrect tokens through programs, we could validate the token with words in the domain specific dictionary. The program will take the domain specific dictionary and token from tokenizer as inputs and check if the token from the tokenizer match any word in the dictionary. If a match is found, then the token is correct. Otherwise, the token is incorrect.

To improve the tokenizer, we could integrate maximum matching word segmentation algorithm into the tokenizer. When the tokenizer identifies a token, we run the maximum matching word segmentation algorithm from the first character of the token and return the maximum matching token. The tokenizer then identifies the next token from the character after the previous maximum matching token.

*1.2.2 Stemming.*

nltk.stem.PorterStemmer() was used to perform stemming on the previously identified tokens (Figure 2). After obtaining the stemmed tokens, we used python sets to obtain the unique tokens.

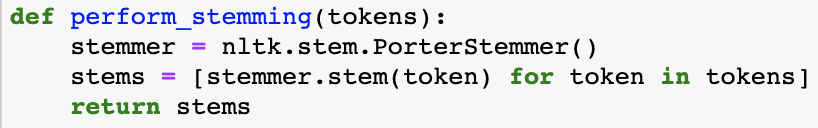


Figure 2: Code snippet of stemming function.

*1.2.2.1 Comparison Between the Number of Distinct Tokens Before and After Stemming.* Figure 3 shows a comparison between the number of distinct tokens before and after stemming.

Chart, bar chart

Description automatically generated

Figure 3: Bar plot representing the number of tokens before (in blue) and after stemming (in orange).

For all 3 domains, the number of distinct tokens before stemming is always higher than the number of distinct tokens after stemming. This is because different tokens may share the same stems. For example, the tokens ‘change’, ‘changes’, ‘changing’, ‘changed’ have the same stem — ‘change’. Below is a table of the percentage decrease in number of distinct tokens after stemming.

|  |  |
| --- | --- |
| **Domain** | **% Decrease After Stemming** |
| Stack Overflow | 24.46% |
| Research Papers | 20.58% |
| Patents | 15.24% |

From the table above, we can see that stack overflow had the largest percentage decrease in the number of distinct tokens and patents had the smallest percentage decrease in the number of distinct tokens.

*1.2.2.1 Comparison of the Length Distribution of Tokens Before and After Stemming.* Figure 4 shows a comparison of the length distribution of the tokens before and after stemming for each domain.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

A picture containing application

Description automatically generated

Figure 4: Histograms showing the length distribution of the tokens in terms of number of characters for each domain.

Stemming reduces a word to its word stem by removing affixes. Therefore, when we apply stemming to a token of length L, the length of resulting stem will be no longer than length L — the maximum length of the stem is L. In the probability distributions of the 3 domains above, the probability densities of stems are higher than that of tokens when the length of the token is short and vice versa. It can also be observed that majority of tokens and stems are shorter than 20 characters. The longer tokens are due to URLs and chemical names.

1.3 Sentence Segmentation

nltk.sent\_tokenizer() was used to perform sentence segmentation (Figure 5). The number of tokens in each sentence was counted, so that the distribution of sentence length could be compared. We used multiple types of visualizations to analyze difference aspects of the distributions.

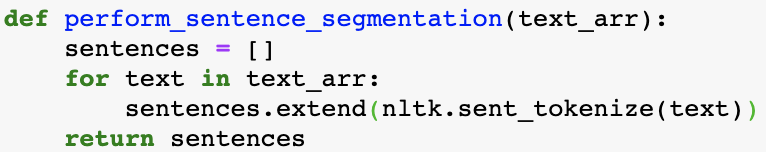


Figure 5: Code snippet of sentence segmentation function.

Chart

Description automatically generated

Figure 6: Scatterplot of number of sentences against sentence length in terms of number of tokens for each domain.

From the scatterplot Figure 6, we can see that most of the sentences contain less than 100 tokens. However, there are some sentences with more than 400 tokens. Most of these longer sentences are not segmented properly by the tokenizer. For example, research papers and patents have tables to show findings and people usually post source code and code error messages on Stack Overflow. The sentence tokenizer considers such tables and source code as one sentence. Moreover, there is an outlier where more than 5000 sentences in patents have 2 tokens per sentence. This is because each patent contains indexes and multiple country codes, such as EP which stands for European Patent. To study the length distribution of sentences across 3 domains in general, we took the logarithm of the number of tokens to reduce the effect of outliers.

Chart, box and whisker chart

Description automatically generated

Figure 7: Boxplot of the logarithmic distribution of the sentence length in terms of number of tokens for each domain.

From the boxplot in Figure 7, we can see that, by median, sentences are longer (in terms of number of tokens) in the Stack Overflow forum followed by research papers and patents. However, the range of sentence length varies more in patent followed by research and Stack Overflow forum.

A picture containing graphical user interface

Description automatically generated

Figure 8: Histogram of the distribution of sentence length in terms of number of tokens for each domain.

From the histogram in Figure 8, we can see that the patents have a very high percentage of sentences with very small sentence length due to 2 tokens per sentences issue mentioned previously. We can also see that the Stack Overflow sentences have a more even distribution of sentence lengths since its curve is flatter. This is likely due to the high amount of source code of varying lengths included in the Stack Overflow sentences.

1.4 POS (Part-of-speech) Tagging

POS tagging was applied to 3 randomly selected sentences from each dataset. nltk.pos\_tag() was used to perform the POS tagging (Figure 9). The POS tagging results can be found in Appendix A.

Text

Description automatically generated

Figure 9: Code snippet of POS tagging function.

*1.4.1 Stack Overflow.* For the Stack Overflow sentences, the POS tagger was mostly accurate, despite some of the sentences being ungrammatical. The POS tagger was able to correctly identify “React” as a singular proper noun (NNP) in the first sentence, but not in the third sentence.

Due to some misspellings in the second sentence, some of the tokens were tagged incorrectly. For example, the author incorrectly wrote “appending and extra .$” instead of “appending an extra .$”. This caused the POS tagger to tag “appending” as a noun (NN) instead of as a verb (VBG). Additionally, the author used the wrong case for the word “Element”, which should be “element”. This caused the POS tagger to tag “element” as a proper noun (NNP) instead of as a singular noun (NN).

In the second sentence, the POS tagger treated source code as proper nouns. For example, “React.Children.map/NNP” and “React.cloneElement/NNP”. This tagging is appropriate given the POS tagger’s limited tag set.

*1.4.2 Research Papers.* The POS tagger was able to accurately tag the sentences from the research papers. This is likely because research papers use proper grammar, making them easier to tag. The POS tagger was even able to tag unknown tokens based on the context, such as “2-h/JJ” which means 2 hours.

However, there were still some incorrectly tagged tokens, possibly due to ambiguity:

|  |  |  |
| --- | --- | --- |
| **Token** | **Tag by POS tagger** | **Expected Tag** |
| subject (sentence 1) | JJ | NN |
| vocabulary (sentence 2) | JJ | NN |

Additionally, due to the word “kindergarteners” being separated by a hyphen, the POS tagger incorrectly treated the word as 2 words (“kinder-/JJ garteners/NNS”).

*1.4.3 Patents.* From the POS tagging of the patent, we can see that the POS tagger is mostly accurate as most tokens are tagged with the correct POS. Two tokens are tagged incorrectly in sentence 2:

|  |  |  |
| --- | --- | --- |
| **Token** | **Tag by POS tagger** | **Expected Tag** |
| Approximate | NNP | JJ |
| oxyntomodulin | MD | NNP |

Overall, the POS tagger can handle the domain specific terms well as most of the domain specific term has only one POS tag.

2 Development of Noun-Adjective Pair Ranker

The dataset used for the noun-adjective pair ranker program consists of 30 reviews for the product ‘Samyang Spicy Chicken Roasted Noodles’. Each review has a review text and rating and they are carefully selected and manually added into a Comma Separated Values (CSV) file. The following are the most meaningful top 5 pairs of noun-adjective identified manually:

1. (noodle, spicy)
2. (noodle, instant)
3. (sauce, spicy)
4. (flavor, good)
5. (chicken, spicy)

A Python Jupyter Notebook is used to develop the noun-adjective pair ranker program. We have utilized the popular open-source natural language processing library called spaCy to perform POS tagging and dependency parsing. Pandas is another library that we have used for the manipulation and analysis of data.

2.1 Approaches to Find Noun-Adjective Pair

In order to identify the noun-adjective pair in each review text (“document”), there are three possible approaches.

Approach 1: We enumerate through each word in the document using a loop, identify a word that is an adjective (“ADJ”). If the word is an adjective, dependency parsing is used to traverse up the dependency tree of the adjective to find a noun (“NOUN” or “PROPN”). The adjective is paired with the nearest noun it finds, added to the data frame and breaks out of the loop. We repeat this process for the next adjective.

Approach 2: We perform the same procedure as the first approach. However, instead of finding only the nearest noun and breaking out of the loop, we continue finding as many nouns as possible to form multiple noun-adjective pairs for the same adjective.

Approach 3: We enumerate through each word in the document using a loop, identify a word that is a noun (“NOUN” or “PROPN”). If the word is a noun, the dependency parsing is used to traverse up the dependency tree of the noun to find an adjective (“ADJ”) by finding the syntactic dependency label (“amod”).

We will use the following review text as an example to illustrate the accuracy of all three approaches.

|  |
| --- |
| If you're looking for a real food challenge these noodles will give you a run for your money. They can be found at any asian supermarket and are very inexpensive. You can buy them as single pack, noodle bowl, or a large pack. They also come in different flavors such as chicken, carbonara, and more. Extremely spicy but extremely delicious. |

The following table is the results of the noun-adjective pairs identified using approach 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Noun** | **Noun Base** | **Adjective** | **Adjective Base** |
| challenge | challenge | real | real |
| supermarket | supermarket | asian | asian |
| pack | pack | single | single |
| flavors | flavor | different | different |
| flavors | flavor | such | such |
| carbonara | carbonara | more | more |

Table 1: Results of noun-adjective pairs using Approach 1

Using approach 1, only a single pair is found for an adjective. For example, the word “real” is identified as an adjective which is identified in a single pair (challenge, real). We noticed that there are two noun-adjective pairs that are incorrectly identified, which are (flavor, such) and (carbonara, more). These two pairs are incorrect due to the context of the sentence. Thus, the accuracy of the first approach = 5/7 = 0.714.

The following table is the results of the noun-adjective pairs identified using approach 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Noun** | **Noun Base** | **Adjective** | **Adjective Base** |
| challenge | challenge | real | real |
| supermarket | supermarket | asian | asian |
| pack | pack | single | single |
| pack | pack | large | large |
| bowl | bowl | large | large |
| pack | pack | large | large |
| flavors | flavor | different | different |
| flavors | flavor | such | such |
| carbonara | carbonara | more | more |
| chicken | chicken | more | more |
| flavors | flavor | more | more |

Table 2: Results of noun-adjective pairs using Approach 2

Using approach 2, multiple pairs are found for an adjective. For example, the word “large” is identified as an adjective which has multiple noun-adjective pairs: (pack, large), (bowl, large). We noticed that (pack, large) appeared twice in the results. We also noticed that there are five noun-adjectives pairs that are incorrectly identified, which are (flavor, such), (carbonara, more), (chicken, more), (bowl, large), (pack, large). Thus, the accuracy of the second approach = 6/11 = 0.545.

The following table is the results of the noun-adjective pairs identified using approach 3.

|  |  |  |  |
| --- | --- | --- | --- |
| **Noun** | **Noun Base** | **Adjective** | **Adjective Base** |
| challenge | challenge | real | real |
| supermarket | supermarket | asian | asian |
| pack | pack | single | single |
| pack | pack | large | large |
| flavors | flavor | different | different |

Table 3: Results of noun-adjective pairs using Approach 3

Using approach 3, each noun-adjective pair are correctly identified once. For example, (challenge, real), (supermarket, asian), (pack, single), (pack, large), (flavor, different). Thus, the accuracy of the third approach = 1. From our analysis, we have determined that the best approach to take will be Approach 3 as it has the highest accuracy and correctness.

The output of the noun-adjective pairs is generated into a CSV file “noun\_adj\_pairs.csv” in the output folder. The original word and its lemma form for both the noun and adjective are saved. For example, the adjective “best” has lemma form “good”, and (noodles, spicy) and (noodle, spicy) is considered the same pair. The frequency of the noun-adjective pairs is calculated and ranked from highest frequency to lowest.

2.2 Results of Top 5 Noun-Adjective Pairs

The following table shows result of the top 5 noun-adjective pairs with its frequency count.

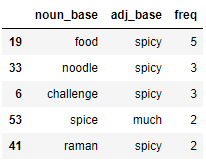


Table 4: Top 5 noun-adjective pairs frequency

The ranking is according to the word frequency instead of the meaning of the word as we are using the lemma form of the word. For example, if the original word was “best” or “good”, we are using its lemma form “good to perform the pair ranking.

Due to the limited size of the dataset, the top pair with the highest frequency is (food, spicy) with frequency = 5, followed by (noodle, spicy) with frequency = 3, followed by (challenge, spicy) with frequency = 3, followed by (spice, much) with frequency = 2, followed by (raman, spicy) with frequency = 2.

The results of the top 5 noun-adjectives from the pair ranker program is different from the 5 pairs we have manually identified. We managed to only obtain one correct noun-adjective pair (noodle, spicy) that are in the results of the top 5 noun-adjective pairs.

An interesting observation is that Spacy converted the noun “ramen” into the noun base form “raman” which is incorrect. Spacy probably did not anticipate the word “ramen” and probably detected the word “men” and converted it into “man”.

The following shows the number of pairs for each frequency of noun-adjective pairs.

|  |  |
| --- | --- |
| **Frequency** | **Number of Pairs** |
| 5 | 1 |
| 4 | 0 |
| 3 | 2 |
| 2 | 9 |
| 1 | 49 |

Table 5: Number of pairs for each frequency

A challenge that we have encountered is the small dataset of 30 reviews and some reviews are much shorter or have less sentences than other reviews. As a result, the number of highest frequency noun-adjective pairs compared to the number of lowest frequency noun-adjective pair is unbalanced. For example, there are 49 noun-adjective pairs with frequency = 1 compared to 1 noun-adjective pair with frequency = 5. A larger dataset with longer and more descriptive review texts would mitigate this issue.

3 Sentiment Analysis Application

The sentiment analysis library used in our application is called TextBlob. The sentiment analyzer contains two implementation of sentiment analysis: PatternAnalyzer which is based on the Pattern library and the NaiveBayesAnalyzer which is a Natural Language Toolkit (NLTK) classifier that has been trained on movie review corpus. The sentiment analysis is performed on the same dataset which is the 30 reviews for the product ‘Samyang Spicy Chicken Roasted Noodles’. The motivation behind using the TextBlob library is because we are using a small dataset of 30 reviews, which is inadequate to train our own model. However, using the TextBlob model which has been trained on movie reviews may not be suitable for our application due to different domains which may affect the sentiment classification.

3.1 Polarity Value

Each review’s text is input into the TextBlob’s sentiment function and produce the sentiment polarity float value, where the maximum value is the most positive with a value of 1.0 and minimum value is the most negative with a value of -1.0. For neutral, we defined its range from 0.01 to -0.01.

The following table shows the number of sentiment reviews classified as negative, neutral and positive.

|  |  |  |
| --- | --- | --- |
| **Negative** | **Neutral** | **Positive** |
| 6 | 2 | 22 |

Table 6: Number of reviews classified under each category

Out of the 30 reviews, 6 reviews were classified as negative, 2 reviews were classified as neutral and 22 reviews are positive. In addition, out of the 30 reviews, the distribution of review ratings which ranges from 1 to 5 are illustrated in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rating** | 1 | 2 | 3 | 4 | 5 |
| **Review Count** | 3 | 2 | 8 | 6 | 11 |

**Table 7: Number of reviews classified under each rating**

The following scatterplot shows the plot of review polarity against review rating. Positive reviews are plotted using green dots, neutral reviews are plotted with blue dots and negative reviews are plotted with red dots.

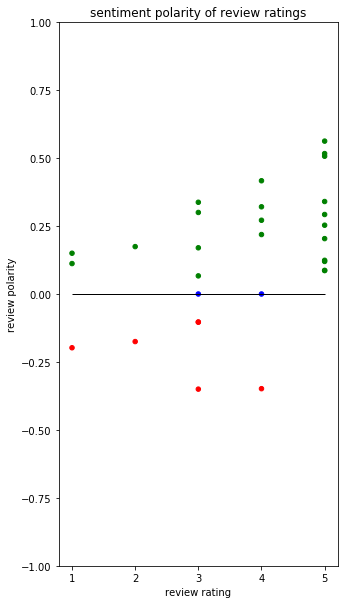


Figure 10: Scatterplot of review polarity against review rating.

From our observation, we can see that there is a pattern. A review with a higher rating such as 5 tends to have higher positive polarity of above 0.25.

3.2  Anomaly

It is observed that 2 out of 3 reviews with a rating of 1 is classified as positive and is investigated further.

Out of these two reviews, one of the review’s content is indeed written positively but was given a 1 out of 5 rating from the author. As for the other review, it contains a sentence “didn't really taste good” and it is observed that both sentences: “didn't really taste good” and “really taste good” produced the same polarity value of 0.45, a positive value. We infer that the sentiment function did not take into consideration of “didn’t”, and thus inaccurately classified the review as positive instead of negative.

4 Contributions of Individual Members

|  |  |
| --- | --- |
| **Member** | **Contributions** |
| Chulpaibul Jiraporn | Domain Specific Dataset Analysis (Patents, Cross-domain analysis) |
| Leong Ko Rixie Tiffany | Domain Specific Dataset Analysis (Stack Overflow, Research Papers) |
| Leong Kai Ling | Development of a ⟨ Noun - Adjective ⟩ Pair Ranker, Sentiment Analysis Application |
| Liew Zhi Li | Development of a ⟨ Noun - Adjective ⟩ Pair Ranker, Sentiment Analysis Application |

REFERENCES

[1] Amazon. 2020. Samyang New Ramen/Spicy Chicken Roasted Noodles, 4.93 oz (Pack of 5) (2 Case (Pack of 5)). November 2020). Retrieved November 1, 2020 from <https://www.amazon.com/gp/aw/reviews/B08113YPZK>

[2] Influenster. 2020. Samyang Ramen / Spicy Chicken Roasted Noodles 140g (November 2020). Retrieved November 1, 2020 from <https://www.influenster.com/reviews/samyang-ramen-spicy-chicken-roasted-noodles-140g>

APPENDIXConference Name:ACM Woodstock conference

APPENDIX A: POS Tagging Results

**Stack Overflow**

Sentences without POS Tags:

1. Because React is not oriented to work with nested states and all solutions proposed here look as hacks.
2. However React.Children.map unlike React.cloneElement changes the keys of the Element appending and extra .$ as the prefix.
3. React Router makes the properties and methods of the history instance associated with your router available through the context, under the router object.

Sentences with POS Tags:

1. Because/IN React/NNP is/VBZ not/RB oriented/VBN to/TO work/VB with/IN nested/JJ states/NNS and/CC all/DT solutions/NNS proposed/VBN here/RB look/VBP as/IN hacks/NNS ./.
2. However/RB React.Children.map/NNP unlike/IN React.cloneElement/NNP changes/VBZ the/DT keys/NN of/IN the/DT Element/NNP appending/NN and/CC extra/JJ ./. $/$ as/IN the/DT prefix/NN ./.
3. React/JJ Router/NNP makes/VBZ the/DT properties/NNS and/CC methods/NNS of/IN the/DT history/NN instance/NN associated/VBN with/IN your/PRP$ router/NN available/JJ through/IN the/DT context/NN ,/, under/IN the/DT router/NN object/NN ./.

**Research Papers**

Sentences without POS Tags:

1. The second year students were taking a subject ‘‘Maternal and Child Care’’, which included weekly 2-h lectures and 1-h laboratory sessions for one 14-week semester.
2. Reading storybooks to kinder- garteners helps them learn new vocabulary words.
3. Some students even confided their own experiences to each other or difficulties in dealing with schoolwork, and how they struggled with assignments and demands from the lecturers.

Sentences with POS Tags:

1. The/DT second/JJ year/NN students/NNS were/VBD taking/VBG a/DT subject/JJ ‘/NN ‘/NNP Maternal/NNP and/CC Child/NNP Care/NNP ’/NNP ’/NNP ,/, which/WDT included/VBD weekly/JJ 2-h/JJ lectures/NNS and/CC 1-h/JJ laboratory/NN sessions/NNS for/IN one/CD 14-week/JJ semester/NN ./.
2. Reading/VBG storybooks/NNS to/TO kinder-/JJ garteners/NNS helps/VBZ them/PRP learn/VB new/JJ vocabulary/JJ words/NNS ./.
3. Some/DT students/NNS even/RB confided/VBD their/PRP$ own/JJ experiences/NNS to/TO each/DT other/JJ or/CC difficulties/NNS in/IN dealing/VBG with/IN schoolwork/NN ,/, and/CC how/WRB they/PRP struggled/VBD with/IN assignments/NNS and/CC demands/NNS from/IN the/DT lecturers/NNS ./.

**Patents**

Sentences without POS Tags:

1. Heterogeneity of autoimmune diseases : pathophysiologic insights from genetics and implication for new therapies .
2. TABLE 5 Target Particle Formulation 6 (wt %) 10.9 16.5 10.6 43.3 18.7 1OOO Approximate Solid Ratio 1.O 1.6 1.O 4.1 1.8 Component Sodium Citrate Citric Acid Methionine oxyntomodulin SUCOSE total \*Sodium Citratefitric Acid formed the citrate buffer for this particle formulation.
3. Additional reactants , including other combinations of acids and bases which produce an inert gas by product are also contemplated.

Sentences with POS Tags:

1. Heterogeneity/NN of/IN autoimmune/JJ diseases/NNS :/: pathophysiologic/NN insights/NNS from/IN genetics/NNS and/CC implication/NN for/IN new/JJ therapies/NNS ./.
2. TABLE/NN 5/CD Target/NNP Particle/NNP Formulation/NNP 6/CD (/( wt/CD %/NN )/) 10.9/CD 16.5/CD 10.6/CD 43.3/CD 18.7/CD 1OOO/CD Approximate/NNP Solid/NNP Ratio/NNP 1.O/CD 1.6/CD 1.O/CD 4.1/CD 1.8/CD Component/NNP Sodium/NNP Citrate/NNP Citric/NNP Acid/NNP Methionine/NNP oxyntomodulin/MD SUCOSE/NNP total/JJ \*/NNP Sodium/NNP Citratefitric/NNP Acid/NNP formed/VBD the/DT citrate/NN buffer/NN for/IN this/DT particle/NN formulation/NN ./.
3. Additional/JJ reactants/NNS ,/, including/VBG other/JJ combinations/NNS of/IN acids/NNS and/CC bases/NNS which/WDT produce/VBP an/DT inert/NN gas/NN by/IN product/NN are/VBP also/RB contemplated/VBN ./.

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00