Attn: Dr. Sun Aixin

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.

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

Name	Signature / Date
KOH YIANG DHEE, MITCHELL	[illegible] 25/10/21
RYAN LEE RUIXIANG	[illegible] 25/10/21
LI PINGRUI	李坪芮 25/10/21
WANG BINLI	王林力 25/10/21
TANG YUTING	唐玉婷 25/10/21

CZ4045 Assignment 1 Report

KOH YIANG DHEE, MITCHELL
C190142@e.ntu.edu.sg

RYAN LEE RUIXIANG
ryan0040@e.ntu.edu.sg

LI PINGRUI
pli013@e.ntu.edu.sg

TANG YUTING
ytang021@e.ntu.edu.sg

WANG BINLI
Wang1448@e.ntu.edu.sg

Nanyang Technological University

ABSTRACT

Natural language processing (NLP) is an important topic in Computer Science. This report is for Assignment 1, CZ4045 at NTU. Various NLP tasks are conducted and analysed on the Yelp review dataset, including tokenization, stemming, POS tagging, writing style analysis, extraction of most frequent noun-adjective pair, extraction of indicative adjective phrases, and topic modelling.

1. INTRODUCTION

This assignment is to perform multiple NLP tasks on a dataset of restaurant reviews posted on yelp. The dataset contains 15300 reviews on 153 businesses. To understand the dataset, tokenization and stemming were performed first to understand the frequency of words in reviews for different businesses. POS tagging was then carried out with two difference tagging methods. A writing style analysis was conducted between posts on Stack Overflow, HardwareZone and news articles from Channel NewsAsia. Lastly, most frequent noun-adjective pair for different ratings was analyzed and attempts to figure out indicative adjective phrases for a random business was carried out. With the knowledge and understanding obtained from the data exploration, an application for topic modeling was developed.

2. DATASET ANALYSIS

2.1. Tokenization and Stemming

Libraries Used

- Regular-expression-based tokenizer provided by NTLK [1]
- Stemmer based on Porter stemming algorithm provided by NTLK

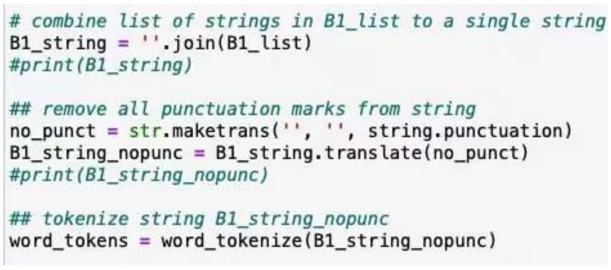
Process Summary

1. Load review dataset.
2. Randomly select Business B1.
3. Process data.
4. Extracted reviews are tokenized into words.
5. Remove meaningless words (e.g., stop words) from the list of tokens.
6. Plot word frequency distribution with top-10 most frequent words listed
7. Apply stemming
8. Repeat steps (5) and (6)
9. Repeat steps (1)-(8) for another random Business B2

Data Processing

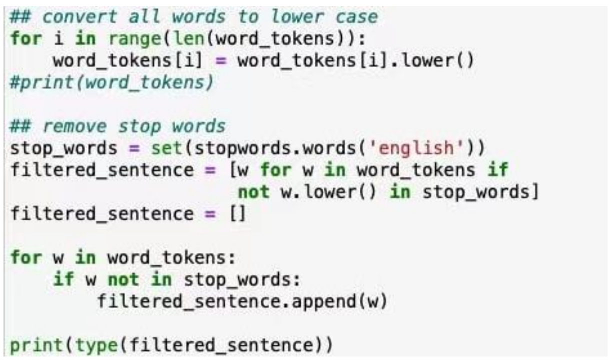
Before tokenization, the data is processed by:

- Removing all punctuations
- Uncapitalizing words
- Removing all stopping words, specified in NLTK library, such as “the”, “of”, “above”, etc.



```
# combine list of strings in B1_list to a single string
B1_string = ''.join(B1_list)
#print(B1_string)

## remove all punctuation marks from string
no_punct = str.maketrans('', '', string.punctuation)
B1_string_nopunc = B1_string.translate(no_punct)
#print(B1_string_nopunc)

## tokenize string B1_string_nopunc
word_tokens = word_tokenize(B1_string_nopunc)
```

Figure 1 the code snippet of data pre-processing



```
## convert all words to lower case
for i in range(len(word_tokens)):
    word_tokens[i] = word_tokens[i].lower()
#print(word_tokens)

## remove stop words
stop_words = set(stopwords.words('english'))
filtered_sentence = [w for w in word_tokens if
                     not w.lower() in stop_words]
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(type(filtered_sentence))
```

Figure 2 the code snippet of data pre-processing

Result

- **Word Frequency Distribution**

The word frequency plots shown below includes the top 200 most frequent words as their y-axis, and these words are arranged in descending order to observe findings.

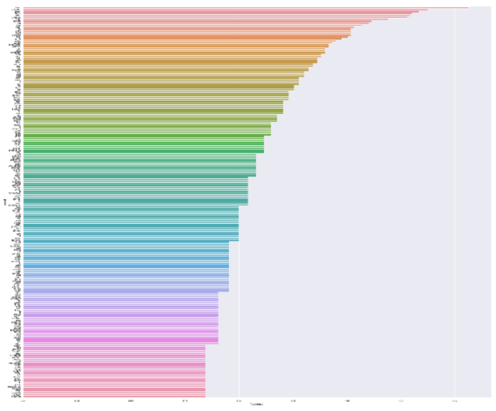


Figure 3 the word frequency distribution of B1 before stemming

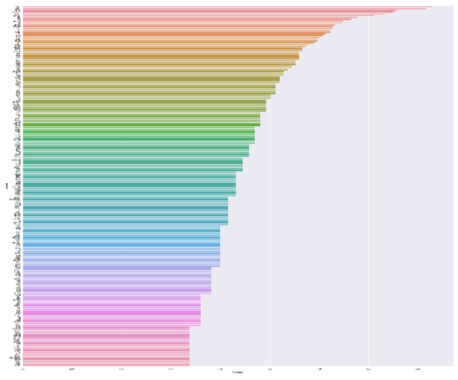


Figure 4 the word frequency distribution of B1 after stemming

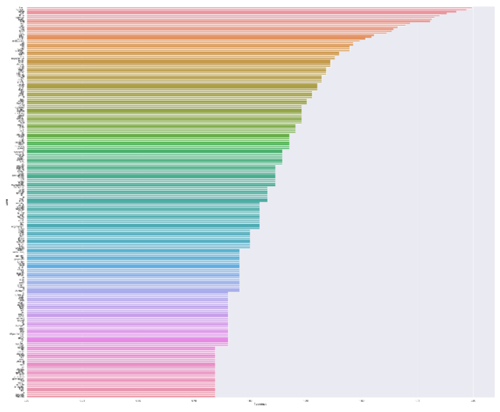


Figure 5 the word frequency distribution of B2 before stemming

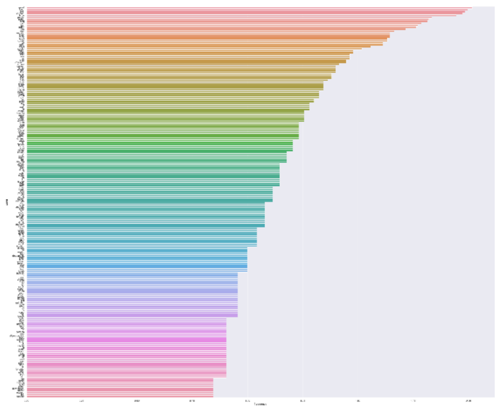


Figure 6 the word frequency distribution of B1 after stemming

1. The overall shape of the plot remains the same which indicates that after stemming had been applied, the distribution of word frequency remains about the same.
2. However, from the figures above, most of the existing bars is the bar plots have become longer after stemming. This indicates that after stemming had been applied, many of the words have been reduced to their stem which results in the increase in frequency of certain bars which represent stem words. This holds for both B1 and B2.

- Top 10 Most Frequent Words Findings
  The Top 10 most frequent words are:

  B1

  - Before stemming

```
[('food',  116),  ('chicken',  75),
('good',    68),   ('place',    63),
('order',   62),  ('chinese',   60),
('little', 49), ('shrimp', 41),
('like', 40), ('rice', 37)]
```

- After stemming

```
[('food', 119), ('order', 110),
('place', 78), ('chicken', 75),
('good', 68), ('chines', 60),
('littl', 49), ('like', 46),
('shrimp', 42), ('go', 41)]
```

B2

- Before stemming

```
[('food', 99), ('korean', 93),
('meat', 84), ('grill', 76),
('place', 71), ('service', 67),
('good', 65), ('bbq', 64), ('eat',
52), ('get', 50)]
```

- After stemming

```
[('meat', 104), ('food', 99),
('grill', 96), ('korean', 94),
('place', 88), ('servic', 68),
('good', 66), ('bbq', 65), ('get',
61), ('eat', 59)]
```

From the results obtained, after stemming had been applied, most of the top 10 frequent words have an increase in frequency. This indicates that many words that were not in their stem form have been stemmed contributing to the increase in frequency of these stem words.

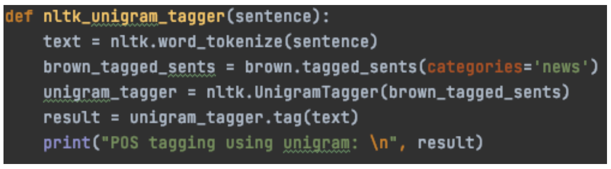
Certain words that belonged in the top 10 list have been replaced by other words in the new top 10 list after stemming. This indicates that some of these words in the stem form were in many other forms before stemming was applied which resulted in the great increase in frequency to now be in the top 10 list.

2.2. POS Tagging

Taggers Used in NLTK library

- **Unigram Tagger**

The Unigram Tagger is a statistical tagger in NLTK implements a simple tagging algorithm that is assigning each token the tag which maximize the likelihood for that tag type. It will set "None" as the default tag until its type was encountered in training. The performance of this tagger is highly relied on the training set.

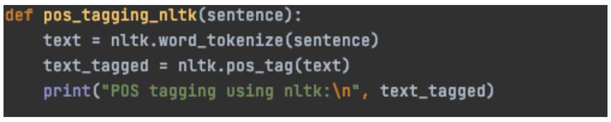


```
def nltk_unigram_tagger(sentence):
    text = nltk.word_tokenize(sentence)
    brown_tagged_sents = brown.tagged_sents(categories='news')
    unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
    result = unigram_tagger.tag(text)
    print("POS tagging using unigram: \n", result)
```

Figure 7 the code snippet of unigram tagger application

- **Perceptron Tagger**

The method pos_tag always use the currently recommended POS tagger by NLTK, the Perceptron Tagger. It implements POS tagging using a greedy, averaged, and structured perceptron algorithm. It has a dictionary of weights which are associated with predicting features of the correct tags for the given set of features. In the process of training, the tagger guesses a tag and changes weights accordingly. The weight adjustments are averaged over times of iterations.



```
def pos_tagging_nltk(sentence):
    text = nltk.word_tokenize(sentence)
    text_tagged = nltk.pos_tag(text)
    print("POS tagging using nltk:\n", text_tagged)
```

Figure 8 the code snippet of Perceptron tagger application

Results & Discussion

Sentence 1: WE LOVE THIS PLACE!

```
POS tagging using unigram:
 [('WE', None), ('LOVE', None),
('THIS', None), ('PLACE', None),
('!', '.')]
POS tagging using nltk:
 [('WE', 'NNP'), ('LOVE', 'NNP'),
('THIS', 'NNP'), ('PLACE', 'NN'),
('!', '.')]
```

Unigram Tagger: All the letters in the selected sentence are capitalized. However, most words in the training set are either lowercase words or words with a capital letter. Thus, all words in this sentence were tagged "NONE".

Perceptron Tagger: Because of the capitalization of all letters, the verb "LOVE" is wrongly assigned to NNP.

Sentence 2: Let's just say I'm never ever ever going back to this store.

```
POS tagging using unigram:
 [('Let', 'VB'), ("'s", None),
('just', 'RB'), ('say', 'VB'), ('I'
'PPSS'), ("'m", None), ('never',
'RB'), ('ever', 'RB'), ('ever',
'RB'), ('going', 'VBG'), ('back',
'RB'), ('to', 'TO'), ('this', 'DT')
('store', 'NN'), ('.', '.')]
POS tagging using nltk:
 [('Let', 'VB'), ("'s", 'POS'),
('just', 'RB'), ('say', 'VBP'),
('I', 'PRP'), ("'m", 'VBP'),
('never', 'RB'), ('ever', 'RB'),
('ever', 'RB'), ('going', 'VBG'),
('back', 'RB'), ('to', 'TO'),
('this', 'DT'), ('store', 'NN'),
('.', '.')]
```

The abbreviation cannot be recognized by Unigram Tagger such as Let's and I'm. In other respects, the two taggers perform good.

Sentence 3: When I came in to sign the paperwork I didn't have to wait more than a few minutes before we got started.

```
POS tagging using unigram:
 [('When', 'WRB'), ('I', 'PPSS'),
('came', 'VBD'), ('in', 'IN'),
('to', 'TO'), ('sign', 'VB'),
('the', 'AT'), ('paperwork', None),
('I', 'PPSS'), ('did', 'DOD'),
("n't", None), ('have', 'HV'),
('to', 'TO'), ('wait', 'VB'),
('more', 'AP'), ('than', 'IN'),
('a', 'AT'), ('few', 'AP'),
('minutes', 'NNS'), ('before',
'IN'), ('we', 'PPSS'), ('got',
'VBD'),  ('started',  'VBD'),  ('.',
'.')]
POS tagging using nltk:
 [('When',  'WRB'),  ('I',  'PRP'),
('came',   'VBD'),   ('in',   'IN'),
('to',   'TO'),   ('sign',   'VB'),
('the', 'DT'), ('paperwork', 'NN'),
('I',   'PRP'),   ('did',   'VBD'),
("n't",   'RB'),   ('have',   'VB'),
('to',   'TO'),   ('wait',   'VB'),
('more',  'JJR'),  ('than',  'IN'),
('a',    'DT'),    ('few',    'JJ'),
('minutes',   'NNS'),   ('before',
'IN'),   ('we',   'PRP'),   ('got',
'VBD'),  ('started',  'VBN'),  ('.',
'.')]
```

The abbreviation cannot be recognized by Unigram Tagger. Besides, the word "paperwork" cannot be recognized by Unigram Tagger due to the limited vocabulary of the training set.

Sentence 4: You could do worse in Streetsboro.

```
POS tagging using unigram:
 [('You', 'PPSS'), ('could', 'MD'),
('do',  'DO'),  ('worse',  'JJR'),
('in', 'IN'), ('Streetsboro', None),
('.', '.')]
POS tagging using nltk:
 [('You',  'PRP'),  ('could',  'MD'),
('do',  'VB'),  ('worse',  'JJR'),
('in',    'IN'),    ('Streetsboro',
'NNP'), ('.', '.')]
```

The word "Streetsboro", a city in America, cannot be detected by Unigram Tagger because it is a rarely seen proper noun. However, this unknow word can be tagged as NNP by Perceptron Tagger.

Sentence 5: I wish I remembered his name so I could give a proper complaint.

```
POS tagging using unigram:
 [('I',  'PPSS'),  ('wish',  'VB'),
('I',    'PPSS'),    ('remembered',
'VBN'),  ('his',  'PP$'),  ('name',
'NN'), ('so', 'QL'), ('I', 'PPSS'),
('could',  'MD'),  ('give',  'VB'),
('a',   'AT'),   ('proper',   'JJ'),
('complaint', 'NN'), ('.', '.')]
POS tagging using nltk:
 [('I',  'PRP'),  ('wish',  'VBP'),
('I', 'PRP'), ('remembered', 'VBD'),
('his',  'PRP$'),  ('name',  'NN'),
('so',   'RB'),    ('I',    'PRP'),
('could',  'MD'),  ('give',  'VB'),
('a',   'DT'),   ('proper',   'JJ'),
('complaint', 'NN'), ('.', '.')]
```

The Unigram Tagger cannot flexibly assigned the different terms of verbs' tags for a word such as "wish", "remembered" in the above sentence while Perceptron Tagger can handle it well.

Conclusion

In conclusion, the Perceptron Tagger has shown a better performance than the Unigram Tagger in the following aspects:

Unigram Tagger is highly dependent on the quality of the training set. The limitation of training set causes plenty of unrecognition:

- Words with unusual case formats can be successfully tagged by Perceptron Tagger but cannot be recognized by Unigram Tagger (will be tagged "NONE").
- The abbreviation can be successfully tagged by Perceptron Tagger but cannot be recognized by Unigram Tagger (will be tagged "NONE").
- The rarely seen proper words can be successfully tagged by Perceptron Tagger but cannot be recognized by Unigram Tagger (will be tagged "NONE").

Unigram Tagger otherwise than Perceptron Tagger which concerns about the context, simply assign the most frequently appeared tag to the word without considering the various terms of verbs which results in many unproperly tagging.

2.3. Writing Style

Data Source

The data analyzed is randomly sampled from various domains on the Internet.

- two posts from Stack Overflow
  - https://stackoverflow.com/q/69678014/11180198
  - https://stackoverflow.com/q/69672969/11180198
- two posts from HardwareZone
  - https://www.hardwarezone.com.sg/review-asus-zenbook-14x-oled-ux5400-review-singapore-price-specs
  - https://www.hardwarezone.com.sg/tech-news-2022-macbook-air-could-have-white-bezels-mini-led-and-usb-c-only-ports
- two news articles from Channel NewsAsia
  - https://www.channelnewsasia.com/business/epic-games-opposes-apples-effort-pause-antitrust-trial-orders-2263607
  - https://www.channelnewsasia.com/singapore/car-crashes-ulu-pandan-community-club-2-taken-hospital-2263552

Only the body content is kept during processing stage. The sentences are split into lines manually. For the code segment, a statement is treated as a sentence.

Probability of Sentences Starting with Capital Letter

The probability of sentences starting with capital letter is calculated by

$$P = \frac{Number\ of\ sentences\ with\ initial\ captial\ letter}{Total\ Sentence\ Number}$$

The result is shown in Table 1.

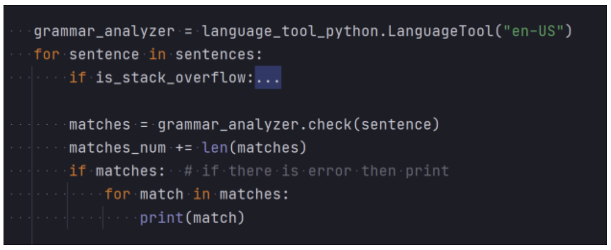
Table 1 probability of sentences starting with capital letter

	Stack Overflow	HardwareZone	CNA
P	0.87	0.98	1

Higher *P* means the text in the domain is more likely to follow the grammar rule of starting a sentence with a capital letter. The result shows that in terms of *P*, CNA > HardwareZone > Stack Overflow.

Grammar

The LanguageTool [2] is used for grammar checking as shown in Figure 9.



```python
grammar_analyzer = language_tool_python.LanguageTool("en-US")
for sentence in sentences:
    if is_stack_overflow:...

    matches = grammar_analyzer.check(sentence)
    matches_num += len(matches)
    if matches:  # if there is error then print
        for match in matches:
            print(match)
```

Figure 9 the code snippet of grammar checking

Every sentence is checked by the grammar checker, and the grammar errors are recorded:

- CNA Domain
  - 1 grammar error on "which requires Apple show"
- HardwareZone Domain
  - 4 punctuation errors on the missing comma between two independent clauses
  - Many false positive misspelling errors on strange proper nouns like "ZenBook"
  - 3 space errors about "16.9mm" and "1.4kg" which miss the space between numbers and units
- Stack Overflow
  - Typo grammar error like "such a its price"
  - Uncapitalized "I"
  - Many punctuation errors on the missing comma between two independent clauses
  - Some sentences not starting with capital letter

Proper Noun Capitalization

Through manual check and the noun checking tool[1], the result on proper noun capitalization is as shown below:

- Hardware Zone
  - The proper nouns, such as the products' names and computer terminologies, all start with capital letter.
- Stack Overflow
  - 2 proper nouns do not start with capital letter.
- CNA
  - All proper nouns start with capital letter

Findings

Based on the results from the perspectives on capitalization and grammar checking, the findings are:

1. The order on the correctness of the grammar is CNA > HardwareZone > Stack Overflow.
2. HardwareZone has the most unknown proper nouns like the products' names.

Tokenization and POS tagging on Stack Overflow

- **Tokenization**

The regular-expression-based tokenizer is applied to the text from Stack Overflow. Some examples are shown below:

```
['with', 'all_views', 'as', '(',
'select', '*', 'from',
'information_schema.views', 'where',
'table_schema', '!', '=',
"'INFORMATION_SCHEMA", "'", ')']

['SELECT', '*', 'FROM', 'TABLE',
'(', 'get_object_references', '(',
'database_name=', '>',
'all_views.TABLE_CATALOG', ',',
'schema_name=', '>',
'all_views.TABLE_SCHEMA', ',',
'object_name=', '>',
'all_views.TABLE_NAME', ')', ')',
';']

['I',   'ran',   'the',   'above',
'query', 'in', 'snowflake', 'but',
'getting', 'the', 'below', 'error',
'.']
```

The code segment is not tokenized properly, for example "object_name=" and separated '!', '=' for non-equal operator.

- **POS Tagging**

The toolkit spaCy [3] was used for POS tagging here. For the Stack Overflow sentences, although the grammar mistakes were widely existed, the performance of spaCy tagger is mostly good and accurate.

Most terminology and instance's names are tagged properly to proper noun (NNP) such as the "flutter_cs_cahce", "ItemId" and "API". However, when encountering sections of code, its performance becomes bad. The example is as below.

```
The sentence is:  with all_views as
(select * from
information_schema.views  where
table_schema !=
'INFORMATION_SCHEMA')
------------------------------
POS tagging using spacy:
 [('with', 'IN'), ('all_views',
'NN'), ('as', 'IN'), ('(', '-LRB-'),
('select', 'JJ'), ('*', 'NN'),
('from', 'IN'),
('information_schema.views', ':'),
(' ', '_SP'), ('where', 'WRB'),
('table_schema', 'NN'), ('!', '.'),
('=', 'NFP'), ("'", "''"),
('INFORMATION_SCHEMA', 'PRP'), ("'",
"''"), (')', '-RRB-')]
--------------------
```

The proper nouns "all_views", "information_schema.views", "table_schema" and "INFORMATION_SCHEMA" were tagged inadequately.

[1] The link of tool: https://inkforall.com/noun-checker

Besides, due to the limitation of tag list of spaCy, many special characters within the code were tagged improperly. The example is as below.

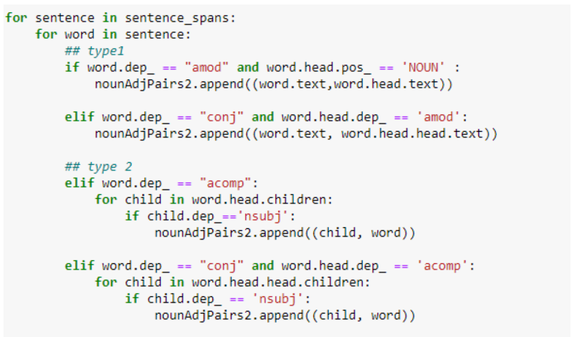
```
The sentence is:  SELECT * FROM
TABLE(get_object_references(databas
e_name=>'<db_name>',
schema_name=>'<schema_name>',
object_name=>'<view_name>'));
--------------------
POS tagging using spacy:
 [('SELECT', 'NNP'), ('*', 'NFP'),
('FROM', 'IN'),
("TABLE(get_object_references(datab
ase_name=>'<db_name", 'NNP'), ('>',
'NFP'), ("'", "'"), (',', ','),
("schema_name=>'<schema_name",
'VBD'), ('>', 'ADD'), ("'", "'"),
(',', ','),
("object_name=>'<view_name", 'NNP'),
('>', 'ADD'), ("'", "'"), (')', '-
RRB-'), (')', '-RRB-'), (';', ':')]
```

2.4. Most frequent ⟨ Noun - Adjective ⟩ pairs for each rating

This section uses the spaCy python library. More specifically, it uses the en_core_web_sm model of spaCy for tagging and displaCy for dependency visualization. The process to obtain the most common Noun-Adjective pairs for each rating can be summarized in the following steps:

1. Reviews of businesses are extracted from the data and there is no more than 1 review per business
2. The text portion of the reviews are extracted and split into a list of sentences
3. Each sentence is passed through the en_core_web_sm tagger
4. Visualization of the dependencies between words is generated using displaCy
5. An algorithm is used to generate all Noun-Adjective pairs
6. The 10 most frequent pairs are identified

Algorithm to Extract Noun-Adjective Pairs



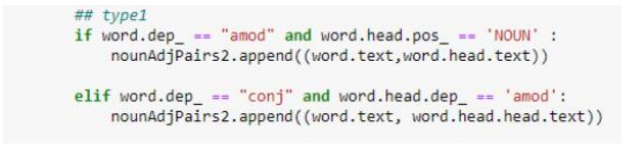
```python
for sentence in sentence_spans:
    for word in sentence:
        ## type1
        if word.dep_ == "amod" and word.head.pos_ == 'NOUN' :
            nounAdjPairs2.append((word.text,word.head.text))

        elif word.dep_ == "conj" and word.head.dep_ == 'amod':
            nounAdjPairs2.append((word.text, word.head.head.text))

        ## type 2
        elif word.dep_ == "acomp":
            for child in word.head.children:
                if child.dep_=='nsubj':
                    nounAdjPairs2.append((child, word))

        elif word.dep_ == "conj" and word.head.dep_ == 'acomp':
            for child in word.head.head.children:
                if child.dep_ == 'nsubj':
                    nounAdjPairs2.append((child, word))
```

Figure 10 code segment to extract noun-adjective pairs

The algorithm makes use of both the POS tags of words and the dependencies between words to extract the Noun-Adjective Pairs. There are two different conditions(types) to identify noun-adjective pairs. These two conditions utilize two main dependencies. They are *amod* dependency and *acomp* dependency.

The first way to identify noun-adjective pairs is with the *amod* dependency. It is the dependency between an adjectival modifier and a noun phrase. The adjectival phrase in this case serves to modify the meaning of the noun phrase. An example of this dependency is: *amod*(clown, happy) in *"The happy clown"*

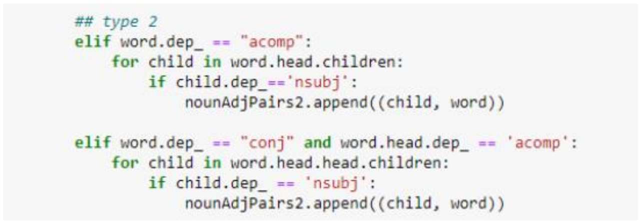
Therefore, the first condition of the algorithm is to pick out all *amod* dependencies and check if the head of the dependency is a noun. If so, the pair of words is a noun-adjective pair and is extracted. Aside from solely using the *amod* dependency we can also utilize a secondary dependency called *conj*. *Conj* is the relation between two elements connected by a coordinating conjunction, such as "and" or "or". Hence, allowing us to pick out nouns-adjective pairs for scenarios where there are multiple adjectives modifying a particular noun. An example to illustrate this is: *"The black and blue sheep" amod*(sheep, black), *conj*(black, blue) Noun-adjective pairs: (sheep, blue) and (sheep, black)



```python
## type1
if word.dep_ == "amod" and word.head.pos_ == 'NOUN' :
    nounAdjPairs2.append((word.text,word.head.text))

elif word.dep_ == "conj" and word.head.dep_ == 'amod':
    nounAdjPairs2.append((word.text, word.head.head.text))
```

Figure 11 the code snippet of type 1

The second way we identify noun-adjective phrase is using the *acomp* dependency. The *acomp* is a relationship between a verb and an adjective phrase. After identifying this verb, we use the *nsubj* dependency to see if the verb has any noun subjects. If there is a noun subject relationship, we can infer that the adjective found earlier describes this noun and we have found a noun-adjective pair. For example: "*The waiter looks pleasant*" *acomp*(looks, pleasant), *nsubj*(looks, waiter) Noun-adjective pair: (waiter, pleasant) Like type 1, we can also make use of the *conj* dependency in the same way explained above to check if there is more than one adjective modifying the verb.



```python
## type 2
elif word.dep_ == "acomp":
    for child in word.head.children:
        if child.dep_=='nsubj':
            nounAdjPairs2.append((child, word))

elif word.dep_ == "conj" and word.head.dep_ == 'acomp':
    for child in word.head.head.children:
        if child.dep_ == 'nsubj':
            nounAdjPairs2.append((child, word))
```

Figure 12 the code snippet of type 2

Table 2 the pairs under different ratings

Rating	Most common Noun-adjective pairs
1 star	[(('desk', 'front'), 4), (('service', 'rude'), 3), (('experience', 'bad'), 3), (('service', 'such'), 2), (('service', 'poor'), 2), (('i', 'disappointed'), 2), (('place', 'other'), 2), (('time', 'second'), 2), (('experience', 'worst'), 2), (('hours', 'few'), 2)]

2 stars	[(('food', 'good'), 5), (('time', 'first'), 3), (('management', 'new'), 3), (('time', 'last'), 2), (('management', 'old'), 2), (('service', 'slow'), 2), (('service', 'bad'), 2), (('they', 'defensive'), 2), (('office', 'front'), 2), (('locations', 'few'), 1)]
3 stars	[(('eats', 'cheap'), 2), (('service', 'good'), 2), (('I', 'fine'), 2), (('food', 'great'), 2), (('picture', 'top'), 2), (('matte', 'true'), 2), (('finger', 'middle'), 2), (('things', 'many'), 1), (('place', 'famed'), 1), (('critics', 'various'), 1)]
4 stars	[(('rice', 'fried'), 5), (('place', 'busy'), 4), (('beans', 'green'), 2), (('service', 'good'), 2), (('place', 'nice'), 1), (('suites', 'small'), 1), (('area', 'small'), 1), (('shuttle', 'free'), 1), (('amenities', 'more'), 1), (('security', 'more'), 1)]
5 stars	[(('options', 'many'), 2), (('staff', 'friendly'), 2), (('I', 'disappointed'), 2), (('place', 'little'), 2), (('place', 'great'), 2), (('bite', 'better'), 2), (('service', 'great'), 1), (('She', 'personable'), 1), (('amount', 'right'), 1), (('She', 'amount'), 1)]

Discussion and Limitations

The first limitation is that the most common noun-adjective pairs occur 2-3 times only. Hence, they are not a good representation of the data set. Moreover, some "most common" pairs occurred 1 time, meaning that we are unable to find 10 representative pairs. We might to extract noun-adjective pairs from much more than 50 reviews for useful results.

The second limitation is that some noun-adjective pairs are not accurate to what the review is trying to say. For the sentence, *the service was not bad*, the noun-adjective pair generated here would be *service-bad*, but the review is trying to say the service is not bad. To resolve this issue, we need to extract more than a pair of words especially when there is a negation relationship.

The third limitation is the abundance of neutral noun-adjective pairs. For example, *desk-front* which is the most common result for the 1-star reviews but does not tell us anything about the set of reviews. To improve this, we could use a sentiment dictionary and only extract pairs with sentiment value.

To sum it up, this method of obtaining the most common noun-adjective pairs is not a good representation of the set of reviews. More dependencies other than noun-adjective pairs must be identified to get representative results.

3. EXTRACTION OF INDICATIVE ADJECTIVE PHRASES

Methodology

Extraction of indicative adjective phrases about a certain business can be done by comparing the probability of an adjective phrase appearing in reviews for that business and in reviews for other businesses.

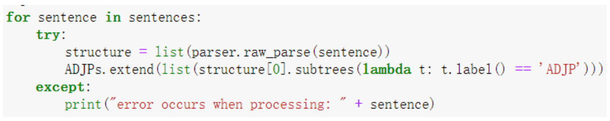
Process

The task can be carried out in the following steps:

1. Extract all adjective phrases in the reviews for the selected business
2. Compute probability in reviews for the selected business and in other businesses
3. Figure out indicative phrases by comparing the difference between probabilities

Extraction of Adjective Phrases

Stanford NLP parser is used to analyze the constituent structure of the sentences. To extract all adjective phrases, all subtrees labeled with 'ADJP' will be extracted for further processing.

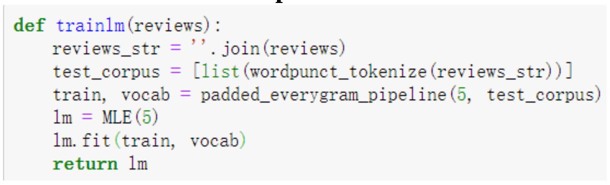


```
for sentence in sentences:
    try:
        structure = list(parser.raw_parse(sentence))
        ADJPs.extend(list(structure[0].subtrees(lambda t: t.label() == 'ADJP')))
    except:
        print("error occurs when processing: " + sentence)
```

Figure 13

The constituents will be clean to remove unnecessary cases. First, an adjective phrase constituent may contain more than one parallel child adjective phrase, hence only child phrases are kept avoiding redundant computation. Second, although labeled with 'ADJP', some phrases are composed of a single adjective. Since the task is only interested in adjective phrases, the single word constituents are discarded. Thirdly, there are long adjective phrases, which are not very likely to occur in reviews. To speed up the processing, phrases with more than 5 words are discarded. Forth, punctuation marks in phrases are replaced with empty spaces.

Criterion for indicative phrases



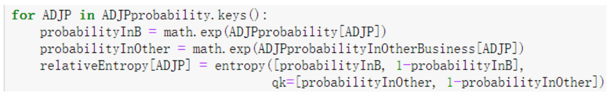
```
def trainlm(reviews):
    reviews_str = ''.join(reviews)
    test_corpus = [list(wordpunct_tokenize(reviews_str))]
    train, vocab = padded_everygram_pipeline(5, test_corpus)
    lm = MLE(5)
    lm.fit(train, vocab)
    return lm
```

Figure 14

With the n-gram model, the probability of an adjective phrases to occur can be calculated, and an adjective phrase is deemed indicative if the probability of the phrase to appear in reviews for the selected business is high and probability to appear in other reviews in relatively low. To compare the difference between 2 probabilities, relative entropy can be used.

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

The higher the relative entropy, the two probability distributions are more different, which means the phrase is a possible indicative phrase.



```
for ADJP in ADJPprobability.keys():
    probabilityInB = math.exp(ADJPprobability[ADJP])
    probabilityInOther = math.exp(ADJPprobabilityInOtherBusiness[ADJP])
    relativeEntropy[ADJP] = entropy([probabilityInB, 1-probabilityInB],
                                    qk=[probabilityInOther, 1-probabilityInOther])
```

Figure 15

Result

In the selected business, the following adjective phrases have infinite relative entropy and are the possible indicative phrases

```
'turbo charged', 'bodega ever',
'insanely delicious', 'plentiful
too', 'leaderless american','kinda
soggy', 'pippin hot', 'pure
awesomeness ', 'so legit', 'stale
spicy', 'surprisingly organized',
'disgusting orange', 'awesome open',
'perfect by themselves', 'amazing
late night', 'extraordinary very
impressive', 'huuuge and delicious',
'fastest and biggest', 'kind of
scary', 'not really mexican',
'multicultural and diverse',
'convenient in convenience', 'new
to tempe', 'a little finer', 'drab
and dingy', 'awesome authentic and
cheap', 'definition college town
convenient', 'uninviting from the
outside', 'too much to drink night',
'the guy the register too'
```

Analysis

Among the adjective phrases with large relative entropy, some indeed reflect the unique features of the selected business, but some are not valid result.

Types of invalid result

1. invalid adjective phrase

Phrases like 'the guy the register too', 'bodega ever' and ' convenient in convenience' are not valid adjective phrases. The parser fails to figure out the correct constituent structure because the writing style of the reviews is casual and some abbreviations, cyber language, slang, and non-standard spelling appear frequently in the reviews.

For example, 'the guy the register too' appears originally in sentence ‘The cook is super nice and the guy @ the register too’ and punctuations are removed when rebuilding the phrase from parsing tree. The usage of @ is not aligned with standard English grammar, hence the parser fails to attach the correct tag to the phrase. Similarly, ‘bodega ever’ appears as ‘Best bodega EVER!’ in the original text. The sentence is contracted and leaves out many grammatical elements, hence is not parsed correctly.

2. uncommon words and phrases

It can be observed that some phrases are uncommon or contain uncommon words in the context of restaurant review.

Based on the result, relative entropy of the phrase 'turbo charged' is high. However, the reason for high entropy is 'turbo charged' rarely appears in any restaurant review but appears in reviews for the selected business by coincidence, which can’t reflect any characteristic. Another example is ‘pippin hot’, which contains a wrongly spelled word ‘pipping’, hence such a word is highly unlikely to appear in other reviews, which result in high relative entropy.

By manually going through the reviews for the selected business, some characteristics are identified.

Table 3 unique characteristic of the business and relevant adjective phrases

Unique characteristic	Relevant adjective phrases
The restaurant is inside a convenience store and the appearance is not attractive	‘uninviting from the outside’ and ‘drab and dingy’
open until late night	'amazing late night'
cheap	'awesome authentic and cheap'
large portion size	'huuuge and delicious', 'fastest and biggest'
offer alcohol	None
fast service	'fastest and biggest'
authentic Mexican food	'awesome authentic and cheap'

Most of the unique features are reflected in the selected adjective phrases, however, the indicative phrases only accounts for a small portion of the selected phrases.

Issues with current methodology and potential improvement

- **The 2 types of invalid cases in selected adjective phrases**

Out of the 30 selected adjective phrases, 6 are incorrectly identified as adjective phrases due to the casual writing style and non-standard English expression. The parser used is Stanford NLP parser, which is trained on Penn treebank. A parser trained on corpus that covers sentences and words on the internet may perform better in such case.

There are 3 uncommon phrases in the selected phrase. The issue associated with uncommon phrases arises from the definition of indicative adjective phrase. Although the probability of an adjective phrase appearing in reviews for the business is low, it will be relatively high if the probability is even lower for other reviews. This problem can’t be eliminated, but with more reviews for the selected business, the effect could be reduced.

- **Nature of adjective phrase**

It can be observed that the same characteristic of the restaurant is expressed differently by different users. For example, the unique character of opening until late night can be expressed in the following different ways:

‘This is my go to late night Mexican food place’

‘Ate here twice only because they were open late and We were desperate.’

‘These guys know how to serve quality food at affordable prices and even late at night to serve all your hunger needs at any time.’

‘I stopped in late about 9pm after class one day and ordered a carne asada burrito and drinky-drank to sip on.’

Firstly, the characteristics are not necessarily expressed with adjective phrases. They could be expressed without any adjective using noun phrases, prepositional phrases, attributive clauses, participial, etc.

Secondly, the rest part in an adjective phrase except the head adjective could be adverbs or prepositional phrases, or clauses, which could be very diversified. Even though 2 adjective phrases can have the same head adjective and similar meaning, the choice of the rest part and the sequence of the words may vary, hence they are considered as different adjective phrases and the probability will be computed separately. For instance, ‘really spicy’, ‘very spicy’ and ‘so spicy’ express the same meaning but are considered as different adjective phrases. In this case, although the reviews mention the spicy flavor many times, the probability of each different adjective phrases may still be low hence not considered as indicative adjective phrases. At the same time, phrases appear frequently must be common usage hence are very likely to appear in other reviews. This will in turn make it more possible to have the uncommon phrases identified as indicative phrases.

Conclusion

due to the nature of adjective phrase, the probabilistic model can’t perform a good job in identifying the indicative adjective phrases with limited size of training data. Deep learning models might have better performance and probabilistic models might do a better job in identifying indicative noun phases which have less variation than adjective phrases.

4. APPLICATION: TOPIC MODELING AND INDICATIVE PHRASES

Motivation

An analysing tool is helpful for the business to gain insight from numerous reviews from the Internet. Therefore, the application proposed for this assignment is an automatic review analyser. Given the dataset of reviews and the business id, the tool extracts the reviews about the business and outputs the analysis results of topic modelling.

Methodology

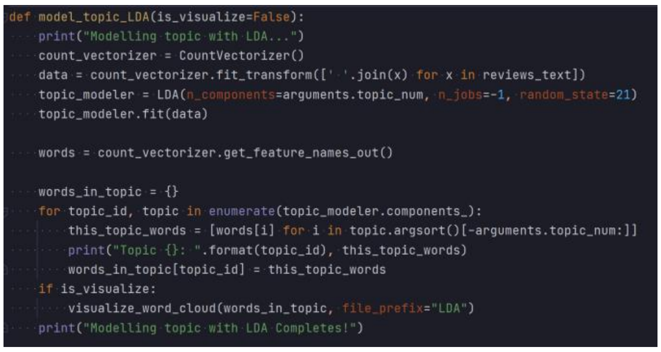
The algorithms to model the topic of reviews are Gibbs Sampling Dirichlet Multinomial Mixture (GSDMM) and Latent Dirichlet Allocation (LDA).

Latent Dirichlet Allocation (LDA) assumes one review is the combination of multiple topics, while GSDMM assumes short text tends to refer to only one topic.

Result

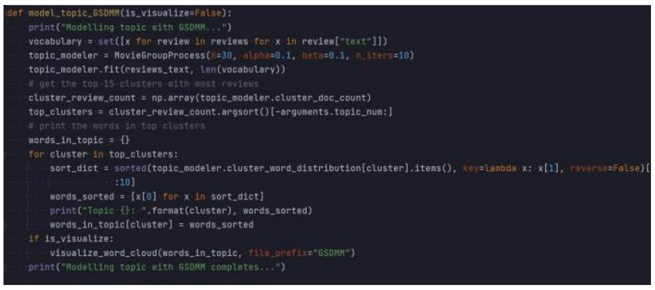
The application takes the arguments, including business_id of interests and the number of topics, specified by the user from the command line.

The application pre-processes the data and conduct topic modeling as shown in Figure 16 and Figure 17.



```python
def model_topic_LDA(is_visualize=False):
    print("Modelling topic with LDA...")
    count_vectorizer = CountVectorizer()
    data = count_vectorizer.fit_transform([' '.join(x) for x in reviews_text])
    topic_modeler = LDA(n_components=arguments.topic_num, n_jobs=-1, random_state=21)
    topic_modeler.fit(data)

    words = count_vectorizer.get_feature_names_out()

    words_in_topic = {}
    for topic_id, topic in enumerate(topic_modeler.components_):
        this_topic_words = [words[i] for i in topic.argsort()[-arguments.topic_num:]]
        print("Topic {}: ".format(topic_id), this_topic_words)
        words_in_topic[topic_id] = this_topic_words
    if is_visualize:
        visualize_word_cloud(words_in_topic, file_prefix="LDA")
    print("Modelling topic with LDA Completes!")
```

Figure 16 the code snippet of topic modelling in LDA



```python
def model_topic_GSDMM(is_visualize=False):
    print("Modelling topic with GSDMM...")
    vocabulary = set([x for review in reviews for x in review["text"]])
    topic_modeler = MovieGroupProcess(K=30, alpha=0.1, beta=0.1, n_iters=10)
    topic_modeler.fit(reviews_text, len(vocabulary))
    # get the top 15 clusters with most reviews
    cluster_review_count = np.array(topic_modeler.cluster_doc_count)
    top_clusters = cluster_review_count.argsort()[-arguments.topic_num:]
    # print the words in top clusters
    words_in_topic = {}
    for cluster in top_clusters:
        sort_dict = sorted(topic_modeler.cluster_word_distribution[cluster].items(), key=lambda x: x[1], reverse=False)[
                    :10]
        words_sorted = [x[0] for x in sort_dict]
        print("Topic {}: ".format(cluster), words_sorted)
        words_in_topic[cluster] = words_sorted
    if is_visualize:
        visualize_word_cloud(words_in_topic, file_prefix="GSDMM")
    print("Modelling topic with GSDMM completes...")
```

Figure 17 the code snippet of topic modelling in GSDMM

Given the business to analyze is AktuBx1W7c3ZdzwuaOp8xg and the number of topics to model is 10, LDA produces 10 topics and GSDMM produces 9 clusters after convergence.

From each topic, top 10 most frequent words are printed and visualized in a word cloud. Figure 18 and Figure 19 show different results on topic modeling.

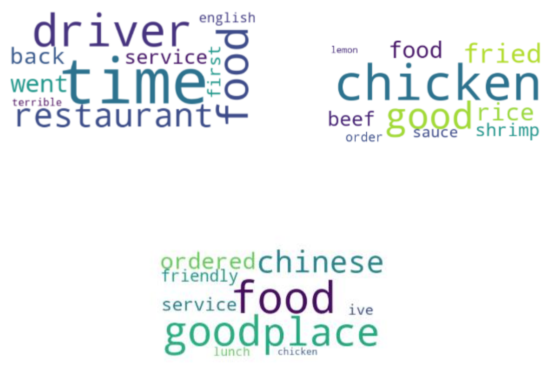


Figure 18 Three sample word clouds produced by GSDMM

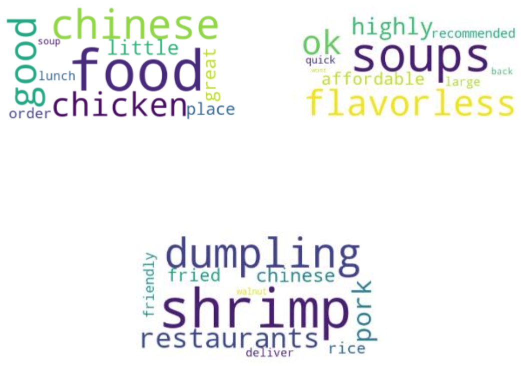


Figure 19 Three sample word clouds produced by LDA

5. REFERENCE

[1] "NLTK::Natural Language Toolki," [Online]. Available: https://www.nltk.org/index.html. [Accessed October 2021].

[2] jxmorris12, "language_tool_python: a free python grammar checker," Github, [Online]. Available: https://github.com/jxmorris12/language_tool_python. [Accessed October 2021].

[3] "spaCy - Industrial-Strength Natural Language Processing," [Online]. Available: https://spacy.io/. [Accessed October 2021].

[4] SciPy.org, [Online]. Available: https://www.scipy.org/. [Accessed October 2021].

[5] The Stanford Natural Language Processing Group, "Software - The Stanford Natural Language Processing Group," [Online]. Available: https://nlp.stanford.edu/software/. [Accessed October 2021].

[6] "Kullback-Leibler Divergence," ScienceDirect, [Online]. Available: https://www.sciencedirect.com/topics/mathematics/kullback-leibler-divergence. [Accessed October 2021].

[7] "seaborn: statistical data visualization," [Online]. Available: https://seaborn.pydata.org/. [Accessed October 2021].