Attn: Dr. Sun Aixin

AI6122 Text Data Management and 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.

Text

Name	Signature / Date
ZHAO ZEMING	2021/10/24
ZHANG JIAHE	2021/10/24
ZHENG WEIXIANG	2021/10/24
ZHAO LINGZHUO	2021/10/24
MO WEIQI	2021/10/24

AI6122 Text Data Management and Processing

Review Data Analysis and Processing

Weixiang Zheng†
School of Computer Science and Engineering
Nanyang Technological University
Singapore
wzheng014@e.ntu.edu.sg

Jiahe Zhang†
School of Computer Science and Engineering
Nanyang Technological University
Singapore
e210039@e.ntu.edu.sg

Zeming Zhao
School of Computer Science and Engineering
Nanyang Technological University
Singapore
zzhao015@e.ntu.edu.sg

Weiqi Mo†
School of Computer Science and Engineering
Nanyang Technological University
Singapore
weiqi001@e.ntu.edu.sg

Lingzhuo Zhao†
School of Computer Science and Engineering
Nanyang Technological University
Singapore
zhao0358@e.ntu.edu.sg

ABSTRACT

The report is to get familiar with the main components in end-to-end text management and processing applications, the challenges faced by each component and the solutions. We get hands on experiences on various packages available for information retrieval and natural language processing tasks.

KEYWORDS

Text data, data analysis, search engine, sentiment analysis, POS tagging, tokenization

1. Dataset

We will use the Yelp Open Dataset. We will only process the yelp_academic_dataset_review.json file under the JSON dataset. The word count terminal tells us there are in total 8635403 lines and 967361446 words in the file. Each review is one line in the file consisting of the following components: review id, user id, business id, stars, date, text, useful, funny, cool. Most of our work is based on the comment text, which is the text field of the review. For example, in Section 2 Dataset Analysis, we will do POS tagging on the text field of every review and also comment on the writing style of it, and at the end, extract the most frequent adjective-noun pair.

2. Dataset Analysis

Tokenization and stemming.

Code Snippet:

```
# First randomly choose a business b1
# Then form a dataset B1 that contains only the reviews for b1
def r_dataset(filename):
    with open(filename,'r') as f:
        lines = f.readlines()
    n_lines = len(lines)
    print(f"> Success! There are in total:{n_lines} lines")
    random_choice = random.randint(0, n_lines-1)
    # random_business = json.loads(lines[random_choice])['business_id']
    random_business = "PEnMU_He_qHoCfdoAKmjDQ" # hardcoding business
    print(f"> Success! The randomly chosen business_id is {random_business}")
    all_data = [json.loads(line) for line in lines]
    selected_business_reviews = \
    [one_data['text'] for one_data in all_data if one_data['business_id'] == random_business]
    print(len(selected_business_reviews))
    return selected_business_reviews

def break_words(reviews):
    words = []
    for s in reviews:
        words += [word for word in word_tokenize(s) if word not in stop_words and word.isalnum()]
    return words

def generate_plot_data(words):
    word_freq_d = dict(Counter(words))
    inv_dict = {}
    for k,v in word_freq_d.items():
        inv_dict[v] = inv_dict.get(v, []) + [k]
    kv_pairs = sorted([(k,v) for k,v in inv_dict.items()])
    print(kv_pairs[-10:])
    xy_pairs = sorted([(k,len(v)) for k,v in inv_dict.items()])
    x,y = zip(*xy_pairs)
    return x,y
```

Our method:

In this section, we use Python and NLTK package[1] to do text data processing. Firstly, we read all lines and parse each of them as json object. Then we randomly selected a business. It's the business with business id PEnMU_He_qHoCfdoAKmjDQ. Next, We will filter all reviews for that particular business. Next, we take all comments out and break all sentences into single words.

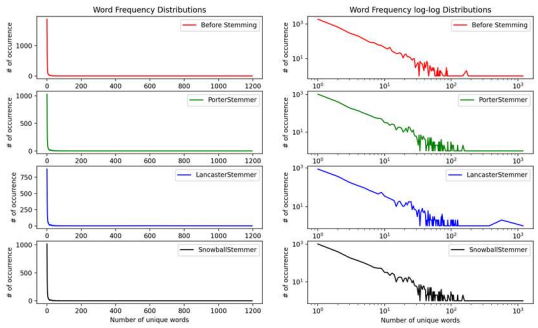
```
def porter(words):
    ps = PorterStemmer()
    return [ps.stem(word) for word in words]

def lancaster(words):
    lc = LancasterStemmer()
    return [lc.stem(word) for word in words]

def snowball(words):
    ss = SnowballStemmer('english')
    return [ss.stem(word) for word in words]
```

For these words, we plot four graphs in total: the word frequency distribution before stemming, and after stemming using three different approaches (Porter[2], Lancaster, Snowball). From the plot, we can see that there are very few numbers of words that appear very often, and as the rank grows, the number of occurrences drops drastically. If for every distribution, we take the log of it, and put the distribution on a log-log scale plot, result looks very much like a decreasing straight line, indicating an inverse relation between log of rank and log of frequency, which is consistent with Zipf's law. Also, for comment text of this particular business_id, there is a zigzag shape near the end of tail, this is because the sample size is not enough and there is some random fluctuation.

plots:



Top-10 most frequent words (exclude stop words) for business 1:

Rank	Number of Occ	Word
1	1196	I
2	565	The
3	359	Burger
4	331	Good
5	250	We
6	239	Great
7	235	Food
8	197	Place
9	180	Back
10	169	Fries; Service

Top-10 most frequent words (Porter Stemmer) for business 1:

Rank	Number of Occ	Word
1	1196	I
2	572	Burger
3	572	The
4	365	Good
5	307	Great
6	270	Food
7	251	We
8	235	Order
9	227	Fri
10	219	Place

Top-10 most frequent words (Lancaster Stemmer) for business 1:

Rank	Number of Occ	Word
1	1196	I
2	573	Burger; ~~The~~
3	372	Serv
4	365	Good
5	308	Gre
6	270	Food
7	251	We
8	235	Ord
9	224	Fri
10	219	Plac

Top-10 most frequent words (Snowball Stemmer) for business 1:

Rank	Number of Occ	Word
1	1196	I
2	573	Burger
3	572	The
4	365	Good
5	308	Great
6	270	Food
7	251	We
8	235	Order
9	227	Fri
10	219	Place

From the tables above, we can see that different stemmers show very little differences; whether or not use a stemmer, however, does show a lot of differences, I guess the reason is, although different stemmers use different algorithms to stem the words, the difference won't be that huge, i.e. the difference between "service" and "serv" depending on the stemmer you are using. But if you do not use a stemmer, "burger" and "burgers" will be different words and won't be counted as a single word, this kind of "separate counting" might let some words with more lexical forms be in disadvantages. So the conclusion from the generated tables is: Use stemmers.

POS Tagging

2 methods/libraries: NLTK[1], spaCy[7]

Code Snippet:

```
#nltk
sents_5_onestring = list_to_string(sents_5)
text = nltk.word_tokenize(sents_5_onestring)
#words=[word.lower() for word in text if word.isalnum()]
result = nltk.pos_tag(text)
```

```
#spacy
doc = nlp(sents_5_onestring)
spacy_result={}
token_list=[]
for token in doc:
    #token.text.lower() not in token_list and
        spacy_result[token.text]= token.tag_
df_1 = pd.DataFrame.from_dict(spacy_result,orient='index',columns=['spaCy'])
```

Result:

	NLTK	spaCy
Service	NNP	NN
was	VBD	VBD
decent	JJ	JJ
,	,	,
prices	NNS	NNS
fair	VBP	JJ
food	NN	NN
...	:	.
Nothing	NNP	NN
special	JJ	JJ
.	.	.
Came	NN	VB
here	RB	RB
for	IN	IN
dinner	NN	NN
last	JJ	JJ
night	NN	NN
with	IN	IN

a	DT	DT
few	JJ	JJ
girlfriends	NNS	NNS
I	PRP	PRP
the	DT	DT
one	NN	CD
who	WP	WP
suggested	VBD	VBD
it	PRP	PRP
based	VBN	VBN
on	IN	IN
Yelp	NNP	NNP
reviews	NNS	NNS
The	DT	DT
just	RB	RB
very	RB	RB
bland	RB	JJ

Table 2.1: POS tagging result from 2 different methods

Discussion: Tagging results are almost the same except for a few. For "Nothing", NLTK recognizes it as NNP (proper noun), while spaCy recognize it as NN (noun). It is also interesting to see that for "Came", NLTK gives it a NN, while spaCy gives it a VB (verb). In this case, the tagging result of spaCy is more accurate since "Came here for..." which makes it a verb in actual analysis of the sentence. However, if only given "came", both should be correct tagging, although using "came" as a noun is uncommon. To sum up, the tagging result may vary if using different tagging tools due to its training model accuracy as well as the word shows different parts of speech in different sentences. More about spaCy[1] and NLTK[2] tagging result list.

Writing style

Posts Urls (2 for each):
StackOverflow:
https://stackoverflow.com/questions/69686312/find-the-employee-count-based-on-hire-date-in-oracle

https://stackoverflow.com/questions/263/gtk-implementation-of-messagebox

Hardwarezone :
https://forums.hardwarezone.com.sg/threads/have-you-figure-out-why-are-you-single.6624988/

https://forums.hardwarezone.com.sg/threads/air-conditioning-any-recommendations-please-part-2.5671374/

ChannelNewsAsia:
https://www.channelnewsasia.com/singapore/covid19-new-cases-deaths-singapore-moh-oct-22-2261741

https://www.channelnewsasia.com/asia/un-fears-mass-atrocity-crimes-myanmar-troops-gather-north-2263396

Analysis:

StackOverflow: One of the two posts from StackOverflow doesn't strictly follow the rule of capitalization of the first word in a sentence. Grammars of both of them are basically right. One of the posts contains proper nouns such as 'GTK', and they are correctly capitalized. Posts from StackOverflow are mostly written in the first person point of view. There are often tables and codes attached to them. Also, proper nouns related to computer and coding, as well as names of functions, interfaces, and classes of a programming language or toolkit frequently appear in sentences.

Hardwarezone : These two posts form Hardwarezone belong to Eat-Drink-Man-Women category and Home Decor & Furnishing category respectively. In one of these posts, first words in sentences are not capitalized, and the punctuations are also not correctly used. Most of the expressions in these posts are informal. Some proper nouns are not capitalized, for example, in the second post, company names such as Toshiba, Daikin, and Mitsubishi are all written in lower case. Some slang expressions and buzzwords from Internet are used in these posts.

ChannelNewsAsia: News articles form ChannelNewsAsia strictly follows the basic rules of written expression. For example, the first word in a sentence is capitalized; sentences follow good grammars; and proper nouns and names are capitalized. Besides, these articles also follow typical news-writing style. They are written in the third person point of view, thus words such as 'I' and 'you' hardly appear. On the contrary, numbers, names of people and places, and quotations of speeches are frequently seen.

Tokenization and POS tagging on posts: We try to directly apply usual tokenization and POS Tagging toolkits on posts from StackOverflow. As the two posts sampled from StackOverflow both contain table or code block, we firstly ignore these table or code block and only consider the parts in natural language. Then the two posts will be as below.

"I want to find out the employee count who are all joined between January month to December month for the year(2020). suppose if any employee is not joined on any particular month,the count should display as 0.

I tried below things.

Above query is not displaying count 0 for the months whose employee is not joined."

[1] https://github.com/explosion/spaCy/blob/master/spacy/glossary.py

[2] https://www.nltk.org/book/ch05.html

"I have been trying to implement Win32's MessageBox using GTK. The app uses SDL/OpenGL, so this isn't a GTK app.

I handle the initialization (gtk_init) sort of stuff inside the MessageBox function as follows:

Now, I am by no means an experienced GTK programmer, and I realize that I'm probably doing something horribly wrong.

However, my problem is that the last dialog popped up with this function staying around until the process exits. Any ideas?"

NLTK and SpaCy is used to apply tokenization and POS tagging on the above posts.

For the first post, the toolkit performs well. Figures below show part of the results of tokenization and POS tagging. The left one is the result by NLTK, and the right one is by SpaCy. Intuitively, the result by SpaCy is mostly correct.

```
('I', 'PRP')
('want', 'VBP')
('to', 'TO')
('find', 'VB')
('out', 'RP')
('the', 'DT')
('employee', 'NN')
('count', 'NN')
('who', 'WP')
('are', 'VBP')
('all', 'DT')
('joined', 'VBD')
('between', 'IN')
('January', 'NNP')
('month', 'NN')
('to', 'TO')
('December', 'NNP')
('month', 'NN')
('for', 'IN')
('the', 'DT')
('year', 'NN')
('(', '(')
('2020', 'CD')
(')', ')')
('.', '.')
```

```
('I', 'PRP')
('want', 'VBP')
('to', 'TO')
('find', 'VB')
('out', 'RP')
('the', 'DT')
('employee', 'NN')
('count', 'NN')
('who', 'WP')
('are', 'VBP')
('all', 'RB')
('joined', 'VBN')
('between', 'IN')
('January', 'NNP')
('month', 'NN')
('to', 'IN')
('December', 'NNP')
('month', 'NN')
('for', 'IN')
('the', 'DT')
('year(2020', 'NN')
(')', '-RRB-')
('.', '.')
```

SpaCy also performs well on the second post for tokenization and POS tagging. The second post is different from usual way of speaking because it contains lots of proper noun. SpaCy can deal with it well. The only exception is that "gtk_init" is tagged as "NN", which should also be "NNP".

```
('Win32', 'NNP')
('MessageBox', 'NNP')
('GTK', 'NNP')
('SDL', 'NNP')
('OpenGL', 'NNP')
('GTK', 'NNP')
('MessageBox', 'NNP')
('GTK', 'NNP')
```

The above cases consider only the contents excluding tables and code block. But a post from *StackOverflow* often includes tables or codes. If tables and code blocks are also taken into consideration, then the usual tokenization and POS tagging toolkits will be ineffective and meaningless.

Most Frequent < Noun, Adjective > pairs for Each Rating

Below are tables of top-10 most frequent < Noun, Adjective > pairs under each setting. Noted that each pair should appear in the same sentence in these selected reviews. Library used: spaCy[7]

Results:

	frequency
food-good	5
experience-worst	4
day-next	3
desk-front	3
time-first	3
food-great	2
service-worst	2
wear-normal	2
order-online	2
table-other	2

Table 2.1: 50 reviews (one from each business) of rating 1

	frequency
time-last	2
place-many	2
food-fast	2
haircut-great	2
language-different	1
stand-hostess	1
greasy-little	1
staff-friendly	1
chair-giant	1
chair-comfy	1

Table 2.2: 20 reviews (one from each business) of rating 2

	frequency
hour-happy	4
music-more	3
music-local	3
food-good	2
place-good	2
music-new	2
music-different	2
spot-good	2
experience-perfect	1
food-bad	1

Table 2.3: 20 reviews (one from each business) of rating 3

	frequency
breakfast-good	2
place-full	2
place-great	2
spot-great	2
food-good	2
line-long	2
cappuccino-best	1
day-other	1
birthday-happy	1
job-horrible	1

Table 2.4: 20 reviews (one from each business) of rating 4

	frequency
service-friendly	2
time-first	2
place-favorite	2
food-whole	1
experience-great	1
lesson-interesting	1
lesson-mini	1
people-outstanding	1
food-good	1
side-fresh	1

Table 5: 20 reviews (one from each business) of rating 4

Code snippet:

```
#main function of finding noun-adj pair frequency dictionary
def noun_adj_freqdict(s):
    noun_adj_dict = {}
    new_s = ""
    noun_list = []
    adj_list = []
    doc = nlp(s)
    for token in doc:
        if token.pos_ == 'ADJ':

            if token.dep_== 'amod':
                if token.head.pos_ =='NOUN':
                    noun_list = find_conj_noun(token.head, noun_list)
                adj_list = find_conj_adj(token, adj_list)
                if noun_list != None:
                    for i in noun_list:
                        for j in adj_list:
                            new_s = (i + "-" + j).lower()
                            dict_insertion(new_s,noun_adj_dict)
                noun_list=[]
                adj_list =[]

            if token.dep_ == 'acomp':
                adj_list = find_conj_adj(token, adj_list)
                for t in token.head.lefts:
                    if t.pos_ == 'NOUN':
                        noun_list = find_conj_noun(t, noun_list)
                        if noun_list != None:
                            for i in noun_list:
                                for j in adj_list:
                                    new_s = (i + "-" + j).lower()
                                    dict_insertion(new_s,noun_adj_dict)
                noun_list=[]
                adj_list =[]
    sorted_dict = {}
    sorted_keys = sorted(noun_adj_dict, key=noun_adj_dict.get, reverse=True)

    for w in sorted_keys:
        sorted_dict[w] = noun_adj_dict[w]
    return sorted_dict
```

Discussion: The following are some interesting problems that arise while implementing. Let us first see some example sentences:

1. This is an unreasonable and bad idea.
2. Expensive food and drink here are not so good.
3. Hot food and pretty flower are so nice.

We cannot simply pair the adjective and the noun up that shown in the sentence. One or more nouns and adjectives might appear in the same sentence. An adjective can describe one noun or more than a noun, while two or more adjective can also describe one noun. For the first sentence, we will have two pairs: idea-unreasonable, idea-bad. For the third sentence, we will have food-hot, flower-pretty, food nice, and flower-nice. We need to address these situations by finding the relation of the noun and adjective, or more formally called dependency.

More problems arise when finding the dependency. If we have 2 or more parallel adjective describing the same noun, the adjectives after the first one has dependency *conj* (conjunction), which head node is the previous adjective. For example, in 1, "unreasonable" has dependency *amod* (adjectival modifier) and head node "idea", while "bad" has dependency *conj* and head node "unreasonable". The same situation happens when there are 2 or more parallel nouns, where the second noun has dependency *conj*, which head node is the previous noun. Another problem is that the first adjective may have two types of dependencies, one is *acomp* (adjectival complement) and the other one is *amod.* Since dependency is parsed into a tree, the head node of *acomp* is not directly the noun as *amod*. For example, in the third sentence, the dependency of "hot" is *amod*, and the head node of "hot" is "food", while the dependency of "nice" is *acomp* and its head node is "are". More details of dependency please check *Stanford typed dependencies manual*[3].

After addressing all the problems above, we minimize the limitations of our method, and obtain our results as accurately as possible. The results are considerably accurate. From the result we could see that the frequency all pairs are smaller than five, and for pair frequency of 1, some of them are tied and should also be considered in the top-10 list. There are still limitations using this method:

1. Negative words are omitted. For example, in example sentence 2 above, after retrieving pairs, we obtain food-good and drink-good, while the semantics of the sentence gives the pair food-bad due to "not" is omitted.
2. Word might be in "wrong" tagging due to its multi-POS. For example, for sentence "She is a pretty, tall and attractive woman", "pretty" is tagged as an adverb. In this case, we miss the pair woman-pretty.
3. If the user mistype the word or forget to put whitespace after the punctuation or the text is in other language or putting emoji, it also affects the result accuracy.

[3] https://downloads.cs.stanford.edu/nlp/software/dependencies_manual.pdf

3. Development of a Simple Search Engine

In this part, we used the JAVA Apache Lucene API to design and implement the search engine. Each document contains fields which are *business_id, stars, date, text, useful, funny and cool*. Considering the reality, we decided to make the text field, business_id field and date field indexed and searchable. Text field has the review text, and it is the key information to the user. Business_id field would let the user search a specific restaurant if he or she knows the business_id. Date field would let the user search reviews on a specific date. All of three fields are indexed by documents, term frequencies and positions. Other fields such as stars, useful, funny and cool are stored and printed as additional information to the user. Such processes are shown in Figure3.1. We decided not to implement any parsing/linguistic processing such as stemming, case folding and stopword removal on the words in the searchable field. This is because if the search engine does such parsing/linguistic processing, the results would lose certain important information. Take an example of Porter Stemmer, the adjective words really and comfortable would become realli and comfort. It causes the review text to become meaningless and incomprehensible. Also, the results will not have any punctuation and become hard to read because of the tokenizer. Instead of processing document with stemming, case folding or stopword removal, we think it would be better to add certain parsing/linguistic processing to user's query. Our search engine would lowercase and remove the stopword for user's query. It generalizes user's query and returns more related and meaningful results.

Code Snippet:

```
//specify what is a document, and how its fields are indexed
protected Document getDocument(String review, String business_id, int stars, String date, int useful, int funny, int cool) throws Exception {
    Document doc = new Document();

    FieldType ft = new FieldType(TextField.TYPE_STORED);
    ft.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS);
    ft.setStoreTermVectors(true);

    doc.add(new Field("review", review, ft));
    doc.add(new Field("date", date, ft));
    doc.add(new Field("business_id", business_id, ft));
    doc.add (new StoredField ("stars", stars));
    doc.add(new StoredField("useful", useful));
    doc.add(new StoredField("funny", funny));
    doc.add(new StoredField("cool", cool));

    return doc;
}
```

Figure 3.1: Indexing Function of the Search Engine

```
//search for keywords in specified field, with the number of top results
public ScoreDoc[] search(String field, String keywords, int numHits) {
    //the query has to be analyzed the same way as the documents being index
    //using the same Analyzer
    QueryBuilder builder = new QueryBuilder(new StandardAnalyzer());
    Query query = builder.createBooleanQuery(field, keywords);
    Query phrasequery = builder.createPhraseQuery(field, keywords);

    ScoreDoc[] hits = null;
    try {
        //Create a TopScoreDocCollector
        TopScoreDocCollector collector = TopScoreDocCollector.create(numHits,300);
        if ((!keywords.contains("and") && !keywords.contains("or") && !keywords.contains("not")) && keywords.contains(" ")) {
            lSearcher.search(phrasequery, collector);
        }
        else {
            lSearcher.search(query, collector);
        }
        //collect results
        hits = collector.topDocs().scoreDocs;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return hits;
}
```

Figure 3.2: Search Function of the Search Engine

Our search engine was run on a 2015 MacBook Pro. The total indexing time is 1073.7 seconds which is about 18 minutes. Table 3.1 lists the processing time to index every 10% of the documents. It is shown that most processing times for every 10% are stable at 102-107 seconds. Nevertheless, the processing time at 60%-70% is much larger than others. This might be because it has more review text in this portion compared with other portions. Not like indexing, it took a short time which is 0.401 second to process a query and return the search results.

Processing Percentage of the document	Processing Time (Second)
0%-10%	105.903
10%-20%	102.438
20%-30%	104.578
30%-40%	104.695
40%-50%	105.691
50%-60%	107.153
60%-70%	117.664
70%-80%	108.893
80%-90%	109.135
90%-100%	107.584

Table 3.1: Processing time of Indexing the document

Our search engine supports free text keyword queries including single keyword query, Boolean query and phrase query. When processing a user's query, our search engine first determines if it contains any Boolean operator and if it is a single word. If it contains any Boolean operator or is a single word, our search engine would process it as a Boolean query. Otherwise, our search engine would process it as a phrase query. Figure3.2 shows the search function we used and the processes mentioned above. Besides ranking the search results by Lucene scores, our search engine can also rank the search results by stars, useful, funny and cool. This provides more convenience to the user. For example, if a user wants to see 5-star restaurants in New York, he or she may only need to search for "New York and choose to rank the results by star.

In the appendix, some sample queries and the Top-5 results returned by our search engine are shown. The words corresponding to the query are marked in yellow in the results. It can be seen that our search engine returned results as expected when the user inputs a Boolean query, a single keyword query or a phrase query. For Boolean queries, our search engine returns results containing the two terms even if they are separated in the review text. For phrase queries, keywords are treated as continuous and our search engine will search the whole phrase. For single keyword queries, our search engine successfully find review text containing the keyword. Basically, if we rank search results according to Lucene's score, we will rank according to term frequency. Furthermore, if the user searches by business_id, our search engine will return the reviews of the same restaurant. If the user searches by date, our search engine will return the review on that specific date. In conclusion, our search engine successfully implements Boolean queries, single keyword queries and phrase queries and provides extra functions.

4. Extraction of Indicative Adjective Phrases

In this part, we use slightly modified $tf.idf$ to reflect if an adjective phrase is the unique characteristics of a business.

Term frequency: Term frequency $tf(t,d)$ is the frequency of the adjective phrase t,

$$tf(t,d) = \frac{f_{t,d}}{n}$$

where $f_{t,d}$ is the number of reviews containing t in a selected business d. n is the total number of reviews in the selected business d.

Inverse document frequency:

$$idf(t,D) = \log \frac{N}{|\{d \in D: t \in d\}|}$$

inverse document frequency shows that whether it's rare or common in all reviews. N is the total number of reviews. $|\{d \in D: t \in d\}|$ means the number of reviews containing adjective phrase t. idf(t, D) is the logarithmically scaled inverse fraction of the reviews which contain the adjective phrase t (obtained by dividing the total number of reviews by the number of reviews containing t, and then taking the logarithm of that quotient).

idf.tf:

$$tf.idf(t,d,D) = tf(t,d) \times idf(t,D)$$

A high value of $tf.idf$ is reached by a high frequency of adjective phrases in the selected business and a low frequency of adjective phrases in the whole collection of reviews; Since common phrases lead to low values of $idf(t,D)$, $tf.idf$ tend to filter out common adjective phrases such as good, great. The ratio inside the logarithm is always greater than or equal to 1, so the value of $tf.idf$ is always greater than or equal to 0. As a phrase appears in more reviews, the ratio inside the log function is close to 1, bringing the idf and $tf.idf$ closer to 0.

Process of extraction:

```
chunkGram = r"""Chunk: {<RB.*>*<JJ.*>}"""
chunkParser = nltk.RegexpParser(chunkGram)
```

Figure 4.1: regular expressions are used to extract phrases.

We perform tokenization and POS tagging(use NLTK package) to each review. Then adjective phrases are filtered out by regular expressions and their frequencies are counted. After we get the frequency of phrases, $tf(t,d)$, 10000 reviews were randomly selected from all the reviews. Through the process of tokenization and POS tagging, we get the frequency of adjective phrases in all review.

```
tf = str[i][1]/len(reviews)
idf = math.log10(len(reviews_random)/(str2[str[i][0]]+0.1))
indicate_adj[str[i][0]]=tf*idf
```

Figure 4.2: calculate the values of $tf.idf(t,d,D)$ of adjective phrases.

Finally, we calculate the values of $tf.idf(t,d,D)$ of adjective phrases and sort them.

Results: Take the business b with 70 reviews whose $business\_id$ is $RA4V8pr014UyUbDvI-LW2A$ for example, the following figures show the results of our program.

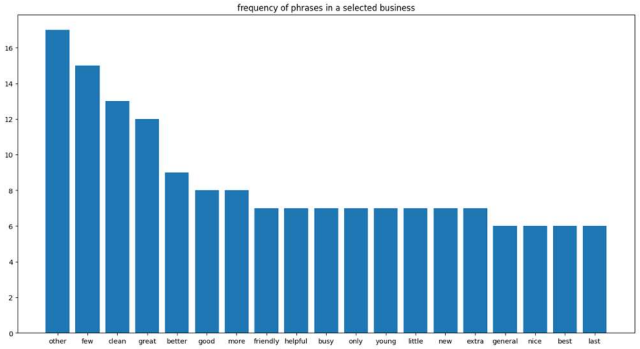


Figure 4.3: frequencies of phrases in the selected business b(top 20)

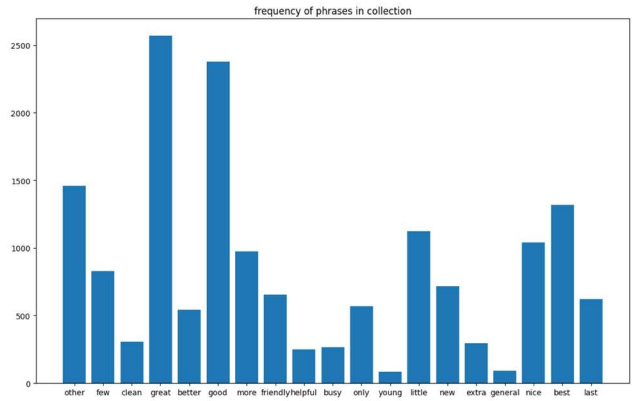


Figure 4.4: frequencies of phrases in the 10000 reviews chosen randomly

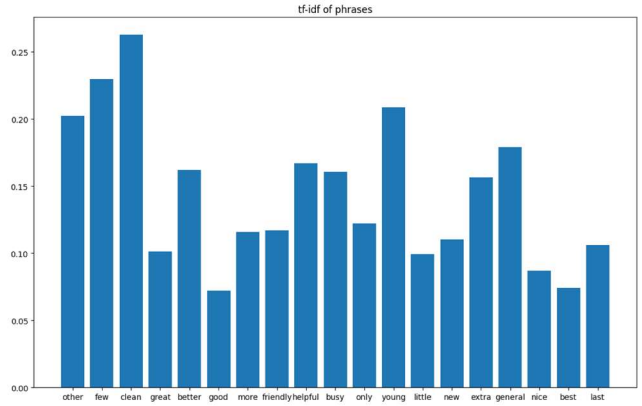


Figure 4.5: $tf.idf$ of phrases

Figure 4.3 indicates frequencies of phrases in the selected business b and figure 4.4 shows frequencies of phrases in the reviews selected randomly. As can be seen from figure 4.3, the four words, other, few, clean and great, appear most frequently in reviews of business b (more than 10 times in 70 reviews). While as figure 4.4 shows, the word, great, appears most frequently in the whole collection, the words, few and other, come second and third, and clean is the lowest. Though the word, other, has the highest tf

value, its low idf value makes its value of $tf.idf$ lower than value of 'clean', which means it is common in all reviews. Because of the high values of tf and idf, the $tf.idf$ value of clean is the highest as figure 4.5 shows. That means it is the indicative adjective phrase to represent business b.

Manually going through all reviews of b, we can find that the adjective phrase, clean, appears in 13 of 70 reviews and the phrase, great, appears in 12 of 70 reviews. Going through reviews of another business whose $business\_id$ is $hwispCvACnVUk6-0thwLmQ$, we can see that the adjective phrase, clean, appears in only 2 of 169 reviews while the phrase, great, appears at least 90 of 169 reviews. What the manual result reflects is the same as what the result of program reflects and therefore the adjective phrase, clean, can reflect the unique characteristics of business b to some extent.

However, randomly choosing some other $business\_id$ to run the program, we find that there is a certain probability to extract common phrases such as good, great, many, because the value of tf weights more than the value of idf in $tf.idf$.

Limitations: In some cases, the formulation, $tf.idf$, which is used in our method cannot filter out some common phrases such as good, great because of their high frequencies in the reviews of selected business.

We find some limitations when manually go through the reviews. One of the limitations of our method is that some implicit negation expressions cannot be detected. For example, the review, 'many people told me it was clean, but I don't think so.', means that the reviewer thinks the business is not clean. But using regex, we can get the adjective phrase, clean, from this review and regard it as one of the characteristics of the business, which is contrary to what the reviewer means.

Another limitation is that our method cannot judge if the adjective phrases are used to describe the business or not. For example, the word, clean, in the review, 'I was contemplating licking it clean', is not used to describe the business but the program cannot judge it. In general, our method just extracts the key words simply and it is hard to extract the unique characteristics of a business on the semantic level because of many complex cases.

5. Application

Sentiment Analysis:

In this section, we try to implement a command line application that, input is a given business(given business_id), output a score based on all the comments for that particular business that marks the overall "sentiment" of all reviewers.

Code Snippet:

```python
from textblob import TextBlob
import json,random


def r_dataset(filename, business_id):
    with open(filename,'r') as f:
        lines = f.readlines()
    n_lines = len(lines)
    print(f"> Success! There are in total:{n_lines} lines")
    business_id = json.loads(lines[random.randint(0, n_lines-1)])['business_id']
    # random_business = "PEnMU_He_qHoCfdoAKmjDQ" # hardcoding business
    print(f"> Success! The randomly chosen business_id is {business_id}")
    all_data = [json.loads(line) for line in lines]
    selected_business_reviews = \
    [one_data['text'] for one_data in all_data if one_data['business_id'] == business_id]
    print(f"> Success! The application has already processed {len(selected_business_reviews)} lines of comments")
    return selected_business_reviews


def sentiment_analysis(all_data):
    subjective_sentiment = []
    for line in all_data:
        tb = TextBlob(line)
        if tb.sentiment.subjectivity >= 0.6:
            subjective_sentiment.append(tb.sentiment.polarity)
    return sum(subjective_sentiment)

if __name__ == "__main__":
    all_review_data = r_dataset("review.json","PEnMU_He_qHoCfdoAKmjDQ")
    sub = sentiment_analysis(all_review_data)
    print(f"> Subjective sentiment score:{sub}")
```

We use the same function as previous steps to read all review data for a particular business. Then we use TextBlob[4] to parse and analyze the comment text. We filter all the comments with subjectivity (0: totally objective;1: totally subjective) higher than 0.6, i.e. we filter all comments that actually show customers' emotions, and sum them up, since the range of polarity is from -1 to 1, -1 marks totally negative, 1 marks totally positive. The final score would be representative enough to show the overall sentiment for that particular business.

Our idea:

This application would be equipped with ranking system to build an application similar to Yelp, but our ideal application is better in a way that the star rating is not based on user review but based on sentiment. Let's imagine a situation where customers all comment reviews like "very good" with star rating very low, then the overall star rating for that particular business would be very low with sentiment of all comments very high. Although that situation might not happen, I am saying that we don't need to relay on the "consistency of people" to make sure star rating actually reflect whether the restaurant is good or bad, we actually don't need customers to do star rating at all. Moreover, comments need thinking and phrasing a sentence, from the practical perspective, it's more reliable than the star rating. Therefore, I could make the claim that our method would be better in reflecting the true rating of a restaurant.

Our discoveries:

1. There are far more restaurants with high score than restaurants with low score, i.e., lower than 0,

```
zwx@MacBookPro Assignment % python3 sentan.py
> Success! There are in total:1000000 lines
> Success! The randomly chosen business_id is eXkbL-ZTC1Q74uNi9iBa0g
99
subjective sentiment score:-2.0650836489898987
zwx@MacBookPro Assignment %
```

2. Business with more comments tend to have higher score, which means bias. Customers who have bad experience are more likely to keep silent and not make any comments (not responding bias), therefore we can only sample positive feedback (selection bias)

[4] https://spacy.io/universe/project/spacy-textblob

```
zwx@MacBookPro Assignment % python3 sentan.py
> Success! There are in total:1000000 lines
> Success! The randomly chosen business_id is 5av9ao6Ygib4al_0-1FXuA
> Success! The application has already processed 111 lines of comments
> Subjective sentiment score:5.548708531293341
zwx@MacBookPro Assignment % python3 sentan.py
> Success! There are in total:1000000 lines
> Success! The randomly chosen business_id is SHJuAVPchwPlBH3ePsCiIA
> Success! The application has already processed 368 lines of comments
> Subjective sentiment score:63.25949404973632
zwx@MacBookPro Assignment % python3 sentan.py
> Success! There are in total:1000000 lines
> Success! The randomly chosen business_id is SP94WHsL3Btm8ucfxpzJzw
> Success! The application has already processed 33 lines of comments
> Subjective sentiment score:3.861170226638297
zwx@MacBookPro Assignment % python3 sentan.py
> Success! There are in total:1000000 lines
> Success! The randomly chosen business_id is Yd9vg2PYi1ZGyaSiMgroiw
> Success! The application has already processed 165 lines of comments
> Subjective sentiment score:18.692584144279067
zwx@MacBookPro Assignment %
```

APPENDIX

Sample Queries and Results for 3

1. Boolean query(rank by *stars*)

Query: COLumbus and STEAK
==========Results for review text search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
1

Result 1 DocID: 3657788 Score: 6.6936636
Review Text: Absolute best steak house in Columbus. Isn't afraid to make a blue rare steak either.
Review Date: 2018-10-18 22:19:38 Review Business_id: MyuVJzBb0WYWUU2bOBBDrg
Stars: 5; Useful: 0; Funny: 0; Cool: 0

Result 2 DocID: 1798840 Score: 6.5219045
Review Text: Best food in Columbus. Very innovative - interesting flavors. Never disappointed. The steak tartar was amazing (sadly no longer on the menu). Rarely bother going anywhere else in Columbus.
Review Date: 2017-01-11 01:37:40 Review Business_id: FOYFzi_IUFpVHtrDXWd-Aw
Stars: 5; Useful: 2; Funny: 0; Cool: 1

Result 3 DocID: 7920961 Score: 6.400753
Review Text: Just as it was in my childhood, York Steak House produces a great steak at a great price. Only location left is in Columbus, Ohio.
Review Date: 2017-04-07 22:46:03 Review Business_id: wxMe_Jktla7T5ka21cwCwg
Stars: 5; Useful: 0; Funny: 0; Cool: 0

Result 4 DocID: 1036826 Score: 6.665456
Review Text: One of my favorite steak house in Columbus Ohio. Make sure that you make a reservation first before you dine in this restaurand. The food is excellent and the place is nice as well. Definitely recommended for someone looking for a good place to eat steak in Columbus Ohio.
Review Date: 2009-03-29 06:28:20 Review Business_id: hqx382sZ_5WHK8W5D4ATGQ
Stars: 4; Useful: 0; Funny: 0; Cool: 0

Result 5 DocID: 6372350 Score: 6.475895
Review Text: It was good. Got a steak sandwich and enjoyed it. We went there cause it had the top ranking in Columbus. Very good and there are places in Columbus that can match it.
Review Date: 2015-03-21 23:34:25 Review Business_id: 2AMrPSOA7VzjPZmqvZOttw
Stars: 4; Useful: 0; Funny: 0; Cool: 0

==========Results for review date search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
0

==========Results for review business_id search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
0

2. Phrase query(rank by *stars*)

Query: delicious chicken
==========Results for review text search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
1

Result 1 DocID: 8257683 Score: 3.4824352
Review Text: Really really good Indian food. Chicken tikka masala was so delicious! Chicken kabob was also delicious. Chicken biryani was really spicy, but very good too. The restaurant is clean and service is excellent. Will definitely be coming back!
Review Date: 2018-09-08 21:56:49 Review Business_id: LpgUGT2u7TcP8mkUh-oLmw
Stars: 5; Useful: 0; Funny: 0; Cool: 0

Result 2 DocID: 2742920 Score: 3.35825
Review Text: If you want delicious chicken, The chicken wings at chinking are the best I have ever eaten. They are so tasty. Quick service and delicious chicken you have to try them out. Watch out all the other fast food restaurants serving chicken. This company really knows how to make it! Delicious.
Review Date: 2020-11-22 23:47:16 Review Business_id: v_ZaLP0e7NGfFnkRFdbX0A
Stars: 5; Useful: 1; Funny: 0; Cool: 0

Result 3 DocID: 952079 Score: 3.3446283
Review Text: delicious chicken kabobs !! juicy tender flavorful chicken!
Review Date: 2015-04-05 00:06:10 Review Business_id: yPjQBv6iyqMjZFnmRZWEQw
Stars: 5; Useful: 0; Funny: 0; Cool: 0

Result 4 DocID: 2629637 Score: 3.3446283
Review Text: Delicious chicken pad thai and fast service!
Review Date: 2020-01-02 01:37:10 Review Business_id: 9i-jlmd8VYg0kam86Fqe0A
Stars: 5; Useful: 0; Funny: 0; Cool: 0

Result 5 DocID: 4429409 Score: 3.3249469
Review Text: Consistently delicious chicken burritos. Fast and friendly service.
Review Date: 2015-03-10 16:40:51 Review Business_id: AtJFEVY58jBUU5KLRsAOUA
Stars: 4; Useful: 0; Funny: 1; Cool: 0

==========Results for review date search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
0
==========Results for review business_id search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
0

3. Single keyword query(rank by Lucene scores)

Query: APPEALING
==========Results for review text search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
0

Result 1 DocID: 5614318 Score: 4.729924
Review Text: Cool atmosphere, but not kid friendly. Didn't find the menu appealing, but when my duck (yeah, it was the most appealing thing) wow! Flavor was terrific. They also accommodated out challenging diets. I'll go back, but not with the kids.
Review Date: 2014-06-04 23:29:05 Review Business_id: Ib2l9D_fT-zjgBvhrCRTkQ
Stars: 3; Useful: 0; Funny: 0; Cool: 0

Result 2 DocID: 2229903 Score: 4.7011275
Review Text: You need to be wary of ordering takeout. They cut the shrimp in half lengthwise and fill the container with rice and frozen veggies from a bag. They proceed to overcharge for that. How appealing Is that? It ain't ten dollars appealing!
Review Date: 2018-08-12 04:49:26 Review Business_id: DAvLtqzuCFGtY02zR7OlwQ
Stars: 1; Useful: 0; Funny: 0; Cool: 0

Result 3 DocID: 2234633 Score: 4.67268
Review Text: We hired Appealing Curb Design when we moved into our new home. They were amazing installing our new sprinkler system! My family is very pleased with our beautiful new yard and we thank Appealing Curb Design for being the best landscaping company in Austin, TX!
Review Date: 2010-08-09 02:32:02 Review Business_id: -ktTrP_SBwCcPzQkeV73Mw
Stars: 5; Useful: 2; Funny: 0; Cool: 0

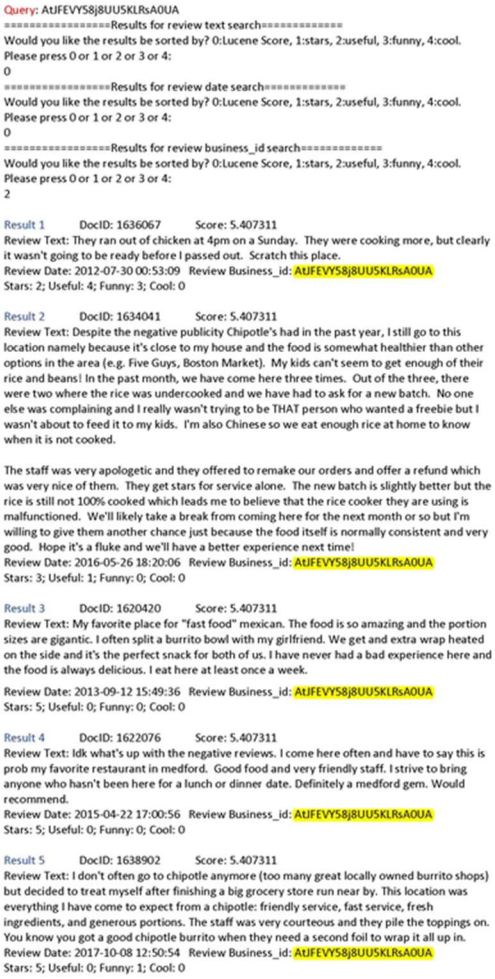
Result 4 DocID: 854498 Score: 4.508969
Review Text: My first night at Oregon; we went to the Habibi Restaurant. I have a lamb shawarma pasta and my friend has lamb shawarma plate with rice. The foods are good but it doesn't look appealing on the plate. The owner need to decorate the dishes so it would be more appealing. The habibi martini is horrible.
Review Date: 2014-09-20 17:41:11 Review Business_id: NIQ0ox3AMnQvjmyTbroH9Q
Stars: 4; Useful: 0; Funny: 0; Cool: 0

Result 5 DocID: 2501576 Score: 4.497459
Review Text: Overall: maybe this place's strength is dinner, but I was not impressed with its brunch. While the menu had a lot of appealing options, I didn't think they were executed well and service was slow.

Likes:
- ambiance is trendy with unique lighting
- appealing menu with good sweet and savory options
- chorizo omelet was pretty good (however ingredients weren't very plentiful), chorizo had pretty good flavor. Would've loved to see more kale and chorizo in the omelet
- potatoes were perfectly cooked with a little crisp on the outside

Gripes:
- slow service and servers with an attitude
- Dutch baby came out partially burnt and less than appealing looking
Review Date: 2014-04-05 17:22:53 Review Business_id: j7eMmgmlS-ROU-22w3VLFw
Stars: 3; Useful: 0; Funny: 0; Cool: 0
==========Results for review date search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
0
==========Results for review business_id search==========
Would you like the results be sorted by? 0:Lucene Score, 1:stars, 2:useful, 3:funny, 4:cool.
Please press 0 or 1 or 2 or 3 or 4:
0

4. Search the business id(rank by *useful*)



REFERENCES

[1] NLTK :: nltk package. (n.d.). NLTK. Retrieved October 23, 2021, from https://www.nltk.org/api/nltk.html

[2] The Porter Stemming Algorithm. (n.d.). Tartarus.Org. Retrieved October 23, 2021, from https://tartarus.org/martin/PorterStemmer/

[3] GeeksforGeeks. (2020, October 15). Snowball Stemmer - NLP. Retrieved October 23, 2021, from https://www.geeksforgeeks.org/snowball-stemmer-nlp/

[4] Bernardo, I. (2021, August 14). Stemming Text with NLTK - Towards Data Science. Medium. Retrieved October 23, 2021, from https://towardsdatascience.com/stemming-corpus-with-nltk-7a6a6d02d3e5#:%7E:text=Lancaster%20Stemmer%20is%20a%20stemmer,overstem%20a%20lot%20of%20words.

[5] Apache Lucene :: open-source search software. (n.d.). Retrieved October 23, 2021, from https://lucene.apache.org/

[6] Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.