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.

Name	Signature / Date
LUO HAO G2202279H	Luo Hao
YU YUE G2202151A	Yu Yue
WANG YANGHAO G2202518G	Wang Yanghao
LI ZENONG G2202242B	Li Zenong.
XU XIANGYU G2202527H	Xu Xiangyu

Important note:

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

AI6122 Review Data Analysis and Processing

Li Zenong
lize0011@e.ntu.edu.sg
Nanyang Technological University
Singapore

Luo Hao
luoh0013@e.ntu.edu.sg
Nanyang Technological University
Singapore

Wang Yanghao
ywang150@e.ntu.edu.sg
Nanyang Technological University
Singapore

Xu Xiangyu
xxu034@e.ntu.edu.sg
Nanyang Technological University
Singapore

Yu Yue
yyu025@e.ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

In the assignment, we implement the whole pipeline of text preprocessing and management technologies on the Amazon product dataset [8], such as tokenization, sentence segmentation, stemming and POS tagging based on NLTK [10] and spaCy [2]. A simple search engine is built for indexing and query searching with Lucene [3]. A summarizer based on TextRank algorithm is implemented for summarize all reviews for a certain product. And some other interesting applications such as word cloud based on the dataset is also achieved. All the code can be downloaded from github.

1 DATASET ANALYSIS

1.1 Dataset Downloading and Sampling

We directly download our dataset from Amazon product data collected by University of California San Diego. We randomly select **Patio, Lawn and Garden** and **Musical Instruments** these two dataset from 24 datasets that have been preprocessed with 5-core filtering. Then, we randomly sample 200 products from each dataset by *random.sample* function, and keep all the reviews of these 200 products in each dataset. All the subsequent analysis will be conducted on these two sampled datasets, each with reviews of 200 products. Table 1 shows the statistics of the two sampled datasets.

1.2 Writing Style

In this task, we randomly select a few reviews and observe the writing style. The following three sentences are randomly picked from the *reviewText* of *Patio, Lawn and Garden*:

(1) super easy to swap out and last a long time, unless you are chopping concrete! I use these all the time, as I feel they are worth the extra cost for the convenience.
(2) I used this over the summer and it never tangled. It is a lot heavier than my other cheap hoses. I will be buying more of these.
(3) his is a must have for any one who grills or smokes with charcoal. Ball up some newspaper and shove in the bottom, fill with coals, light and let sit for 10 minutes or so, your coals will be perfect. I have had 2 of these for years and they work great.

The following three sentences are randomly picked from the *reviewText* of *Musical Instruments*:

	Patio, Lawn and Garden	Musical Instruments
data num	2543	2050
product num	200	200
avg	12.715	10.250

Table 1: sampled datasets' statistics

(1) It came in nice packaging.Cable looks great - feels of high quality.So far it still works well.10 ft is nice for home.If you're a performer... longer is required.
(2) Believe it or not, I've yet to use this screen. I was short a shock mount for one of my Cad condensers. I've yet to use that mic again. Go figure. But I love the way screen and mount look? I know. So esoteric. But I'm still giving it 5 stars. Thanks again Amazon.
(3) This works great with the Mustang I amp. Switches between to preset channels with just a tap of the foot. A great addition to an already great amp. At this price it's kinda like the amp itself- a no brainer!

As we can see from the selected reviews, the first word in a sentence is not always capitalized ((1) in *Patio, Lawn and Garden*), but most of the reviewText start with the capital letter, I guess it is because automatic capitalization function of input method in the mobile phones. The sentences do not follow really good grammars, like "feels of high quality", the subject is omitted, and most text does not follow strict punctuation and use of spaces, and random interrupts often occur. The proper nouns sometimes are not capitalized such as "mic" and "amp" in "Mustang I amp". Besides, there are many spelling errors or abbreviations such as "mic" for "microphone".

The following paragraph comes from the article *Wireless microphone best in market* [9] published in The Straits Times:

- I use a wired lavalier microphone, but still have to shout. Not to mention, the short cable of the microphone means the smartphone has to be near my face, which is not too flattering.

Compared with reviews, the sentences in The Straits Times strictly comply with the requirements of English grammar, punctuation and the application of sentence, such as complete spelling "microphone". Meanwhile, the statement expression is more advanced, such as "still have to shout" to express the complain. In short, sentences in reviews are more casual, more in line with the habits of colloquial and everyday communication.

1.3 POS Tagging

We randomly select 5 sentences from the dataset and apply POS tagging. The results are shown in the following:

(1) (Sentence 1) I've used this in my jam lab for 4 years and use it every time I play electric (about 5-7 hours a week.
- (NLTK POS) [('I', 'PRP'), ("'ve", 'VBP'), ('used', 'VBN'), ('this', 'DT'), ('in', 'IN'), ('my', 'PRP$'), ('jam', 'NN'), ('lab', 'NN'), ('for', 'IN'), ('4', 'CD'), ('years', 'NNS'), ('and', 'CC'), ('use', 'VB'), ('it', 'PRP'), ('every', 'DT'), ('time', 'NN'), ('I', 'PRP'), ('play', 'VBP'), ('electric', 'JJ'), ('(', '('), ('about', 'IN'), ('5-7', 'JJ'), ('hours', 'NNS'), ('a', 'DT'), ('week', 'NN'), ('.', '.')]
- (spaCy POS) [('I', 'PRON'), ("'ve", 'AUX'), ('used', 'VERB'), ('this', 'PRON'), ('in', 'ADP'), ('my', 'PRON'), ('jam', 'NOUN'), ('lab', 'NOUN'), ('for', 'ADP'), ('4', 'NUM'), ('years', 'NOUN'), ('and', 'CCONJ'), ('use', 'VERB'), ('it', 'PRON'), ('every', 'DET'), ('time', 'NOUN'), ('I', 'PRON'), ('play', 'VERB'), ('electric', 'ADJ'), ('(', 'PUNCT'), ('about', 'ADV'), ('5', 'NUM'), ('-', 'SYM'), ('7', 'NUM'), ('hours', 'NOUN'), ('a', 'DET'), ('week', 'NOUN'), ('.', 'PUNCT')]

(2) (Sentence 2) It doesn't have a on/off switch but that's minor.
- (NLTK POS) [('It', 'PRP'), ('does', 'VBZ'), ("n't", 'RB'), ('have', 'VB'), ('a', 'DT'), ('on/off', 'NN'), ('switch', 'NN'), ('but', 'CC'), ('that', 'DT'), ("'s", 'VBZ'), ('minor', 'JJ'), ('.', '.')]
- (spaCy POS) [('It', 'PRON'), ('does', 'AUX'), ("n't", 'PART'), ('have', 'VERB'), ('a', 'DET'), ('on', 'ADP'), ('/', 'SYM'), ('off', 'ADJ'), ('switch', 'NOUN'), ('but', 'CCONJ'), ('that', 'PRON'), ("'s", 'AUX'), ('minor', 'ADJ'), ('.', 'PUNCT')]

(3) (Sentence 3) I put in it two areas where I either knew or suspected that there were ants inside my house.
- (NLTK POS) [('I', 'PRP'), ('put', 'VBP'), ('in', 'IN'), ('it', 'PRP'), ('two', 'CD'), ('areas', 'NNS'), ('where', 'WRB'), ('I', 'PRP'), ('either', 'DT'), ('knew', 'VBD'), ('or', 'CC'), ('suspected', 'VBD'), ('that', 'IN'), ('there', 'EX'), ('were', 'VBD'), ('ants', 'NNS'), ('inside', 'IN'), ('my', 'PRP$'), ('house', 'NN'), ('.', '.')]
- (spaCy POS) [('I', 'PRON'), ('put', 'VERB'), ('in', 'ADP'), ('it', 'PRON'), ('two', 'NUM'), ('areas', 'NOUN'), ('where', 'SCONJ'), ('I', 'PRON'), ('either', 'CCONJ'), ('knew', 'VERB'), ('or', 'CCONJ'), ('suspected', 'VERB'), ('that', 'SCONJ'), ('there', 'PRON'), ('were', 'VERB'), ('ants', 'NOUN'), ('inside', 'ADP'), ('my', 'PRON'), ('house', 'NOUN'), ('.', 'PUNCT')]

(4) (Sentence 4) I have it going into my mixer which works fine.
- (NLTK POS) [('I', 'PRP'), ('have', 'VBP'), ('it', 'PRP'), ('going', 'VBG'), ('into', 'IN'), ('my', 'PRP$'), ('mixer', 'NN'), ('which', 'WDT'), ('works', 'VBZ'), ('fine', 'NN'), ('.', '.')]
- (spaCy POS) [('I', 'PRON'), ('have', 'AUX'), ('it', 'PRON'), ('going', 'VERB'), ('into', 'ADP'), ('my', 'PRON'), ('mixer', 'NOUN'), ('which', 'PRON'), ('works', 'VERB'), ('fine', 'ADV'), ('.', 'PUNCT')]

(5) (Sentence 5) Cubase is a little quirky and will often tell me that there is an error and it needs to close, but I've learned nothing happens and it's fine.I don't use the looper much as I have it before my Digitech Jam Man.
- (NLTK POS) [('Cubase', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('little', 'JJ'), ('quirky', 'NN'), ('and', 'CC'), ('will', 'MD'), ('often', 'RB'), ('tell', 'VB'), ('me', 'PRP'), ('that', 'IN'), ('there', 'EX'), ('is', 'VBZ'), ('an', 'DT'), ('error', 'NN'), ('and', 'CC'), ('it', 'PRP'), ('needs', 'VBZ'), ('to', 'TO'), ('close', 'VB'), (',', ','), ('but', 'CC'), ('I', 'PRP'), ("'ve", 'VBP'), ('learned', 'VBN'), ('nothing', 'NN'), ('happens', 'NNS'), ('and', 'CC'), ('it', 'PRP'), ("'s", 'VBZ'), ('fine.I', 'JJ'), ('do', 'VBP'), ("n't", 'RB'), ('use', 'VB'), ('the', 'DT'), ('looper', 'NN'), ('much', 'RB'), ('as', 'IN'), ('I', 'PRP'), ('have', 'VBP'), ('it', 'PRP'), ('before', 'IN'), ('my', 'PRP$'), ('Digitech', 'NNP'), ('Jam', 'NNP'), ('Man', 'NNP'), ('.', '.')]
- (spaCy POS) [('Cubase', 'NOUN'), ('is', 'AUX'), ('a', 'DET'), ('little', 'ADJ'), ('quirky', 'ADJ'), ('and', 'CCONJ'), ('will', 'AUX'), ('often', 'ADV'), ('tell', 'VERB'), ('me', 'PRON'), ('that', 'SCONJ'), ('there', 'PRON'), ('is', 'VERB'), ('an', 'DET'), ('error', 'NOUN'), ('and', 'CCONJ'), ('it', 'PRON'), ('needs', 'VERB'), ('to', 'PART'), ('close', 'VERB'), (',', 'PUNCT'), ('but', 'CCONJ'), ('I', 'PRON'), ("'ve", 'AUX'), ('learned', 'VERB'), ('nothing', 'PRON'), ('happens', 'VERB'), ('and', 'CCONJ'), ('it', 'PRON'), ("'s", 'AUX'), ('fine', 'ADJ'), ('.', 'PUNCT'), ('I', 'PRON'), ('do', 'AUX'), ("n't", 'PART'), ('use', 'VERB'), ('the', 'DET'), ('looper', 'NOUN'), ('much', 'ADV'), ('as', 'SCONJ'), ('I', 'PRON'), ('have', 'VERB'), ('it', 'PRON'), ('before', 'ADP'), ('my', 'PRON'), ('Digitech', 'PROPN'), ('Jam', 'PROPN'), ('Man', 'PROPN'), ('.', 'PUNCT')]

As we can see from the results, both NLTK and spaCy packages can do POS tagging with relatively high accuracy, especially those tokens that occur frequently, such as "learned", "before", "my" and so on. Besides, for those proper nouns, such as "Digitech", "Jam", "Man", spaCy is relatively better than NLTK, the later one labels them as "NNP" but spaCy labels them as "PROPN", which may caused by the POS tagging labels used by the two packages are different.

1.4 Sentence Segmentation

We also perform sentence segmentation by using NLTK on the reviews and compare the distribution of the two datasets in a single plot. The figure 1 shows the results. As we can see, in both two datasets, the length of a review in number of sentences are clustered in the range from 1 to 20, and the highest points are around 5 sentences. Few reviews are more than 40 sentences long.

For the dataset *Musical Instruments*, reviews that are about 3 sentences are the most, and the number of reviews that are more than 3 sentences decrease sharply as the number of sentences increases. For the dataset *Patio, Lawn and Garden*, reviews that are also about 3 sentences are the most, and the number of reviews that are more than 3 sentences also decrease fast as the number of sentences increases, but the decline is a bit more moderate than in the dataset *Musical Instruments*, so the number of sentences is more widely distributed.

1.5 Tokenization and Stemming

We then tokenize the reviews by using NLTK. Similarlt, we compare the review length distribution in terms of number of tokens for the two datasets. The figure 2 shows the results. As we can see, in

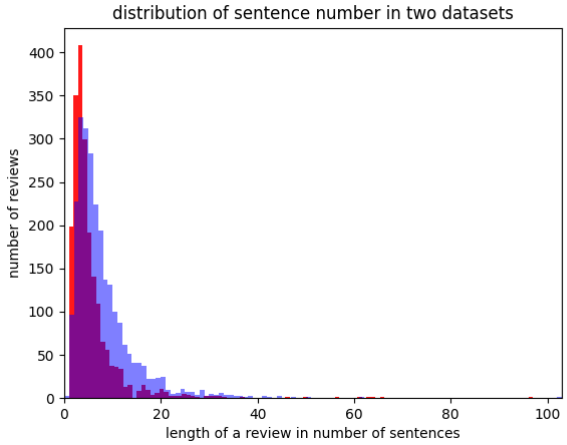

Figure 1: Sentence number distribution of the two datasets. The red one represents *Musical Instruments* dataset and the blue one represents *Patio, Lawn and Garden* dataset. The x-axis is the length of a review in number of sentences, and the y-axis is the number of reviews of each length.

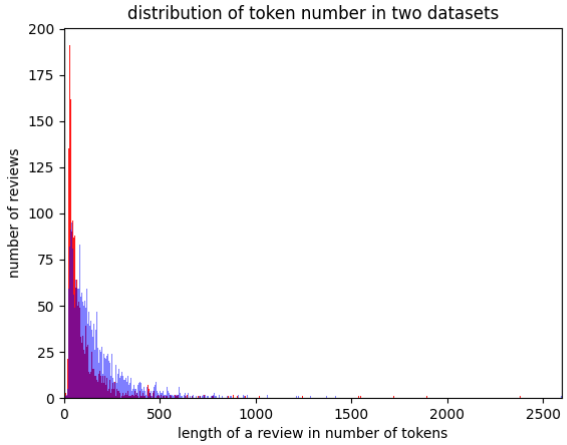

Figure 2: Token number distribution of the two datasets. The red one represents *Musical Instruments* dataset and the blue one represents *Patio, Lawn and Garden* dataset. The x-axis is the length of a review defined by the number of tokens, and the y-axis is the number of reviews having that length.

both two datasets, the length of a review in number of tokens are clustered in the range from 1 to 500. Few reviews are more than 1000 sentences long.

For the dataset *Musical Instruments*, most of reviews are about 20 tokens, and the number of reviews that are more than 20 tokens decrease sharply as the number of tokens increases. For the dataset *Patio, Lawn and Garden*, reviews that are about 25 to 30 tokens are the most, and the number of reviews that are more than 30 tokens also decrease fast as the number of tokens increases, but the decline is a bit more moderate than in the dataset *Musical Instruments*, so the number of tokens is more widely distributed. The observation consistent with the sentence number distribution, that's because the number of tokens and the number of sentences are statistically proportional to each other. In contrast, the average number of tokens in reviews of *Musical Instruments* is more than that in *Patio, Lawn and Garden*.

Then, at the dataset level, we show two distributions to observe the impact of stemming. We implemented PorterStemmer stemming, SnowballStemmer stemming in NLTK and spacy default lemmatization algorithms to perform token stemming, but to briefly describe the impact of stemming on the token, we select the results obtained by PorterStemmer in NLTK.

The figure 3 shows the number of tokens in each datasets before and after Porter stemming. As we can see from the figure, the number of tokens are obviously decreased after stemming: specifically, for the dataset *Musical Instruments*, the number of tokens are decreased from 13837 to 9189 (-33.59%), for the dataset *Patio, Lawn and Garden*, the number of tokens are decreased from 21185 to 14071 (-33.58%). For both datasets, the percentage amount of token reduction is close. This shows that Porter stemming can significantly reduce the number of tokens.

Then, we focus on the specific tokens in each dataset. Because the number of tokens are too large, here we only shows the top

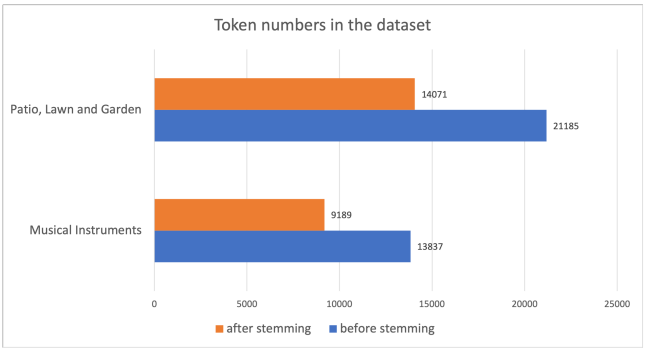

Figure 3: Token numbers in the two datasets before/after stemming.

50 tokens with the most frequency in each dataset. The figure 4 shows top 50 frequent tokens and their occur numbers in the two datasets before stemming, and the figure 5 shows top 50 frequent tokens and their occur numbers in the two datasets after stemming. As we can see from the figures, the most frequent token is "." and the number is the same. The second token is "the", after stemming, there are more "the" because there are some tokens are stemmed as "the", and the token "the" itself are still unchanged. Besides, some tokens after stemming disappear from the top 5 such as "this", and it becomes "thi", and the number of "thi" after stemming (around 2500) is more than "this" before stemming (around 2000), which because some other tokens also becomes "thi" after stemming.

In summary, after stemming, the number of unique tokens becomes less, but because the total number of tokens in the same dataset is unchanged, some tokens after stemming becomes the same token, so the average occurrence number of each token becomes higher.

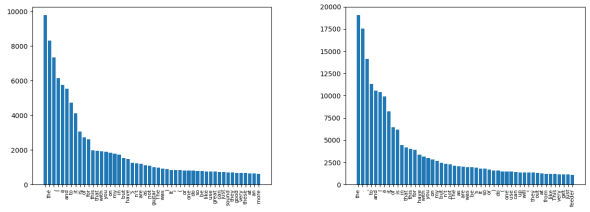

Figure 4: Top 50 frequent tokens and their occur numbers in the two datasets before stemming.

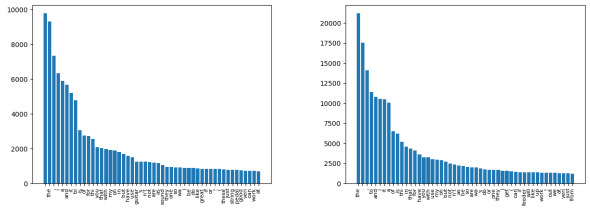

Figure 5: Top 50 frequent tokens and their occur numbers in the two datasets after stemming.

top 10 indicative words in the dataset		
dataset	Musical Instruments	Patio, Lawn and Garden
NO. 1	guitar (0.5623)	feeder (0.2081)
NO. 2	strings (0.3566)	trap (0.1630)
NO. 3	tuner (0.1484)	yard (0.1263)
NO. 4	guitars (0.1421)	garden (0.1079)
NO. 5	mic (0.1224)	plants (0.1077)
NO. 6	capo (0.1184)	hose (0.0903)
NO. 7	acoustic (0.0957)	birds (0.0791)
NO. 8	instrument (0.0815)	bait (0.0717)
NO. 9	Fender (0.0768)	plant (0.0677)
NO. 10	bass (0.0659)	mower (0.0664)

Table 2: Top 10 indicative words in the datasets measured by pointwise relative entropy.

1.6 Indicative Words

We implemented pointwise relative entropy to measure the indicativeness of a word, and list the top-10 most indicative words in each of the two datasets. The pointwise relative entropy is defined as follows: Let $P(w|D_i)$ be the probability of observing word w in all reviews in dataset D_i, then the relative entropy for word w can be computed as:

$$P(w|D_1) \times \log(\frac{P(w|D_1)}{P(w|D_2)})$$

If w doesn't occur in D_2, then we set $P(w|D_2) = 1 \times 10^{-50}$ to avoid runtime error. We use NLTK word tokenization function in this task. Then the top 10 words with the highest indicativeness of the two datasets are shown in the table 2.

As we can see from the results, the indicative words, to some extent, reflects the theme characteristics and general content of the datasets. For the datatset *Musical Instruments*, the top 10 indicative words measured by pointwise relative entropy are related to the musical instruments, such as "guitar", "tuner" and "mic". For the dataset *Patio, Lawn and Garden*, the top 10 indicative words are about gardening topic, such as "feeder", "trap" and "yard".

We can also use other measures such as TF-IDF, but the top 10 results should be the same, so we skip the overview here for brevity.

2 SIMPLE SEARCH ENGINE

In this part, we develop a search engine to index and search reviews. The simple search engine supports various index and query methods.

2.1 Search

Search is an act of information retrieval, which is an significant part of text management and processing. Search engine is an information retrieval model to help user to find the documents which best matches information need. It is widely used in many applications.

Generally speaking, search engine can be divided into two parts: index and query.

- `index`: Inverted index stores maps between a term and where it is stored in set of documents under full-text search.
- `query`: Query is the text in which user expresses a information need.

2.2 Framework

Our search engine also consists of two parts: index and query. We execute index part to establish an index first, and then input the information need to create a suitable query. With the query into the index, we can get the result. Figure 6 shows the framework.

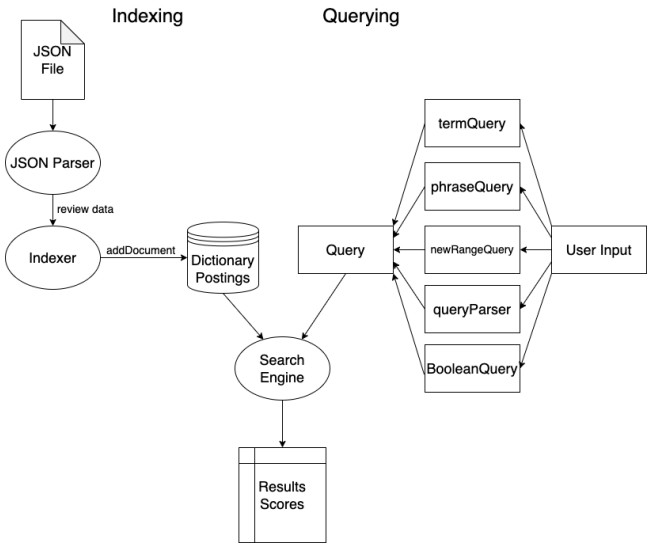

Figure 6: Framework

For index, the json files with review records are firstly parsed by json parser to obtained the records. Then we open an index and

process the review records into documents, which are then added to the index.

For query, there are five methods of query: term query, phrase query, range search on "overall", high-level query and multiple field searching, which will be described in more detail below. After inputting the method, search field and query context, the query will be conveyed to index for search. Finally, the result will be showed to user.

2.3 Achievement

Our achievement is based on Lucene[3] and json-simple[4] this two Java tool packages. In the following, we discuss the details of our implementation.

2.3.1 Index. In the java file of index, we set a file path and create analyzer first. We try three different analyzers for indexing:

(1) default StandardAnalyzer of Lucene: this analyzer will execute case folding but not stopwords removal and stemming.
(2) StandardAnalyzer with stopwords of EnglishAnalyzer: this analyzer will execute case folding and stopwords removal.
(3) EnglishAnalyzer of Lucene: this analyzer will execute case folding, stopwords removal and stemming.

After we create index with the analyzer, parse the json files with json-simple and process its different fields to create documents. Then we convey the documents to the index and save it to the local file. In our experiments, we use standard Analyzer (i.e., with case folding but no stopwords or stemming) for indexing in the next querying or searching stage.

2.3.2 Query. In the java file of query, we may also need an analyzer to parse the query in QueryParser. But for some queries such as term query and phrase query, we want the searched results should be exactly the same as the query itself, so we don't need any process for the query. There are five choices for user:

(1) Term Query: there are five fields for term query: asin, reviewText, reviewerName, reviewerID and summary. It is suitable for searching exactly one term. We use *TermQuery* class in the Lucene to build term query.
(2) Phrase Query: there are three fields for phrase query: reviewText, reviewerName and summary. It is suitable for searching continuous phrase. We use *PhraseQuery* class in Lucene to implement it.
(3) Range Query: query in a range on the overall field which are double type numbers. We use *DoublePoint.newRangeQuery* to implement range query, users are required to input the lower and higher boundaries of search range.
(4) High-level Query: there are three fields for phrase query: reviewText, reviewerName and summary. It is high-level query which is suitable for many situations, such as boolean query, fuzzy query and so on. Figure 7 shows the example [1]. We use QueryParser class in Lucene to parse the query. However, users need to be familiar to those query usage and syntax.
(5) Multiple Field Query: all the fields mentioned above can use multiple field query: specifically, asin, reviewText, reviewerName, reviewerID, summary and overall. It allows user to input the search field and query context as many as you wish, then the system merges them together by using BooleanQuery in Lucene to get the final multiple field query. Users are also required to choose BooleanClause types, such as MUST, SHOULD or MUST_NOT, which need users to be familiar to those query usage.

Lucene query syntax examples

Query expression	Document matches if…
java	Contains the term *java* in the default field
java junit java OR junit	Contains the term *java* or *junit* or both in the default field (*the default operator can be changed to* AND)
+java +junit java AND junit	Contains both *java* and *junit* in the default field
title:ant	Contains the term *ant* in the title field
title:extreme –subject:sports	Contains *extreme* in the title and not *sports* in subject
(agile OR extreme) AND java	Boolean expression matches
title:"junit in action"	Phrase matches in title
title:"junit action"~5	Proximity matches (within 5) in title
java*	Wildcard matches
java~	Fuzzy matches
lastmodified:[1/1/09 TO 12/31/09]	Range matches

Figure 7: High-Level Query Example

2.4 Experiment

2.4.1 Index Analysis. We explore the indexing time by using different analyzers with different pre-processes. For fairness, we repeat indexing 10 times on the same computer and take the average.

(1) only case folding: file size 19.8M
   - *Musical Instrument*: 572.7 ms
   - *Patio, Lawn and Garden*: 768.2 ms
(2) case folding and stopwords removal: file size 18.4M
   - *Musical Instrument*: 676.6 ms
   - *Patio, Lawn and Garden*: 737.5 ms
(3) case folding, stopwords removal and stemming: file size 18.0M
   - *Musical Instrument*: 757.4 ms
   - *Patio, Lawn and Garden*: 909.4 ms

As we can see from the results, for the indexing file sizes, the file is the largest if there is only case folding but no stopwords removal, after remove the stopwords, the file size decreases, and the file becomes even smaller after stemming. Actually, case folding, stopwords removal and stemming can be regard as index compression, and more these process leads to less file size.

For the case with or without stopwords removal, the indexing time is close to each other. However, stemming leads to large time complexity. That's because for stemming, each token needs to be analyzed and processed specifically, but case folding and stopwords removal relatively fast and only requires a bit more process.

For the querying or searching task, we believe that excessive processing of index will reduce the accuracy of search, so we only use case folding pre-processing to obtain high accuracy.

2.4.2 Search Analysis. We tested each query with setting the max number of results to 100. Among these five fields, reviewerID and

asin are not tokenized, so they are only can be searched by accurate query.

(1) Term Query: see figure 8
- *Query Field*: asin
- *Query Context*: B000053F9J
- *Output Number*: 6
- *Query Time*: 133 ms
- *Output Analysis*: All reviews on the product with the asin "B000053F9J".

(2) Phrase Query: see figure 9
- *Query Field*: reviewText
- *Query Context*: like good
- *Output Number*: 12
- *Query Time*: 135 ms
- *Output Analysis*: All reviews with the reviewText contains exactly phrase "like good".

(3) Range Query:
- *Query Field*: overall
- *Query Context*: range from 4 to 5
- *Output Number*: 100
- *Query Time*: 232 ms
- *Output Analysis*: All reviews with overall scores range from 4 to 5.

(4) High-level Query:
- *Query Field*: reviewText
- *Query Context*: useful + now~
- *Output Number*: 100
- *Query Time*: 274 ms
- *Output Analysis*: All reviews with reviewText contains term "useful". No output matches "now" but the outputs match similar words like "no" or "not" by using ~ as fuzzy matching.

(5) Multiple Field Query: see figure 10
- *Query Field 1*: reviewerID
- *Context1 and Type*: ASYHYW0USZVN6 MUST
- *Query Field 2*: reviewerID
- *Context2 and Type*: good MUST NOT
- *Query Field 3*: overall
- *Context3 and Type*: 3 to 5 MUST
- *Output Number*: 4
- *Query Time*: 123 ms
- *Output Analysis*: All reviews with reviewText doesn't contain the term "good" but with overall score range from 3 to 5 posted by the reviewer with reviewerID "ASYHYW0USZVN6".

In summary, the fields asin, reviewText, reviewerName, reviewerID, summary and overall are indexed and searchable. We only use case folding but no stopwords removal and stemming when indexing for search accuracy, which means our search engine doesn't support case sensitive searching, but can distinguish the stopwords and the terms with the same stems but different affixes. Our search engine supports:

- term query on [asin/reviewText/reviewerName/reviewerID/-summary].
- phrase query on [reviewText/reviewerName/summary].
- range search on [overall].

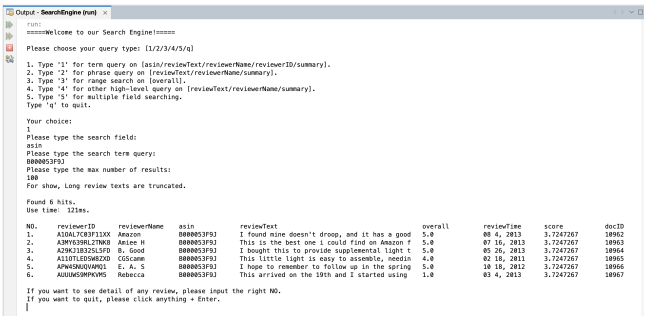

Figure 8: Term Query example for searching reviews with asin "B000053F9J".

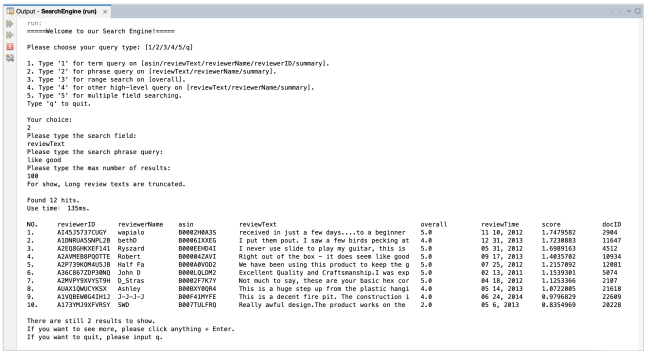

Figure 9: Phrase Query example for searching reviews with reviewText contains the phrase "like good".

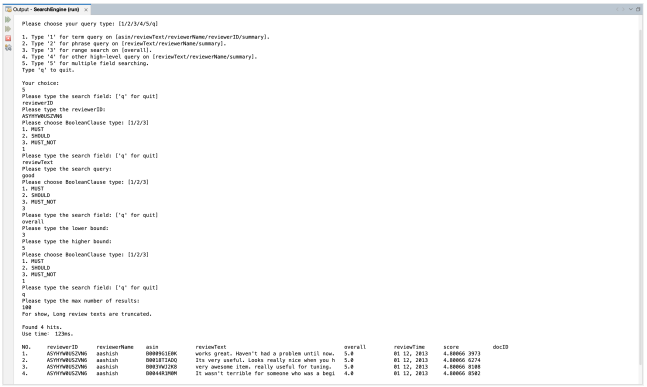

Figure 10: Multi-field Query example for searching reviews with reviewText doesn't contain the term "good" but with overall score range from 3 to 5 posted by the reviewer with reviewerID "ASYHYW0USZVN6".

- other high-level query, such as boolean query and fuzzy query on [reviewText/reviewerName/summary].
- multiple field searching on all the above fields.

and all the researched results are as expected.

3 REVIEW SUMMARIZER

In this part, our goal is to develop a summarizer to generate a summary of all reviews for a certain product. Items and reviews are included in json-formatted text in the dataset.

3.1 Summary

Summary always includes the most important information of an article. It is a series of words, phrases or brief sentences that include some high frequency words and several dramatic features of the given text. An ideal summary has the characteristics below:

- `frequency`: The words showing up in the summary must be in high frequency.
- `feature`: The summary is better to summarize the main features of the product.

3.2 Challenge

The challenge faced by the summarizer is that the extracted words and phrases do not reflect the key information in the article. For example, there are some stop words that appear in large numbers that will interfere with keywords. Different forms of words with the same root cause dispersion in frequency. Therefore, we need a heuristic method to extract keywords accurately and completely, while maintaining the original meaning of the article. The next subsection explains our solution, which is known as the *TextRank* algorithm.

3.3 TextRank Algorithm

The TextRank algorithm proposed by Mihalcea et al [6] generate sentence weights based on word frequency similarity between sentences. The algorithm combines the extraction of high-weight sentences in sequence, so that there will be no grammatical problems and can grasp the key points of the full text. The algorithm flow is shown in algorithm 1.

Algorithm 1: The flow of the TextRank algorithm

1 Generate vocabulary from text;
2 Sentence tokenization;
3 Calculate the similarity between sentences according to the word frequency between sentences;
4 Calculate the mean of the similarity of each sentence to other sentences;
5 Take sentences with mean greater than a specified threshold as part of the summary;
6 End;

By using the TextRank algorithm, the input chapter-level reports can be generated after adjusting the sentence centrality threshold to generate resizable summaries. The higher the sentence centrality threshold is set, the smaller the length of the text, but the more important the composed sentence is; the lower the sentence centrality threshold is set, the larger the text is, and the less important the composed sentence is. There are a series of procedures.

Sentence similarity calculation based on word vector

for a given text T, first perform sentence tokenization according to its content (split according to punctuation marks):

$$T = [S_1, S_2, ...S_m] \tag{1}$$

After the sentence tokenization is completed, remove stop words and perform word tokenization for each sentence:

$$S_i = [T_{i,1}, T_{i,2}, ..., T_{i,n}] \tag{2}$$

After each sentence is segmented, the similarity can be calculated based on the co-occurrence word frequency between the two sentences. For the given two sentences S_i, S_j:

$$Similarity(S_i, S_j) = \frac{\sum_{k=1}(t_k \in S_i) \wedge (t_k \in S_j)}{\log \mid S_i \mid + \log \mid S_j \mid} \tag{3}$$

The higher the similarity score of two sentences, the stronger the correlation. Sentence similarity is also the basis for calculating the importance of sentences in the text.

Sentence Centrality Calculation Based on Inter-sentence Similarity

After the similarity between sentences is obtained, it can be used as the weight between sentences. Let the weight between sentences i and j be Similarity, then the centrality S of sentence i is:

$$S_i = \frac{\sum_{i=1}^{N} Similarity_{i,j}}{N} \tag{4}$$

The centrality of the sentence calculates the arithmetic average of the degree of association between the sentence and other sentences in the full text. If the average value is higher, it indicates that the sentence is of high importance in the full text. The larger the centrality value, the more likely it is to extract the sentence as part of the abstract.

A Sentence Centrality-Based Text Summary Generation Method

We use a typical extractive summary generation algorithm TextRank as shown is algorithm 2 . The algorithm compares the similarity between sentences according to their co-occurrence words, and then calculates the importance of each sentence in the text. When the importance score of a sentence is greater than the threshold, the sentence is included as part of the summary.

Algorithm 2: TextRank algorithm

Input: Document T; split(T) into [S1, S2,..., Sm];
split(S_i) into [t1,t2,...tn];
for *i in m* **do**
  for *j in m* **do**
    $Similarity(S_i, S_j) = \frac{\sum_{k=1}(t_k \in S_i) \wedge (t_k \in S_j)}{\log |S_i| + \log |S_j|}$;
  end
  for *i in m* **do**
    $S_i = \frac{\sum_{i=1}^{N} Similarity_{i,j}}{N}$;
    ifS_i>threshold Res.append(S_i);
  end
end
return Res;

A limitation of the algorithm is that the threshold cannot be adjusted automatically. The higher the threshold, the harder the sentence is to be included in the summary. When the threshold is small, the summary will be longer. If the threshold is too high, the summary may be an empty string. So far, we need to adjust the threshold manually. Another limitation is that it only selects important sentences and concatenate them to generate a summary. It can't sort key words and phrases and compose them to sentences.

To evaluate the solution, we need to define a metric:

$$Score = \frac{\sum_{i=1}^{N} fq_i}{N} \tag{5}$$

where fq_i is the frequency of the ith word in the summary and N is the word count of the summary. A summary that includes more high-frequency words with shorter length is better, which makes score larger.

3.4 Achievement

To run this summarizer, enter the **review_summarizer** folder in command line mode, use the command **$python main.py –category CATEGORY –product_id ASIN** where CATEGORY is the category of the products and ASIN is ID of the product. To validate our solution, we choose two products from the chosen datasets, **Musical_Instruments** and **Patio_Lawn_and_Garden** respectively. The following are the experimental results.

Firstly, we select the product whose asin is B000165DSM in the Musical_Instruments dataset. There are 18 reviews of this product. After setting the threshold to 1.5, the summary is:

This is a well made, low noise and low cost cable. It was cheap, and came fast. No, but it doesn't cost as much either. the shielding was great as well. Cord works well and hasn't been any problem, to date. Plugs directly to the system, or my mini-amp.

The original review includes 18 sentences and 1602 words while the summary includes only 6 sentences and 50 words.

Secondly, from the dataset **Patio_Lawn_and_Garden** we choose product with asin B00004R9TL. There are 27 reviews concerning this product. A summary of the reviews is as follows:

It is factory spooled and is wound tighter than when I wound the spool by hand.To replace the spool:1. Remove battery2. It is just plug and go. The spools last longer than expected. These are very convenient, but you really pay a hefty price for that convenience. Otherwise, it's a good product. It seems to last forever. Summer worth of string in these three spools. But, in 3 packs, more economical. It is fairly easy to replace.

The original review contains 27 sentences and 1578 words while the summary includes 10 sentences and 78 words. The plain text of reviews lies in the folder named **review_summarizer**.

4 APPLICATION

For the application part, we develop a simple sentiment analysis application based on the sentiment scores of the review sentences determined by TextBlob, and draw pie charts according to the datasets to make results more intuitive.

The flow of this application is as follows. First, import reviews from the target dataset. Then switch all review sentences into lowercase forms. Remove punctuation and stopwords after word tokenization. Positive and negative sentiment words are classified based on polarity scores. After that, We print all positive and negative sentiment words in order of frequency. Finally, we draw word clouds of positive and negative words with background images for the different datasets. More specifically, an image of a piano for "Music Instruments" reviews, and an image of a weeder for "Patio Lawn and Garden" reviews. A specific set of asin numbers can be used to perform the above tasks on reviews of products with these aimed asin numbers. The process of generating word cloud is shown in Figure 11. The overall flow chart of the application is showed in figure 12

There are some technical details to explain. In this part, we use TextBlob [5] for Sentence-level sentiment analysis. TextBlob is a Python library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

Here is an example of using TextBlob. First, construct a TextBlob object with the input sentence as a parameter. Then we can get the sentiment score. Determine whether the review is positive, negative or neutral according to the distance between score and zero. If the score is larger than zero, the review is positive. It is negative if the score is less than zero. Otherwise, it is neutral. As the code shown below.

```
from textblob import TextBlob
#...
obj = TextBlob(review)
sentiment_score = obj.sentiment.polarity
if sentiment_score < -0.0:
    tag = 'Negative'
elif sentiment_score > 0.0:
    tag = 'Positive'
else:
    tag = 'Neutral'
}
```

For word-level sentiment analysis, we use **SentimentIntensityAnalyzer** in NLTK [10]. Words with a score not less than 0.5 will be considered positive, and words with a score not more than −0.5 will be considered negative. The example code is shown below:

```
from nltk.sentiment.vader
    import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
...
if (sid.polarity_scores(word)['compound'])\
    >= 0.5:
    pos_word_list.append(word)
elif (sid.polarity_scores(word)['compound'])\
    <= -0.5:
    neg_word_list.append(word)
else:
    neu_word_list.append(word)
```

The last step is to generate word cloud from the reviews with a given background picture. We implement this function with the WordCloud [7] library. It can easily draw a word cloud picture with the outline of a given image. To use this tool, we need to get a dictionary of all the words and their frequency. This could be easily achieved with **nltk.Freqdist()**. Words with high frequency always occupy more area. The word cloud is shown in figure 13 and figure 14 respectively. The pie charts in figure 15 show proportion of reviews with different sentiment. These figures can also be found in the code folder **Applications**.

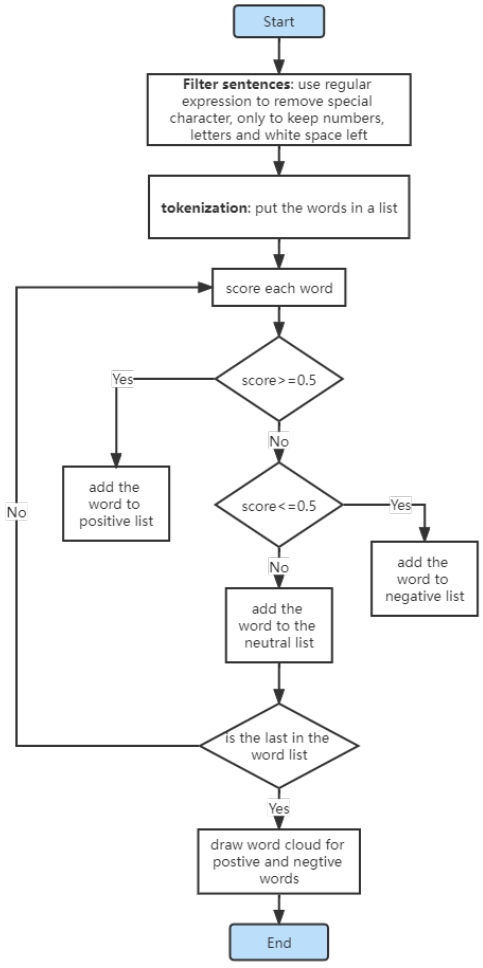

Figure 11: the generation of word cloud

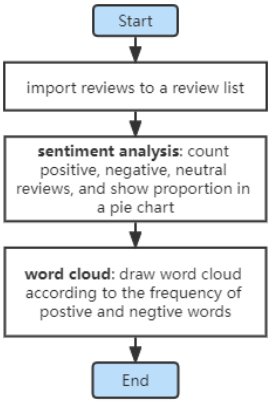

Figure 12: flow of the application

Figure 13: background, positive and negative word cloud of Musical Instruments

Figure 14: background, positive and negative word cloud of Patio Lawn and Garden

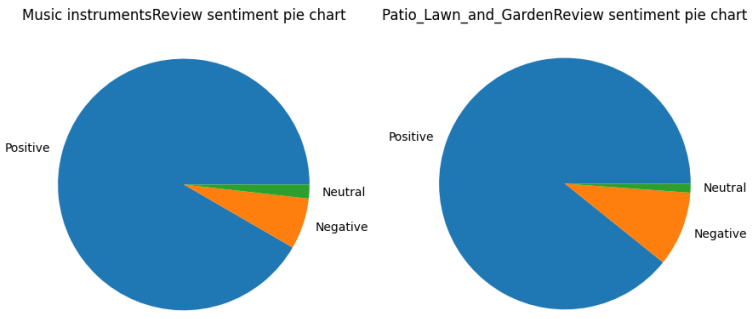

Figure 15: pie charts show the proportion of reviews with different sentiment

Individual contribution	
Member	Contribution
Li Zenong	Application, 15%
Luo Hao	Search Engine, Report, 15%
Wang Yanghao	Summarizer, 15%
Xu Xiangyu	Search Engine, Report, 15%
Yu Yue	Data Analysis, Search Engine, Report, 40%

Table 3: individual contribution

5 CONTRIBUTION OF INDIVIDUAL MEMBERS

We indicate contributions of individual members with percentage, as shown in table 3.

REFERENCES

[1] Sun Aixin. 2022. TEXT DATA MANAGEMENT PROCESSING, lecture notes. https://ntulearn.ntu.edu.sg/webapps/blackboard/content/listContent.jsp?course_id=_2616693_1&content_id=_3055762_1. Accessed October 19, 2022.

[2] Explosion. 2022. spaCy. https://spacy.io/. Accessed October 12, 2022.

[3] Yidong Fang. 2021. json-simple. https://lucene.apache.org/. Accessed April 11, 2021.

[4] The Apache Software Foundation. 2022. Apache Lucene. https://code.google.com/archive/p/json-simple/. Accessed October 15, 2022.

[5] Steven Loria. 2022. TextBlob: Simplified Text Processing. https://github.com/sloria/TextBlob. Accessed October 24, 2022.

[6] Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*. 404–411.

[7] Andreas Mueller. 2022. WordCloud for Python documentation. https://amueller.github.io/word_cloud/index.html. Accessed October 24, 2022.

[8] University of California San Diego. 2022. Amazon product data. https://jmcauley.ucsd.edu/data/amazon/. Accessed October 3, 2022.

[9] Trevor Tan. 2020. Wireless microphone best in market. https://www.straitstimes.com/tech/audio/wireless-microphone-best-in-market. Accessed October 23, 2022.

[10] NLTK Team. 2022. Natural Language Toolkit. https://www.nltk.org/. Accessed October 12, 2022.