1 Abstract

1.1 Data Format

We randomly pick up a review:
{'reviewerID': 'A2YTKPGZXFFKVK', 'asin': 'B0002MDTVS', 'reviewerName': 'Janeelizabeth', 'helpful': [0, 0], 'reviewText': 'Light weight... great for travel... it is perfect for what it is. It is adjustable for standing and sitting. Holds sheet music as it is intended to do. Came with a GREAT little travel bag. Folds up and sets up comfortably and easily.', 'overall': 5.0, 'summary': 'PERFECT!!!!!!!!', 'unixReviewTime': 1365206400, 'reviewTime': '1365206400'}

This review contains following components:

- **reviewerID**: A2YTKPGZXFFKVK
- **asin**: B0002MDTVS
- **reviewText**: Light weight... great for travel... it is perfect for what it is. It is adjustable for standing and sitting. Holds sheet music as it is intended to do. Came with a GREAT little travel bag. Folds up and sets up comfortably and easily.
- overall: 5.0
- **summary**: PERFECT!!!!!!!!
- **unixReviewTime**: 1365206400
- **reviewTime**: 1365206400

1.2 Overview

Our subsequent analysis will range from dataset analysis, simple search engine development, review summarizer development and application. The application is about sentiment analysis, where the most possible emotion of the user can be analyzed by extracting positive and negative words used in a review, regardless of the actual rating marks.

2 Data Analysis

This part is developed by Yang Shunping and Chen Zhelong.

Dataset Download and Sampling

We use two 5-core datasets(Musical Instruments and Digital Music):

```
response = wget.download(URL, "./data")
```

and randomly sample 200 products:

```
index = sorted(random.sample(range(0, length),
200))
```

The subsequent analysis will be based on the sampled datasets.

Writing Style

We randomly select a few reviews and observe the writing style.

Light weight... great for travel... it is perfect for what it is. It is adjustable for standing and sitting. Holds sheet music as it is intended to do. Came with a GREAT little travel bag. Folds up and sets up comfortably and easily.

I normally buy D'Addario Strings here. I buy the .011 or .012 strings. This is my first time playing Ernie Ball since I first began learning. When I first started on guitar I put .009 electric Ernie Ball on my acoustic so I could handle the pain and keep learning. Then I went to D'Addario strings. I wanted to try these since the under $5 price is good.They sound nice. They don't sound as crisp as the D'Addario strings. They sound more mellow and not as BOLD. They are 80/20 and I prefer those to the other strings out there. I can't say I love them or I hate them. I will probably stick with D'Addario since I don't find any difference in them. I bought some other Ernie Ball strings on Amazon that I will try soon. For the price I would buy these strings at least once to test out and see how you feel about them. I don't think they sound bad and if they do, it did not cost much to test them. Enjoy

I play guitar. I bought one each of the various thicknesses of these picks a few years ago. They're very interesting. These are soft rubber with a little bit of give, so they really soften your attack and cut higher frequencies, similar to using your thumb, but not exactly. I still pull one of these out every once in awhile. In certain applications, getting a particular sound for a song, say, these could be exactly what you're looking for. Overall these are well made, and the ones I bought will probably last me forever, but these would never work for me as a main pick. I'm glad I tried them though, as I have nothing else remotely like them.

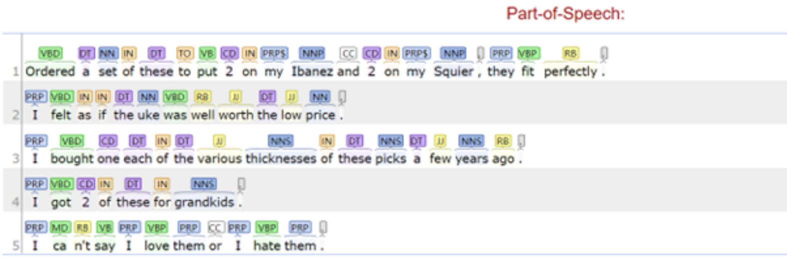
Eh. I wish I had more of a passion to play it, but....ehh. It's not bad if you want a uke, but it's pretty small. I don't know what I was expecting. I don't know shit about ukuleles but I do know get a bigger one if you're not comfortable with small frets. If you wanna be that one fat Hawaiian guy, Israel Kamakawiwo'ole, go for it. but I think he played a little bigger uke than this, lmao.

Ordered a set of these to put 2 on my Ibanez and 2 on my Squier, they fit perfectly. Highly recommended.

Compared with articles on The Straits Times, the reviews are informal and filled with colloquialism with modal particles, such as, Eh, ehh. Besides, there are also some mistakes: the use of definite article and spelling errors.

POS Tagging

We randomly pick up 5 sentences and apply POS tagging on coreNLP to get the results:


Part-of-Speech:

```
  VBD DT NN IN DT TO VB CD IN PRP$ NNP CC CD IN PRP$ NNP , PRP VBP RB .
1 Ordered a set of these to put 2 on my Ibanez and 2 on my Squier , they fit perfectly .
  PRP VBD IN IN DT NN VBD RB JJ DT JJ NN .
2 I felt as if the uke was well worth the low price .
  PRP VBD CD DT IN DT JJ NNS IN DT NNS DT JJ NNS RB .
3 I bought one each of the various thicknesses of these picks a few years ago .
  PRP VBD CD IN DT IN NNS .
4 I got 2 of these for grandkids .
  PRP MD RB VB PRP VBP PRP CC PRP VBP PRP .
5 I ca n't say I love them or I hate them .
```

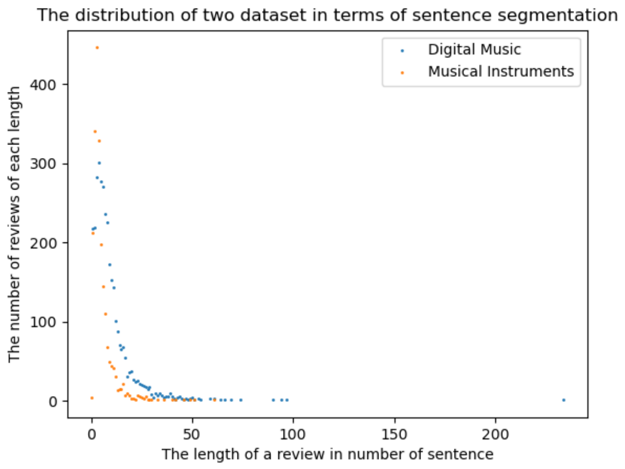
We can see that the errors may lead to wrong results of the tagging.

Sentence Segmentation

The function is used to segment sentences:

```
sentences = sent_tokenize(item['reviewText'])
```

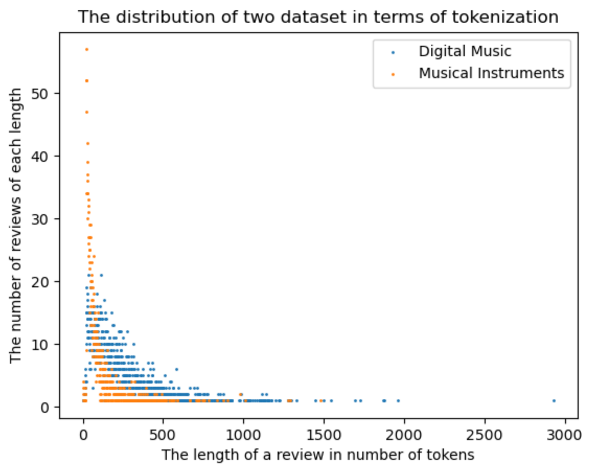
After the sentence segmentation, we compare the distribution of the two datasets: We can conclude that the length of sentences are roughly centralized from 2 to 15, while reviews of digital music tend to be longer than those of musical instruments on both number of reviews and length of reviews.



Tokenization and Stemming

We use Treebank to tokenize the reviews and compare the review length distribution:

```
tokenizer = TreebankWordTokenizer()
```



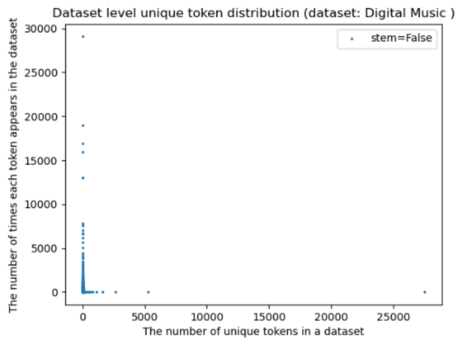
The plot is similar to that of sentence segmentation: when the length is small, there are more reviews on musical instruments. By contrast, reviews on musical instruments are longer when the number is small.

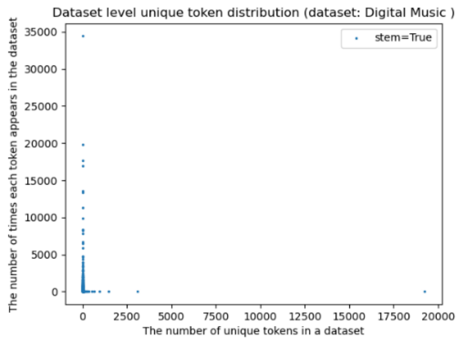
Then, we choose the stemming algorithm porter stemmer and compare different distributions of the data with and without stemming:

```
fdist = FreqDist(words)
result = dict(Counter(l))
```

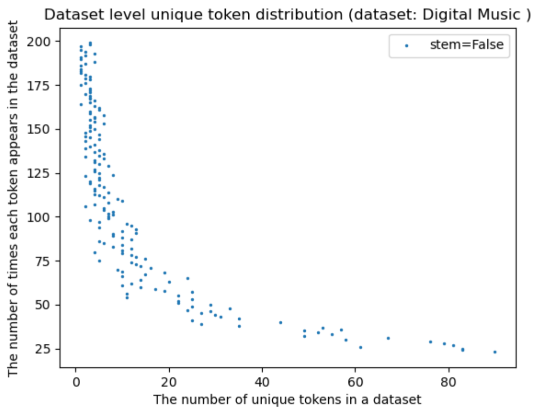
FreqDist helps us with the statistics of the frequency of words.

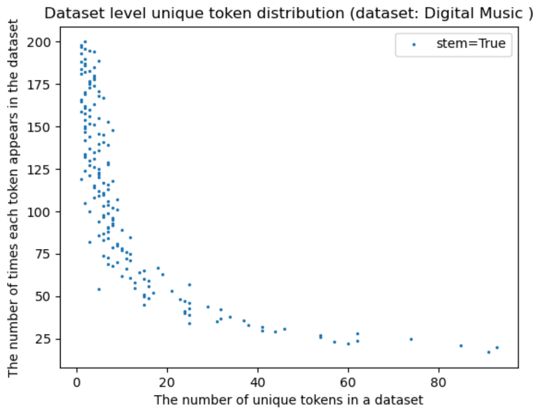
```
porter_stemmer = PorterStemmer()
```



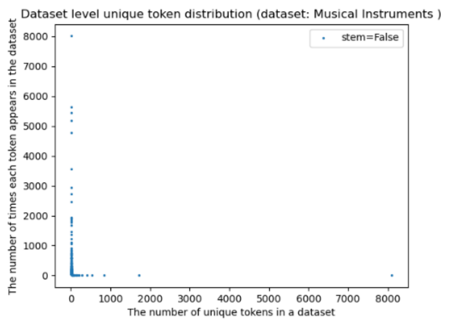


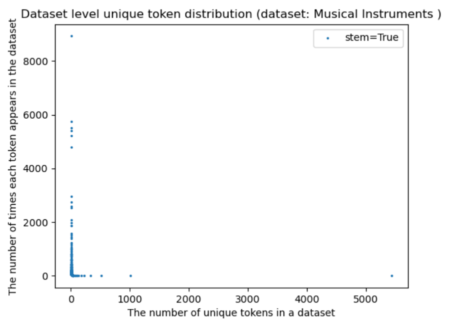
The original plot is not obvious on difference, so we set “zoom = True” to enlarge the plots.

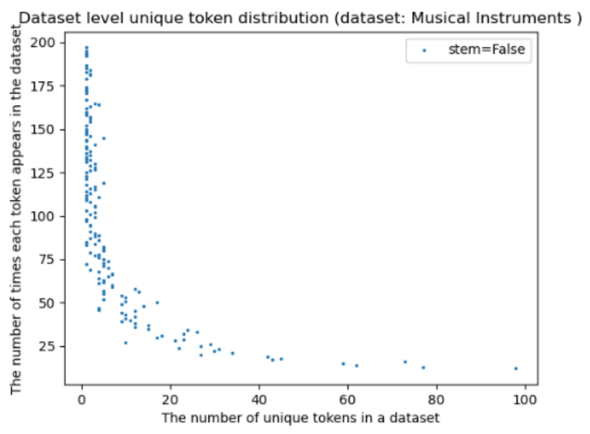


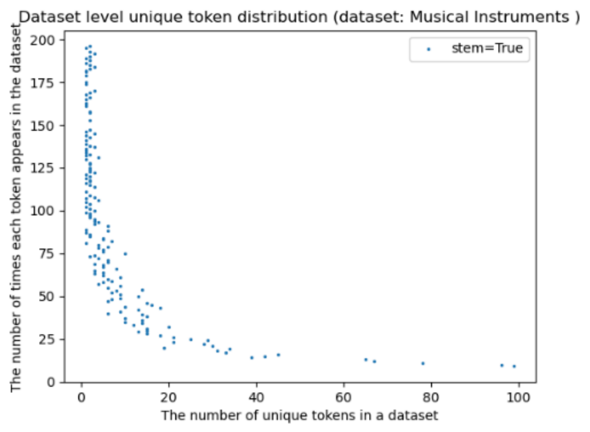


After the stemming, we can observe that there is nearly no difference for the tokens on the top and bottom, while samples in the middle are more intensive.









We can also see that these two datasets keep their features on the original tokenization even though with stemming. Besides, after stemming, they become more intensive in almost the same degree.

Indicative Words

We define "indicativeness" as pointwise relative entropy and list top-10 most indicative words in two datasets.

Top 10 indicative words for Digital Music are:
('album', 1.3178673214324397),
('song', 0.4195308256793319),
('songs', 0.36203158195604673),
('albums', 0.2694783373664743),
('The', 0.23456853303803632),
('lyrics', 0.1922226783809598),
('music', 0.1722552539764658),
('track', 0.1673718497234278),
('CD', 0.16443537336489175),
('tracks', 0.14093927567328543)

Top 10 indicative words for Musical Instruments are:
('pedal', 0.0752431550609445),
('strings', 0.044682528422418856),
('tuner', 0.041345988413314064),
('strap', 0.040120596661636185),
('use', 0.0351176045127393),
('pedals', 0.022777938776708685),
('cable', 0.020604474617834472),
('Fender', 0.019829410000199332),
('price', 0.019209075630510025),
('cables', 0.019165328103780438)

We can see that there are actually some indicative words standing for different features of two datasets such as album for music and pedal for instruments. However, there are also some general words: "the" in the digital music set and "price" in instrument set.

3 Development of a Simple Search Engine

This part is developed by Wu Dongjun.
In development of a Simple Search Engine, all the imported packages are shown below.

```
import string
import math
import demjson
import collections
```

Demjson package is used to identify abnormal json files.

3.1 Tokenization

After importing the useful packages, we need to do the tokenization for the field in json document first. Because Lucene can only match exact words. The search engine will miss a lot without tokenization. Some tokenizations which have been used in this search engine are shown below:
1.Lower case
2.Split field by space
3.N-Gram
4.Bigrams
5.Stemming
6.Strip punctuation...
And The code for this part is shown below:

```
def tokenize_field(field):
    string_field = str(field).lower() #Convert to lowercase
    transtab = str.maketrans('', '', string.punctuation)
    wout_punc_field = string_field.translate(transtab)  #Perform data cleaning
    return list(filter(None, wout_punc_field.split()))    #Returns the filtered split string
```

3.2 Scoring system

Then we need to built Scoring system, by designing schemes for calculations of the weight of a term and the similarity between documents vector and queries vector.
TD*IDF method can be used in calculating the weight of a matching term.
Shown as:

```
def idf(doc_count, inverted_index, term):#calculating idf
        if inverted_index.get(term):
                return math.log(doc_count / len(inverted_index[term]) + 1) + 1
        else:
                return 0
...
TF = math.sqrt(term.count(termnum))
IDF = idf(doc_count, inverted_index, termnum)
weight = TF * IDF * (1 / math.sqrt(field_len) if field_norm else 1)
term_weight_docs[field][termnum] = weight
```

And then, estimate the vector similarity by calculating dot product and do penalties based on the proportion of matched query terms. Finally, do cosine similarity which makes the dot product rebalance to 0-1 decimal.
Shown as:

```
for query_term, query_score in query_vector.items():
   if query_term in doc_vector:
           field_name = doc_vector[query_term]['field_name']
           field_boost = field_boosts.get(field_name, 1)#Boost a term match if found in field x by an exponent of n
           dot_product += query_score['score'] ** field_boost * doc_vector[query_term]['score']
           matching_terms += 1
dot_product = dot_product * (matching_terms / num_terms_in_query)
dot_product = dot_product / (query_vector_norm * doc_vector_norm)#cosine similarity
```

That is the core part of this search engine.

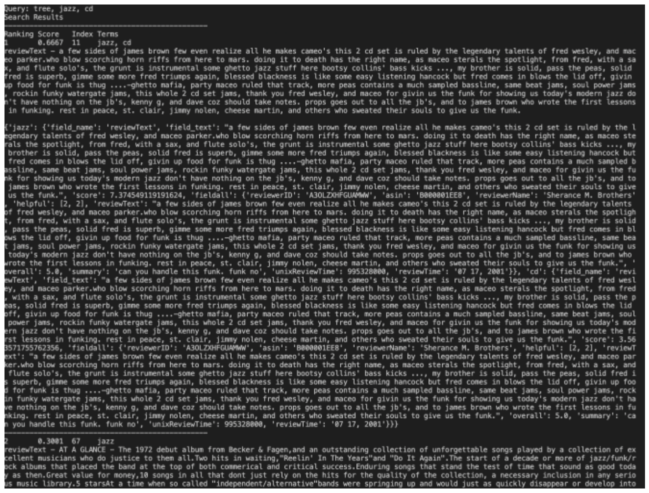
3.3 Reading Json

The last part is about the processing of abnormal json files and the display of query results. The json documents which downloaded from the website is non-standard because the attributes of terms are enclosed in single quotes rather than double quotes. So the program can not use the function"json.read()" to read downloaded json files. Files need to be modified.
Use dataset.py to create Digital Music.json and Musical Instruments.json then choose them as documents to search.
Shown as:

```
#json document, need to add ',' between '}' and '{' in assignment json style.
#add '[' at the beginning and ']' at the end.
#demjson for recognizing abnormal json files which use '' rather than "" to represent fieldects and attributes.
products1 = demjson.decode_file('Digital Music_200.json', encoding = 'utf-8')
products2 = demjson.decode_file('Musical Instruments_200.json', encoding = 'utf-8')
field_boosts = {'title': 1.1}
```

```
● search_engine = SearchEngine(alldocs = products1,
  field_norm = False)
● query1 = search_engine.query("tree, jazz, cd",
  field_boosts = field_boosts, num_results = 10)
● search_engine = SearchEngine(alldocs = products2,
  field_norm = False)
● query2 = search_engine.query("piano, guitar",
  field_boosts = field_boosts, num_results = 10)
```

3.4 The partial results



```
Query: tree, jazz, cd
Search Results
------------------------------------------------
Ranking Score   Index Terms
1       0.6667  11    jazz, cd
reviewText - a few sides of james brown few even realize all he makes cameo's this 2 cd set is ruled by the legendary talents of fred wesley, and mac
eo parker.who blow scorching horn riffs from here to mars. doing it to death has the right name, as maceo sterals the spotlight, from fred, with a sa
x, and flute solo's, the grunt is instrumental some ghetto jazz stuff here bootsy collins' bass kicks ..., my brother is solid, pass the peas, solid
fred is superb, gimme some more fred triumps again, blessed blackness is like some easy listening hancock but fred comes in blows the lid off, givin
up food for funk is thug ....-ghetto mafia, party maceo ruled that track, more peas contains a much sampled bassline, same beat jams, soul power jams
, rockin funky watergate jams, this whole 2 cd set jams, thank you fred wesley, and maceo for givin us the funk for showing us today's modern jazz do
n't have nothing on the jb's, kenny g, and dave coz should take notes. props goes out to all the jb's, and to james brown who wrote the first lessons
 in funking. rest in peace, st. clair, jimmy nolen, cheese martin, and others who sweated their souls to give us the funk.

{'jazz': {'field_name': 'reviewText', 'field_text': "a few sides of james brown few even realize all he makes cameo's this 2 cd set is ruled by the l
egendary talents of fred wesley, and maceo parker.who blow scorching horn riffs from here to mars. doing it to death has the right name, as maceo ste
rals the spotlight, from fred, with a sax, and flute solo's, the grunt is instrumental some ghetto jazz stuff here bootsy collins' bass kicks ..., my
 brother is solid, pass the peas, solid fred is superb, gimme some more fred triumps again, blessed blackness is like some easy listening hancock but
 fred comes in blows the lid off, givin up food for funk is thug ....-ghetto mafia, party maceo ruled that track, more peas contains a much sampled b
assline, same beat jams, soul power jams, rockin funky watergate jams, this whole 2 cd set jams, thank you fred wesley, and maceo for givin us the fu
nk for showing us today's modern jazz don't have nothing on the jb's, kenny g, and dave coz should take notes. props goes out to all the jb's, and to
 james brown who wrote the first lessons in funking. rest in peace, st. clair, jimmy nolen, cheese martin, and others who sweated their souls to give
 us the funk.", 'score': 7.374549119191624, 'fieldall': {'reviewerID': 'A3OLZXHFGUAMWW', 'asin': 'B000001EE8', 'reviewerName': 'Sherance M. Brothers'
, 'helpful': [2, 2], 'reviewText': "a few sides of james brown few even realize all he makes cameo's this 2 cd set is ruled by the legendary talents
of fred wesley, and maceo parker.who blow scorching horn riffs from here to mars. doing it to death has the right name, as maceo sterals the spotligh
t, from fred, with a sax, and flute solo's, the grunt is instrumental some ghetto jazz stuff here bootsy collins' bass kicks ..., my brother is solid
, pass the peas, solid fred is superb, gimme some more fred triumps again, blessed blackness is like some easy listening hancock but fred comes in bl
ows the lid off, givin up food for funk is thug ....-ghetto mafia, party maceo ruled that track, more peas contains a much sampled bassline, same bea
t jams, soul power jams, rockin funky watergate jams, this whole 2 cd set jams, thank you fred wesley, and maceo for givin us the funk for showing us
 today's modern jazz don't have nothing on the jb's, kenny g, and dave coz should take notes. props goes out to all the jb's, and to james brown who
wrote the first lessons in funking. rest in peace, st. clair, jimmy nolen, cheese martin, and others who sweated their souls to give us the funk.", '
overall': 5.0, 'summary': 'can you handle this funk. funk no', 'unixReviewTime': 995328000, 'reviewTime': '07 17, 2001'}}, 'cd': {'field_name': 'revi
ewText', 'field_text': "a few sides of james brown few even realize all he makes cameo's this 2 cd set is ruled by the legendary talents of fred wesl
ey, and maceo parker.who blow scorching horn riffs from here to mars. doing it to death has the right name, as maceo sterals the spotlight, from fred
, with a sax, and flute solo's, the grunt is instrumental some ghetto jazz stuff here bootsy collins' bass kicks ..., my brother is solid, pass the p
eas, solid fred is superb, gimme some more fred triumps again, blessed blackness is like some easy listening hancock but fred comes in blows the lid
off, givin up food for funk is thug ....-ghetto mafia, party maceo ruled that track, more peas contains a much sampled bassline, same beat jams, soul
 power jams, rockin funky watergate jams, this whole 2 cd set jams, thank you fred wesley, and maceo for givin us the funk for showing us today's mod
ern jazz don't have nothing on the jb's, kenny g, and dave coz should take notes. props goes out to all the jb's, and to james brown who wrote the fi
rst lessons in funking. rest in peace, st. clair, jimmy nolen, cheese martin, and others who sweated their souls to give us the funk.", 'score': 3.56
3571755762356, 'fieldall': {'reviewerID': 'A3OLZXHFGUAMWW', 'asin': 'B000001EE8', 'reviewerName': 'Sherance M. Brothers', 'helpful': [2, 2], 'reviewT
ext': "a few sides of james brown few even realize all he makes cameo's this 2 cd set is ruled by the legendary talents of fred wesley, and maceo par
ker.who blow scorching horn riffs from here to mars. doing it to death has the right name, as maceo sterals the spotlight, from fred, with a sax, and
 flute solo's, the grunt is instrumental some ghetto jazz stuff here bootsy collins' bass kicks ..., my brother is solid, pass the peas, solid fred i
s superb, gimme some more fred triumps again, blessed blackness is like some easy listening hancock but fred comes in blows the lid off, givin up foo
d for funk is thug ....-ghetto mafia, party maceo ruled that track, more peas contains a much sampled bassline, same beat jams, soul power jams, rock
in funky watergate jams, this whole 2 cd set jams, thank you fred wesley, and maceo for givin us the funk for showing us today's modern jazz don't ha
ve nothing on the jb's, kenny g, and dave coz should take notes. props goes out to all the jb's, and to james brown who wrote the first lessons in fu
nking. rest in peace, st. clair, jimmy nolen, cheese martin, and others who sweated their souls to give us the funk.", 'overall': 5.0, 'summary': 'ca
n you handle this funk. funk no', 'unixReviewTime': 995328000, 'reviewTime': '07 17, 2001'}}}
------------------------------------------------
2       0.3001  67    jazz
reviewText - AT A GLANCE - The 1972 debut album from Becker & Fagen,and an outstanding collection of unforgettable songs played by a collection of ex
cellent musicians who do justice to them all.Two hits in waiting,"Reelin' In The Years"and "Do It Again".The start of a decade or more of jazz/funk/r
ock albums that placed the band at the top of both commerical and critical success.Enduring songs that stand the test of time that sound as good toda
y as then.Great value for money,10 songs in all that dont just rely on the hits for the quality of the collection, a necessary inclusion in any serio
us music library.5 starsAt a time when so called "independent/alternative"bands were springing up and would just as quickly disappear or develop into
```

4 Development of a Review Summarizer

This part is developed by Jiang Haofeng.
Given a product, the summarizer should summarize all *reviewTexts* received for this particular product.

4.1 Ideal summary

An ideal summarizer will output a list of representative phrases having both pros and cons, regarding just this product, and much regardless of other products. The phrases are either "noun phrase with adjectives" or "verb phrases with adverbs". Based on my understanding, for a certain product, the best way to give readers a sense of the picture is what adjective or adverb users utilize to describe the functionality, appearance, feelings, etc. I prefer phrases to just keywords because modifiers are more informative with the actual objects or actions following them. To achieve this, the algorithm I used is TF-IDF method learned in AI6122 lecture. As shown in the lecture, the largest TF-IDF value is, the more occurrences a term in the document and the more rarity a term in the collection. This is perfectly tailored to the ideal summary.

4.2 Technical challenges

I used NLTK library to help me deal with raw texts. However, when doing word tokenization and POS tagging, NLTK may not have the expected results. For example, some writers forgot to add a space after a period, which results in NLTK recognizing "the last word of a sentence"-"the period"-"the first word of the next sentence" as a single word. It will seriously affect the counting in TF-IDF weight calculation. This can be solved by manually adding strips to all punctuations. Another unexpected result is tagging wrong class to a word, since a word may have multiple classes depending on the context. This is a small issue since I used regular expressions to select grammar and hence my rule can be tolerant enough.

```
●   def punctuation_strip(string):
●       puncts = ",.?:;!"
●       for p in puncts:
●           string = string.replace(p,' '+p+' ')
●       return string
```

In terms of regular expressions, I have to consider what is the best rule for the ideal output. Apart from lexical variants (like plural, comparative adjectives, or verb tenses), I add modal to the "adverb" category, since I think a modal verb can make the verb phrase more complete. Another tricky tag is the proper noun. I finally abandon proper nouns in "noun" category, since proper nouns are bound to win high TF-IDF weights and appear in the summary result. It will make the solution full of hard-understandable words. That is not an approachable summary supposed to be.

```
●   noun = "(<NN>|<NNS>)"
●   verb = "(<VB>|<VBD>|<VBG>|<VBN>|<VBP>|<VBZ>)"
●   adj = "(<JJ>|<JJS>|<JJR>)"
●   adv = "(<MD>|<RB>|<RBR>|<RBS>)"
```

Besides, when doing TF-IDF method, I realize that word normalization is important. Everything is fine except lemmatization. I used WordNetLemmatizer in NLTK and similarly, some lemmatization results are wrong. For example, the verb "has" will be lemmatized to "ha". Although it will not cause huge problems as long as all words in the documents and collections have the same lemmatizing standard, the problem occurs when showing the final summary result. If showing the phrases with lemmatization, some of the words will be strange (like "ha"). Otherwise, very similar nouns phrases like "nice car" and "nice cars" may appear together. To solve it, I used a smart way to compromise both sides. That is, after lemmatizing all phrases, a lemmatized phrase will not be calculated TF-IDF value if the phrase contains a calculated phrase, or reversely, if it is contained by a calculated phrase.

```
●   for s in strlist:
●     keys = list(map(normalize, score_dict.keys()
    ))
●     if normalize(s) in ' '.join(keys) or \
●     sum(map(lambda k: k in normalize(s), keys))>
    0:
●       continue
●     ...
```

Finally, to maintain "length does not beat informativity", I only calculate TF-IDF weights for words whose POS tag is not <DT> or <MD>.

4.3 Code walkthrough

Each component in my solution is as follows:

[summarizer.py]

- **get_global_values**, **set_global_values**
  Enable using datasets among different python files.
- **adj_NP**, **adv_VP**
  Generate the two regular expressions. Using functions to present them is easy to follow and debug.
- **punctuation_strip**
  Avoid unexpected tokenization result. (Shown in §3.2)
- **extract_grammar**
  The core of extract candidate noun phrases and verb phrases based on the two regular expressions just defined. Using nltk.RegexpParser to get a parse tree.
- **compress_tree**, **make_list**
  Process the parse tree and get the list of both candidate strings and the corresponding tags. Using HOF "map" because it is superior to loop regarding on its efficiency and less function calls.
- **remove_abb**, **normalize, normalize_sentence**

Normalize the candidate phrases by case-folding, lemmatization, abbreviation processing. Using HOF for the same reason.

- **process_product**
  The first function that related to the actual dataset. Use a for loop to traverse through the dataset and find the target product id. Utilize a series helper functions defined above. To reduce repeated iteration, I record all useful information just in one loop. Therefore, the output is a tuple of raw_text, strlist, tagdict.
- **tfidf**
  The essence of our weight algorithm: TF-IDF. For TF, I use a "smoothing method" (make it 1 if 0) to deal with unexpected NLTK results. Just as lecture notes, use log to avoid large numbers, since log preserves monotonicity.
- **generate_score**
  Generate the TF-IDF weight of each candidate phrases. I normalize all words in the whole collection, in order to count the TF and DF of each token regardless of its variants. The two "abandoning situation" introduced in §3.2 also apply here. I believe this is the processing that best align to the algorithm I use.
- **generate_summary**
  Sort and output n highest scores among candidate phrases. Using heapq function because it is faster.
- **main body**
  Since our group choose "Digital Music" and "Musical Instrument" as two datasets, the main body will first initialize the dataset with these two types. Then specifically choose a product which has 10 reviews from each dataset, and produce the review summary. (Will show in §3.6)

[summarizerUI.py]

- **main body**
  This python program is an interactive UI for you to play with. You can view review summary of the sampled product **in any category** you want (i.e. not restrained to the datasets used in this assignment), just by following the instructions on the command line. An outstanding point is that this program can "memorize" the dataset you have been created. That is, it will only download and initialize dataset for each type once, unless you exit the program. Users will not have to wait for the dataset initialization if the type has been visited before.
  <u>If you want to view the products in the datasets chosen by our group, please choose index 21 or 22 in the first input.</u>

4.4 Limitations

Writer's typos

Since this part has implemented very few tolerant retrieval algorithms, some writer's typos may appear in the final summary result. In TF-IDF algorithm, typos may be considered quite "unique" and results in high score, even though the phrase is meaningless.

Weak informativity phrases

The summary result may still contain phrases that are not so informative. The reason mainly because my regex rule is tolerant.

4.5 Evaluation

Some alternative solutions may have different intuition about an ideal summary. For example, text classification algorithms can be used to classify positive and negative reviews. I think the evaluation can be based on following criteria:

1. Running time / space complexity
2. Clearness and understandability (i.e. whether readers could understand the words in the summary)
3. Usefulness (i.e. whether a possible figure of the product can be imagined by reading the summary)
4. Alignment with the actual reviews

4.6 Output examples

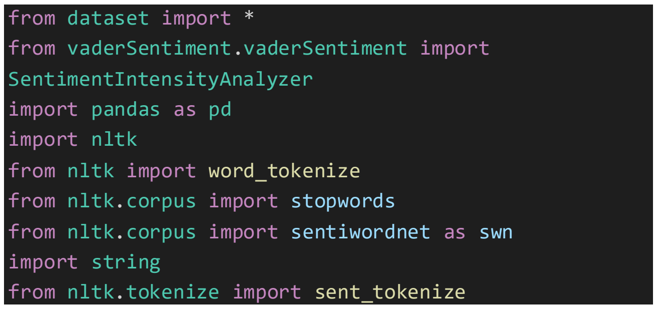
As mentioned in §3.2, the main body of summarizer.py serves this section. It chooses product B000068O4H in "Musical Instrument" and product B000001DYS in "Digital Music", each of which has exactly 10 reviews. Since every product has at least 5 reviews, I think 10 is a moderate number.

```
type = Musical_Instruments, asin = B000068O4H
Review summary:
    a basic female XLR
    high impedance input
    low impedance cable
    an xlr signal
    an old funky mixer
    a nice little adapter
    regular mic cables
    handy many times
    local guitar stores
    particular situation


type = Digital_Music, asin = B000001DYS
Review summary:
    the funky musical pool
    this solid single-disc collection
    the inadequate number
    roller skating
    the smooth soulfulness
    emotive vigor
    innovative funk acts
    great funk dance songs
    unsatisfying example
    more devout fans
```

5 Application

This part is developed by Cao Yifei.
All the third-part libraries used in this part is shown below.



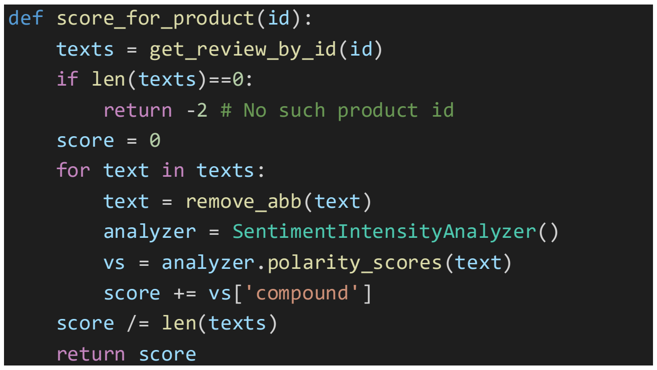
```
from dataset import *
from vaderSentiment.vaderSentiment import
SentimentIntensityAnalyzer
import pandas as pd
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.corpus import sentiwordnet as swn
import string
from nltk.tokenize import sent_tokenize
```

Among them, dataset is a class created by our team. It is mainly used to get raw data and provide data for certain attributes. VaderSentiment can provide a compound score between -1 and 1 for a sentence. Pandas can help me handle data easily and quickly. nltk helps me achieve basic word tokenization and part of speech tagging, get basic stop words and emotion words, etc. string helps me get punctuation.

5.1 Product Ranking

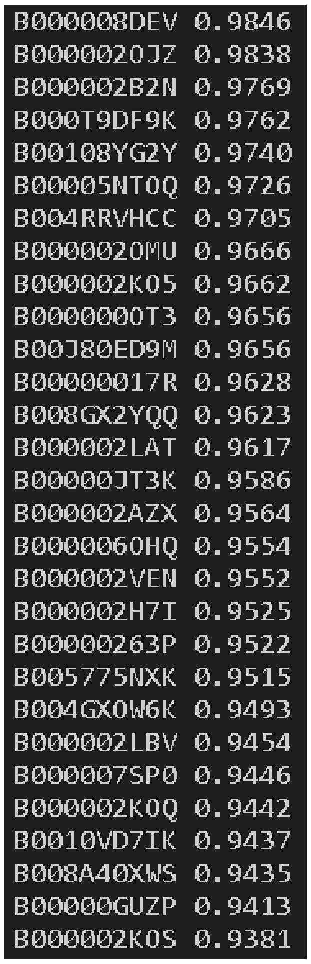
Sometimes users dare not give low marks because the direct rating is too obvious or deliberately give low marks to reflect their emotions. Compared to the 'overall' part of the data, reviews contain more information and are more useful. I used sentiment analysis to average the scores of all reviews on the same product and ranked all the products. This result has certain reference value to the good or bad of products.

score_for_product() is used for scoring for product. It computes sentiment score for each review whose asin is same as the passing parameters, id. It then averages the sentiment scores of all the reviews for a product in sample and returns a overall score.



```
def score_for_product(id):
    texts = get_review_by_id(id)
    if len(texts)==0:
        return -2 # No such product id
    score = 0
    for text in texts:
        text = remove_abb(text)
        analyzer = SentimentIntensityAnalyzer()
        vs = analyzer.polarity_scores(text)
        score += vs['compound']
    score /= len(texts)
    return score
```

ranking_of_products() was created to realize ranking. It adds a tuple of each asin and score to the list. It then sorts the list and returns the list.

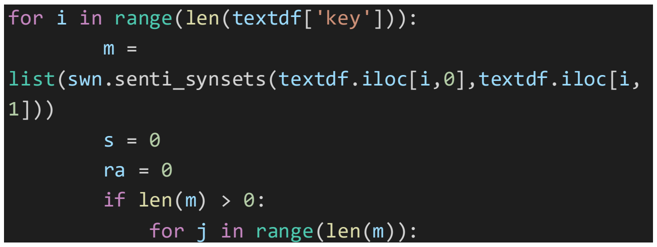


```
B000008DEV 0.9846
B000002OJZ 0.9838
B000002B2N 0.9769
B000T9DF9K 0.9762
B00108YG2Y 0.9740
B00005NTOQ 0.9726
B004RRVHCC 0.9705
B000002OMU 0.9666
B000002KO5 0.9662
B000000OT3 0.9656
B00J8OED9M 0.9656
B00000017R 0.9628
B008GX2YQQ 0.9623
B000002LAT 0.9617
B00000JT3K 0.9586
B000002AZX 0.9564
B000006OHQ 0.9554
B000002VEN 0.9552
B000002H7I 0.9525
B00000263P 0.9522
B005775NXK 0.9515
B004GXOW6K 0.9493
B000002LBV 0.9454
B000007SP0 0.9446
B000002KOQ 0.9442
B001OVD7IK 0.9437
B008A4OXWS 0.9435
B00000GUZP 0.9413
B000002KOS 0.9381
```

5.2 Get Positive & Negative Words

Reading reviews one by one is hard work. Extracting emotional words from reviews can help employees better identify product problems.

Firstly, I have to make tokenlization of reviews and get sentiment score for each word. text_score(text) function realize this. It first counts word frequency and makes part-of-speech tagging. Because a word can have multiple parts of speech, and each part of speech may have multiple meanings, all the scores of the word in the part of speech are weighted to get the final score of the word. This part of the code comes from the web.



```
for i in range(len(textdf['key'])):
        m =
list(swn.senti_synsets(textdf.iloc[i,0],textdf.iloc[i,
1]))
        s = 0
        ra = 0
        if len(m) > 0:
            for j in range(len(m)):
                s += (m[j].pos_score()-
m[j].neg_score())/(j+1)
                ra += 1/(j+1)
            score.append(s/ra)
        else:
            score.append(0)
```

get_neg_words(text) function takes all the negative words in the sentence and returns them in a list. Since phrases need to be considered, word position information needs to be saved. The original word list consists of tuples of words, parts of speech, and scores. For example:

```
words.append((i[0],'n',temp_score))
```

It then decides each word with a score above the threshold(I set it to 0.1) to determine whether it makes a phrase with previous words. It adds the phrase to the results list if it can, otherwise the word itself will be added to the results list. The results list is returned when all the words have been checked. It mainly decides whether there is an adjective before a noun, an adverb before a verb, and an adjective before an adverb. For example,

```
elif words[index][1]=="a":
            if words[index][2]<=-0.1:
                sign*=-1
            temp = index - 1
            while words[temp][1]=="r":
                temp -= 1
                if words[temp][2]<=-0.1:
                    sign*=-1
            temp+=1
```

To determine the positivity of a phrase, I will judge the positivity sign of the score of each word in it. If the product of all symbols is positive it is positive and vice versa. Also look at the code above. sign is the positive sign for the phrase.
I select a product and get its list of positive and negative words. Because the screenshots are too small, I present some of the results.

positive words:
['cool', 'appeal', 'tastes', 'aggressive', 'times', 'exquisite', 'supporting', 'whole', 'tend', 'prefer', 'music', 'enjoy', 'bonus', 'master', 'pretty', 'impressive', 'quality', 'exceptionally good', 'fine', 'outstanding', 'essential', 'likes', 'sensuous', 'very enjoyable', 'sweet', 'expensive', 'considering', 'time', 'trust', 'perfect', 'classics', 'broader', 'originals', 'intimate', 'best', 'old', 'familiar', 'always obeying', 'hypnotic', 'e', 'wide-eyed', 'beauty', 'instantly happy', 'marvellous', 'weary', 'most important', 'significant', 'playful', 'complete', 'richness', 'unique', 'classic', 'truest', 'sense', 'consciousness', 'important', 'better', 'great', 'dynamics', 'far better']

negative words:
['not as good', 'sad', 'soulful', 'lightly crying', 'death', 'dizzy', 'complaint', 'short', 'caveat', 'out angry', 'truly mournful', 'minor', 'melancholy', 'forlorn', 'just fine', 'other', 'not other']

There are some words that are not judged correct, and there are other words that are almost irrelevant that appear in the list. I think the main reason is that sentiment scoring functions does not combine scoring with a specific field. For example, I chose digital music. But I do not have a list of adjectives about music. This causes the function to interpret 'soulful' as a derogatory term.