1. Text preprocessing

- Tokens and Normalization Techniques
- JSON, CSV, TXT handling

Sogang University Department of Business Administration
Professor Myung Suk Kim

Natural language processing overview

◆ The difference between structured and unstructured data
Structured data consists of subjects, variables, and observations, and they are written in a structure (spreadsheet) suitable for general statistical analysis.
Unstructured data are documents, images, audio, video, etc., and it is difficult to apply traditional statistical analysis techniques, and pre-processing is required for analysis.

◆ Difficulty of natural language (text) processing
Text, a representative example of unstructured data, is composed of natural language, and text corresponds to data encoded in natural language.
Since natural language has been created and changed with various usages, it has characteristics such as ambiguity, relevance, and ambiguity, and it is difficult to process with one rule because it dynamically changes meaning according to viewpoints and times.
Natural language is difficult to analyze because it can be interpreted differently depending on the meaning or context even if the preprocessing process is well done .
Sometimes, analysis considering emoticons(encoding of emotional expression) is necessary, and recently, emoji symbols are also actively used as an extension concept of natural language .

◆ corpus _
natural language A collection of related documents contained in the .
corpus > paragraph > syntax or sentence > word
word = syllabus + phonemes + affixes + letters

also includes metadata (information about the document , such as author and date of creation), so these parts must be distinguished when reading the corpus.

Natural language processing overview

◆ token
The unit of text analysis is called a token , and a token is a string encoded in computer binary numbe rs (bytes) to represent text. A token can be a word or a sentence.

◆ Korean Tokenization
Unlike English, Korean has postpositional particles, so you must understand the concept of morpheme.

Two morphemes :
(1) Independent morphemes : self-reliant as words such as
(2) Dependent morphemes : morphemes used in combination with other morphemes, such as

tokenization of words rather than word tokenization to obtain tokenization similar to English
necessary Part -of-speech tagging needs to check which parts of speech

◆ bag of words model
The word is evaluated how often it occurs with other words in a particular situation. See which words are interlocked and appear at the same time.

◆ (n-gram) analysis
By specifying the number of characters or words to be entered as n instead of 1 according to the sequence, it helps to understand the context .

Preprocessing step

❖ **Preprocessing step**

◆ Refine
Required before and after tokenization as a denoising step. When HTML is parsed, tag names are rem oved, and stopwords and special characters are removed after tokenization .

◆ Tokenization
Tokenize words or sentences by breaking text into desired unit .

◆ Normalization
A work that unifies words written in different forms into standard words . Normalization methods incl ude stem extraction and lemma extraction.

◆ Part-of-speech tagging
Tokenized words are marked by classifying parts of speech, and the same word can have different contexts depending on parts of speech, so it is necessary to distinguish the .

Regular expression

◆ Regular expression

regular expression is regular expression Or an expression that uses a type of formal language grammar called a regular grammar.

Regular expressions are also called lock language. The lock judges whether a given lock language sentence matches a particularly meaningful sentence and presents an appropriate response.

◆ The re module

```
import re
p = re.compile ('[a-z]+')
```

re.compile Compile the regular expression using , and work with the compiled object p.

https://wikidocs.net/4308

(1) String search using regular expression

match() : Matches the regular expression

search(): Searches

findall (): returns all strings (substrings) that match the regular expression return as list

finditer (): returns an

Regular expression

(2) method of match object

group() : returns

start() : of the matched string return

end(): returns

span(): returns

(3) Compilation option : used as

r e. DOTALL or re.S : allows

re.IGNORE or re.I : allows

r e. MULTILINE or re.M : allows matching against

re. VERBOSE or re.X : to make the regex easier to see and to use comments etc.

(4) grouping

g group (0) : the entire matched string

g group (n): String corresponding to the nth group

Regular expression

- Meta character : A character used for a separate purpose other than the original meaning of the character

```
. ^ $ * + ? { } [ ] \ | ( )
```

(1) character class [ ], \
in [ ] Matches characters . However , using ^ in [ ] means not .
special When using letters of meaning , uppercase letters mean the opposite of lowercase letters .

[a- zA -Z] : any alphabet ,

[0-9] : all digits , [^0-9] : non-digit characters

\d : any digit , \D : any non-digit character

\s : Matches the whitespace character , an expression like [\t\n\r\f\v] (the leading space means a space.)

\S : whitespace character matches anything that is not , an expression like [a-zA-Z0-9_]

\w : matches letters + digits , expressions such as [a-zA-Z0-9_]

\W : letter + number matches any character that is not , an expression such as [^a-zA-Z0-9_]

Regular expression

(2) dot(.)
It matches with all letters except for the newline character \n

Ex) a.b : between a and b all characters
Note) a[.]b : dot(.) between a and b

(3) Repetition (*): The character preceding * is repeated

Ex) ca*t : ct , cat , caat all applicable

(4) Repeat (+): Matches the character before the + repeated

(5) repeat ({ m,n }, ?):

{m} : immediately Matches the previous character m times Ex) do{2}g : doog
{m, n} : directly Matching the preceding character from number m to number n is permitted
Ex) do{2,3}g : doog , dooog
? : Same meaning as {0,1}

(6) Backslash (\ ) : Backslash Escape by using twice of backslash

Ex) \ \section : Search for a string called section.
If \ section is used , it is recognized as \s of the character class .

Regular expression

(7) or (|):

Ex) a|b : a or b

(8) Matching at the beginning of a string (^): same as

Ex) ^My : My to string Matches only if it is at the beginning

(9) Matches end of string ($) : Same as

Ex) strong$: matches only when strong is located at the end of the string

(10) Grouping ( ): Create a group .

Ex) (\w+) \ s ₩d+ : extracts only

Tokenization

0. Download the main library

```
import nltk
nltk.download('punkt')
nltk.download('webtext')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('omw-1.4')
```

Token split

1. token split

(1) one Split a sentence (split(): Convert a string to a list)

```
sentence = """Thomas Jefferson began building Monticello at the age of twenty-six."""
sentence.split ()
```

(Three quotes are just newlines Line wrapping output possible : Try using print())

(2) Can be extended to a corpus composed of one or more sentences : Paragraphs are divided into sentences , and each s entence is tokenized.

```
corpus = ["Thomas Jefferson began building Monticello at the age of twenty-six.",
"Bats can see via echolocation. See the bat sight sneeze!",
          "Wondering, she opened the door to the studio."  ]
corpus_split = [x.split() for x in corpus]

# tokenize.sent_tokenize(p) 문장으로 분할
# tokenize.word_tokenize(p) 단어로 분할
import nltk
nltk.download('punkt')
from nltk import tokenize
p = "She is a heroine. He is an AI developer. Don't be late, Mr. Kim's house is: very-near here!"
tokenize.sent_tokenize(p)    # Ctrl+Shif+Enter
tokenize.word_tokenize(p)    # Don't과 Kim's ?

from nltk.tokenize import word_tokenize, sent_tokenize
text = '''She is a heroine. He is an AI developer. Don't be late, Mr. Kim's house is: very-near here!'''
tokens = [word for sent in sent_tokenize(text) for word in word_tokenize(sent)]
```

Token split

(3) Other approach

```
# wordpunct_tokenize
import nltk
text = '''She is a heroine. He is an AI developer. Don't be late, Mr. Kim's house is: very-near here!'''
tokens = nltk.wordpunct_tokenize(text)  # Don't과 Kim's 및 .?
text = nltk.Text(tokens)
words = [w.lower() for w in text if w.isalpha()]
```

(4) etc

```
# text_to_word_sequence : Remove punctuation marks such as periods, commas, and exclamation marks
# Preserve apostrophe in cases like don't or jone's
from tensorflow.keras.preprocessing.text import text_to_word_sequence
text = '''She is a heroine. He is an AI developer. Don't be late, Mr. Kim's house is: very-near here!'''
print( text_to_word_sequence (text))

# TreebankWordTokenizer : Keep words composed of hyphens as one ,
# Separate words with an apostrophe ' fold ' , such as doesn't
from nltk.tokenize import TreebankWordTokenizer
tokenizer= TreebankWordTokenizer ()

text = '''She is a heroine. He is an AI developer. Don't be late, Mr. Kim's house is: very-near here!'''
print( tokenizer. tokenize (text))
```

Token frequency analysis

2. Token frequency analysis

```
from collections import Counter
Counter("Good morning, Kim".split ())
Counter("Good morning, Kim!".split ())
Counter( corpus_split [0]) # use


# of words combination : Modification of order ( translation of meaning is possible in relation to grammar )
from itertools import permutations
[" ".join(combo) for combo in permutations( 'Good morning Rosa!'.split (), 3 ) ]
s = "Find textbooks with titles containing 'NLP', or 'natural' and 'language'."
len (set( s.split ()))
```

(Note) " ". join(A): Convert A (consisting of multiple elements) in list form into a string in the form of jumping (each elem ent)

Regular expression

3. Regular Expressions

(1) String search using regular expression

```
import re
p = re.compile ('[az]+') # character repeated at least 1 time

m = p.match ("python") # match: If the first value of the string matches, return the result
print(m)
m.group() # group() : return match string
m.start() # start() : start position of match string
m.end() # end(): end position of match string
m.span() # span(): ( start , end ) position of match string

m1 = p.match("3 python") # match: because the first value of the string is a number None output
print(m1)

p. search("python").group() # search: find matching value in entire string and return if found

print(p. findall("life is too short")) # findall : returns a list of all strings matching the regular expression
p.findall("life is too short").group() # ERROR : cannot use group in list
print(p. finditer ("life is too short")) # finditer : similar to findall , but returns an object
```

Regular expression

(2) compile option

```
import re
p = re.compile('a.b', re.DOTALL)
p.match('a₩nb').group()

p = re.compile('[a-z]', re.I)
p.match('PyThon').group()
p.match('pYTHON').group()

#^python₩s₩w+  must ( on each line ) start with the string python, followed by whitespace, followed by a word .
p = re.compile ("^python₩s₩w+", re.MULTILINE )
data = """python one life is too short python two you need python python three"""
print( p. findall (data))
data="""python one
life is too short
python two
you need python
python three"""
print(p.findall(data))
```

(3) compile and string search together

```
import re
p = re.compile('Crow|Servo')
p.match('CrowHello').group()
re.search('^Life', 'Life is too short').group()   # re.compile('^Life)와 search의 연계
re.search('short$', 'Life is too short').group()
print( re.search ('short$', 'Life is too short, you need python'))
```

Regular expression

(4) grouping ( )

```
import re
p = re.compile ('(ABC)+')
p. search ('ABCABCABC OK?').group()
```

# If you use r in front of ₩ in compile , it means that \ is used as a metacharacter . Ex) r'₩s'

```
p = re.compile (r"₩w+₩s+₩d+[-]₩d+[-]₩d+")
p.search("park 010-1234-1234").group()

p = re.compile(r"(₩w+)₩s+(₩d+[-]₩d+[-]₩d+)")
p.search("park 010-1234-1234").group(1)
p.search("park 010-1234-1234").group(2)
```

(5) Forward searching (https://docs.python.org/3/library/re.html)

```
import re
p = re.compile(".+:")
p.search("http://google.com").group()

p = re.compile(".+(?=:)")
p.search("http://google.com").group()
```

Regular expression

(6) String replacement

```
import re
p = re.compile('(blue|white|red)')
p. sub('colour', 'blue socks and red shoes')

p = re.compile ('(blue|white|red)')
p.subn('colour', 'blue socks and red shoes') # print up to the total number of replaced words


s = '<html><head><title>Title</title>'
print(re.match('<.*>', s).span())
print(re.match('<.*>', s).group())
print(re.match('<.*?>', s).group())
```

Regular expression

(7) HTML tag remove (http://zeany.net/46)

```
import re
html = "<html><head>some header information</head> ₩
  <Body>it's start. <script src='..'>some script</script> ₩
<!-- some comments -->some <b>body</b> contents.. ₩n <a href ='some link'> gogo </a> ₩
  and other stuff ..₩
  <script>another</script></Body></html>"

type(html)
len (html)

# Extract only the contents of <Body> ~ </Body>
body = re.search('<body.*/body>', html, re.I|re.S).group()

Delete the contents of <script> ~ </script>
re.sub('<script.*?>.*?</script>', '', body, 0, re.I|re.S)

# Delete tags and comments
text = re.sub('<.+?>', '', body, 0, re.I|re.S)
print(text)

# ₩t|₩r|₩n|₩. eliminate
result = re.sub('₩t|₩r|₩n|₩.', '', text)
print(result)
```

N-gram

4. Token Split Improvement (Using Regular Expressions)

```
import re
sentence1 = """Thomas. Jefferson; began building Monticello ₩n at the age! of twenty-six?"""
tokens = re.split(r'[-₩s.,;!?]+', sentence1)

pattern = re.compile(r'([-₩s.,;!?])+')
tokens2 = pattern.split(sentence1)
tokens2[-10:]  # 마지막 토큰 10개만 출력

[x for x in tokens if x not in '-₩t₩n.,;!?']

from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'₩w+|$[0-9.]+|₩S+')
tokenizer.tokenize(sentence1)

from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(sentence1)
```

5. N-gram

```
from nltk.util import ngrams
list(ngrams(tokens, 2))
list(ngrams(tokens, 3))

two_grams = list(ngrams(tokens, 2))
[" ".join(x) for x in two_grams]
```

Stopwords / Handling / Unification of lowercase letters

6. **Stop word processing**

(1) Creating a dictionary of

```
stop_words = ['a', 'an', 'the', 'on', 'of', 'off', 'this', 'is']
tokens = ['the', 'house', 'is', 'on', 'fire']
tokens_without_stopwords = [x for x in tokens if x not in stop_words ]
print( tokens_without_stopwords )
```

(2) Download

```
import nltk
nltk.download('stopwords')
stop_words = nltk.corpus.stopwords.words('english')
len(stop_words)
stop_words[:7]
[sw for sw in stop_words if len(sw) == 1]
[ sw for sw in stop_words if len ( sw ) == 2]
```

(3) Download stopword dictionary from Scikit Learn

```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS as sklearn_stop_words
len(sklearn_stop_words )

len(set(stop_words).union(set(sklearn_stop_words))) # number of unions
len(set(stop_words).intersection(set(sklearn_stop_words))) # number of intersections
```

7. **Unification with lower case letters**

```
"Good morning, Kim". lower ()
tokens = ['House', 'Visitor', 'Center']
normalized_tokens = [ x.lower () for x in tokens]
print( normalized_tokens )
```

Stem / headword extraction

8. Stem / headword extraction

(1) s at the end of word

```
def stem(phrase):
    return ' '.join([re.findall('^(.*ss|.*?)(s)?$', word)[0][0].strip("'") for word in phrase.lower().split()])
stem('His house')
phrase = 'His house'
word = 'his'
re.findall('^(.*ss|.*?)(s)?$', word)
re.findall('^(.*ss|.*?)(s)?$', word)[0]
re.findall('^(.*ss|.*?)(s)?$', word)[0][0]
```

(2) Headword extraction : Ex) age value -> age

```
nltk.download (' wordnet ')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer ()
lemmatizer. lemmatize ('better') # default: noun
lemmatizer.lemmatize ('good', pos ='a') # a: adjective
lemmatizer.lemmatize('goods', pos='n')
lemmatizer.lemmatize('mining', pos='a')
```

(3) English stem extraction

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
' '.join([stemmer.stem(w).strip("") for w in "dish washer's washed dished. data mining, miners, mines".split()])
stemmer.stem('goodness')
```

Stem / headword extraction

(3) English stem extraction (계속)

```
import nltk
stem = nltk.stem.SnowballStemmer('english')
stem.stem("dish washer's washed dished. data  miners, mining")

# Several sentences
corpus = ["Thomas Jefferson began building Monticello at the age of twenty-six.",
          "Bats can see via echolocation. See the bat sight sneeze!",
          "Wondering, she opened the door to the studio."    ]
result = []
for sent in corpus:
    out = stem.stem(sent)
    result.append(out)
result

print(result)

import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()

corpus = ["Thomas Jefferson began building Monticello at the age of twenty-six.",
          "Bats can see via echolocation. See the bat sight sneeze!",
          "Wondering, she opened the door to the studio."
          ]
stemming_words = [ps.stem(w) for w in corpus]
```

Example

9. Example
(#https://duckkkk.com/entry/Kaggle-IMDB-%EA%B0%90%EC%A0%95-%EB%B6%84%EC%84%9D-Part-1)

```
from bs4 import BeautifulSoup
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
ps = PorterStemmer()

raw_review = '''One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They
 are right, as this is exactly what happened with me.<br /><br />The first thing that struck me about Oz was its brutal
ity and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint h
earted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of
 the word.<br /><br />It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentar
y. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and fac
e inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christia
ns, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.<br />
<br />I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forge
t pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first e
pisode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I develo
ped a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked
 guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class i
nmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may
become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side.'''
type( raw_review )
```

Example

```
# 1. Remove HTML
review_text = BeautifulSoup(raw_review, 'html.parser').get_text()

# 2. Convert non-alphabetic characters to spaces
letters_only = re.sub('[^a- zA -Z]', ' ', review_text)

# 3. Convert lower case
words = letters_only.lower().split()

# 4. Convert Stopwords to Sets
# It's much faster to search in a set than in a list in Python .
stops = set(stopwords. words('english'))

# 5. Stopwords eliminate
meaningful_words = [w for w in words if not w in stops]

# 6. Stemming
stemming_words = [ps.stem (w) for w in meaningful_words]

# 7. Combine into space separated strings and return the result
print(' '.join(stemming_words))
```

Korean Tokenization

10. Hangul Tokenization

```
# sentence tokenization
!pip install kss
import kss
text = "4차 산업 혁명이라는 용어가 나온 이후로 우리의 산업화의 방향은 하루가 다르게 변화하고 있습니다. 그 가운데에 [AI: 인공지능]과 [빅데이터]라는 화두는 사회 곳곳에 빠짐없이 등장하고 있습니다. 교육계에서도 이러한 사회 경제적 수요를 충족시키기 위해서 경쟁적으로 AI 빅데이터 대학원 및 단기 교육과정들이 전국적으로 우후죽순으로 설립되고 있습니다. 그러나, AI 빅데이터 기술을 어떻게 비즈니스 영역에 연결할 수 있는지에 대한 프로그램은 아직까지는 전무하다고 할 수 있습니다. 아무리 세계적 수준의 우수한 기술을 습득한다고 할지라도 이를 사업화 할 수 있는 역량이 부족하다면 성공으로 나아갈 수 있는 중요한 관문의 열쇠가 없는 것과 같은 상황일 것입니다."
print( kss. split_sentences (text))
len ( kss.split_sentences (text))

# KoNLPy stemmer
# Okt (Open Korea Text)
!pip install konlpy
from konlpy.tag import Okt
okt = Okt ()
text = "AI Big Data Innovation MBA trains AI big data experts with commercialization capabilities ."
print(okt.morphs(text)) # extract morphemes
print(okt.pos(text)) # Part of speech tagging
print(okt.nouns(text)) # Extract nouns
```

Since the term "4th Industrial Revolution" came out, the direction of our industrialization has been changing day by day . Among them, the topics of [AI: Artificial Intelligence] and [Big Data] are appearing everywhere in society. In order to meet these social and economic demands in the education world, AI big data graduate schools and short-term training courses are being established in succession nationwide , but programs on how to connect AI big data technology to the business field are not yet available . It can be said that there is none so far . No matter how excellent world-class technology is acquired, if the capacity to commercialize it is lacking, it will be the same as not having the key to success.

Token frequency analysis

```
# Komoran
from konlpy.tag import Komoran
komoran = komoran ()
text = "AI Big Data Innovation MBA trains AI big data experts with commercialization capabilities ."
print(komoran.morphs(text))
print(komoran.pos(text))
print(komoran.nouns(text))

# Hannanum
from konlpy.tag import Hannanum
tokenizer= Hannanum()
text = "AI Big Data Innovation MBA trains AI big data experts with commercialization capabilities ."
print(tokenizer.morphs(text))
print(tokenizer.pos(text))
print(tokenizer.knowns(text))

# Kkma
from konlpy.tag import Kkma
tokenizer= Kkma()
text = "AI Big Data Innovation MBA trains AI big data experts with commercialization capabilities ."
print(tokenizer.morphs(text))
print(tokenizer.pos(text))
print(tokenizer.knowns(text))
```

Word frequency graph

Reference : Python The Complete Guide to Text Mining Chapter

1. **Project Gutenber Use**

```
import nltk
nltk.download('point')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('gutenberg')

from nltk.corpus import gutenberg
file_names = gutenberg.fileids() # read file names
print(file_names)

doc_alice = gutenberg.open ('carroll-alice.txt').read()
print('#Num of characters used:', len(doc_alice)) # number of characters used
print('#Text sample:')
print(doc_alice[:500]) # print only

from nltk.tokenize import word_tokenize

tokens_alice = word_tokenize(doc_alice) # Execute tokenization
print('#Num of tokens used:', len(tokens_alice))
print('#Token sample:')
print( tokens_alice [:20])

from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
```

Word frequency graph

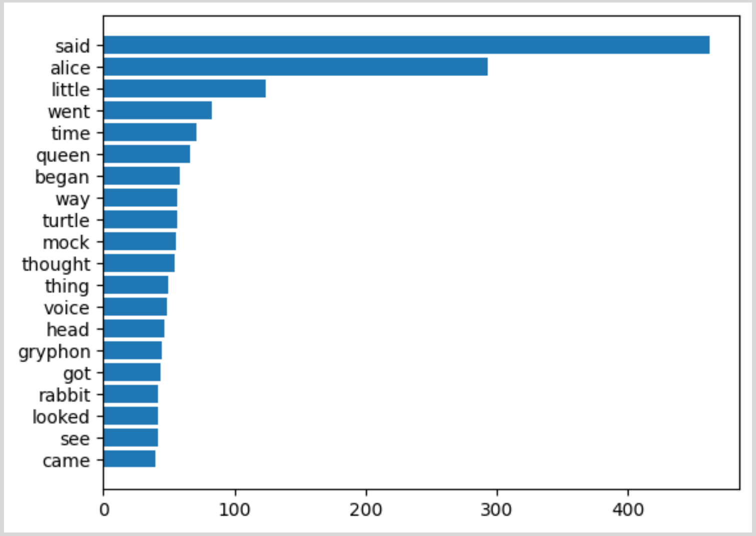
```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stem_tokens_alice = [ stemmer.stem (token) for token in tokens_alice ] # run stemming on all tokens
print('#Num of tokens after stemming:', len(stem_tokens_alice))
print('#Token sample:')
print(stem_tokens_alice[:20])

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lem_tokens_alice = [lemmatizer.lemmatize(token) for token in tokens_alice] # for token in tokens_alice
print('#Num of tokens after lemmatization:', len(lem_tokens_alice))
print('#Token sample:')
print(lem_tokens_alice[:20])

from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer("[₩w']{3,}")
reg_tokens_alice = tokenizer.tokenize(doc_alice.lower())
print('#Num of tokens with RegexpTokenizer:', len(reg_tokens_alice))
print('#Token sample:')
print( reg_tokens_alice[:20])

from nltk.corpus import stopwords # Words not normally analyzed
english_stops = set( stopwords.words('english')) # convert to set to avoid repetition
result_alice = [word for word in reg_tokens_alice if word not in english_stops]
# Create a list with only words excluding stopwords
print('#Num of tokens after stopword elimination:', len(result_alice))
print('#Token sample:')
print(result_alice[:20])
```

Word frequency graph

```
alice_word_count = dict()
for word in result_alice:
    alice_word_count[word] = alice_word_count.get(word, 0) + 1
print('#Num of used words:', len(alice_word_count))

sorted_word_count = sorted(alice_word_count, key=alice_word_count.get, reverse=True)
print("#Top 20 high frequency words:")
for key in sorted_word_count[:20]: #print top 20 frequency words
   print(f'{repr(key)}: {alice_word_count[key]}', end=', ’)


# Extract only some specific Part of Speech
# https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
# https://happygrammer.github.io/nlp/postag-set/
my_tag_set = ['NN', 'VB', 'VBD', 'JJ']
my_words = [word for word, tag in nltk.pos_tag(result_alice) if tag in my_tag_set]
print(my_words)

alice_word_count = dict()
for word in my_words:
    alice_word_count[word] = alice_word_count.get(word, 0) + 1
print('#Num of used words:', len(alice_word_count))

sorted_word_count = sorted( alice_word_count , key= alice_word_count.get , reverse=True)
print("#Top 20 high frequency words:")
for key in sorted_word_count [:20]: # print the top 20 most frequent words
   print(f'{ repr (key)}: {alice_word_count [key]}', end=', ')
```

Word frequency graph

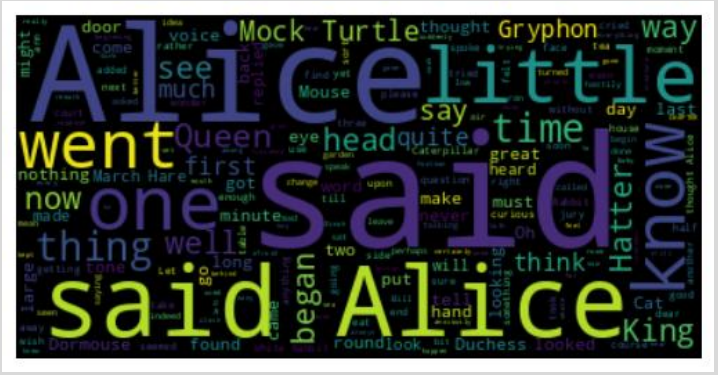
```
import matplotlib.pyplot as plt
%matplotlib inline

w = [ alice_word_count [key] for key in sorted_word_count ]
# Create a list by getting the frequency count for a sorted list of words
plt.plot(w)
plt.show()

n = sorted_word_count [:20] # extract only the top 20 most frequent words
w = [ alice_word_count [key] for key in n] # Extract frequencies for extracted words
plt.bar(range( len (n)), w,tick_label =n) # draw a bar graph
plt.show()

n = sorted_word_count [:20][::-1] # Extract the top 20 most frequent words and sort them in reverse order
w = [ alice_word_count [key] for key in n]
plt.barh (range( len(n)), w,tick_label =n) # horizontal bar graph
plt.show()
```

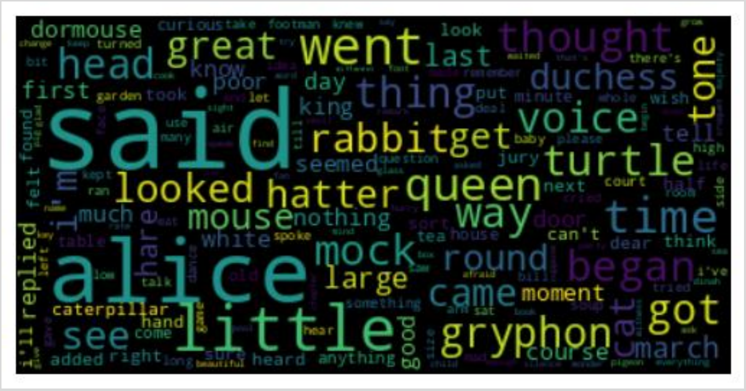


Word Clouds

1. Project Gutenber Use

```
!pip install wordcloud
from wordcloud import WordCloud
import nltk
from nltk.corpus import gutenberg
file_names = gutenberg.fileids () # read file titles .
print(file_names)

doc_alice = gutenberg.open('carroll-alice.txt').read()
# Generate a word cloud image
wordcloud = WordCloud().generate(doc_alice)
plt.axis("off")
plt.imshow(wordcloud, interpolation='bilinear') #이미지를 출력
plt.show()

wordcloud.to_array().shape
wordcloud = WordCloud(max_font_size=60).generate_from_frequencies(alice_word_count)
plt.figure()
plt.axis("off")
plt.imshow(wordcloud, interpolation="bilinear")
plt.show()
```

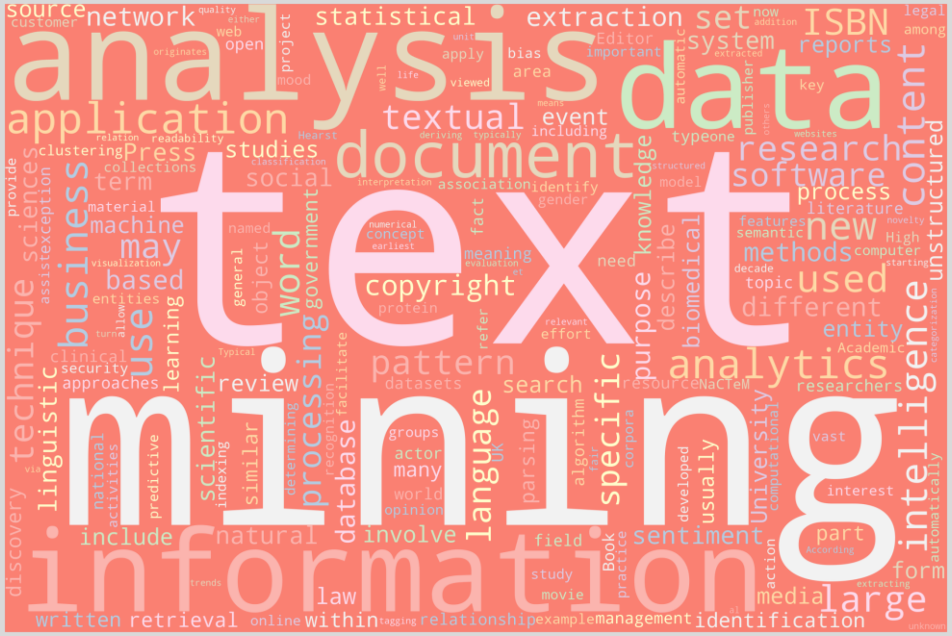




Word Clouds

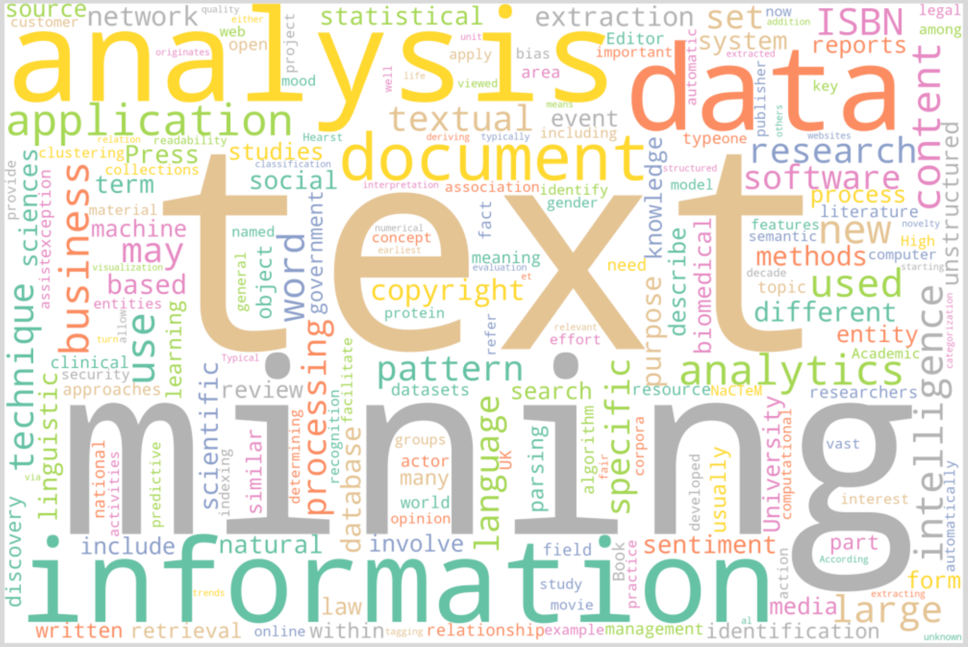
2. Wikipwdia: https://towardsdatascience.com/simple-wordcloud-in-python-2ae54a9f58e5

```
import wikipedia
import re
!pip install wordcloud

# Select
wiki = wikipedia.page ('text mining')
text = wiki.content

# preprocessing work
text = re.sub(r'==.*?==+', '', text)
text = text.replace('₩n', '')

import matplotlib.pyplot as plt

# Write a word cloud drawing function
def plot_cloud(wordcloud):
    plt.figure(figsize =(40, 30)) # set figure size
    plt.imshow(wordcloud )
    plt.axis("off"); #axis _ no notation
from wordcloud import WordCloud , STOPWORDS

# create word cloud 1
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1, background_color='salmon',
colormap='Pastel1', collocations=False, stopwords = STOPWORDS).generate(text)

# 그림 그리기
plot_cloud(wordcloud)
```



Word Clouds

```
# word cloud generation
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1, background_color='white', colormap='Set2', collocations=False, stopwords = STOPWORDS).generate(text)

# draw plot
plot_cloud(wordcloud)

# save image
wordcloud.to_file("wordcloud.png")
```



JSON(JavaScript Object Notation)

- Designed for data storage and exchange data format
- JavaScript programming to the language root leave there is word pad etc. write possible
- as key -value pairssave
- { } and The hierarchical structure of the document using [ ] Justice
- { } is an object save, [ ] is an array save
- height is double quotation marks in less, value is string only if double quotation marks in less
- Keys and values are separated by colons. division
- key-value the pair rest (,)as distinguish Arrangement undergarment value is with a comma classified

JSON

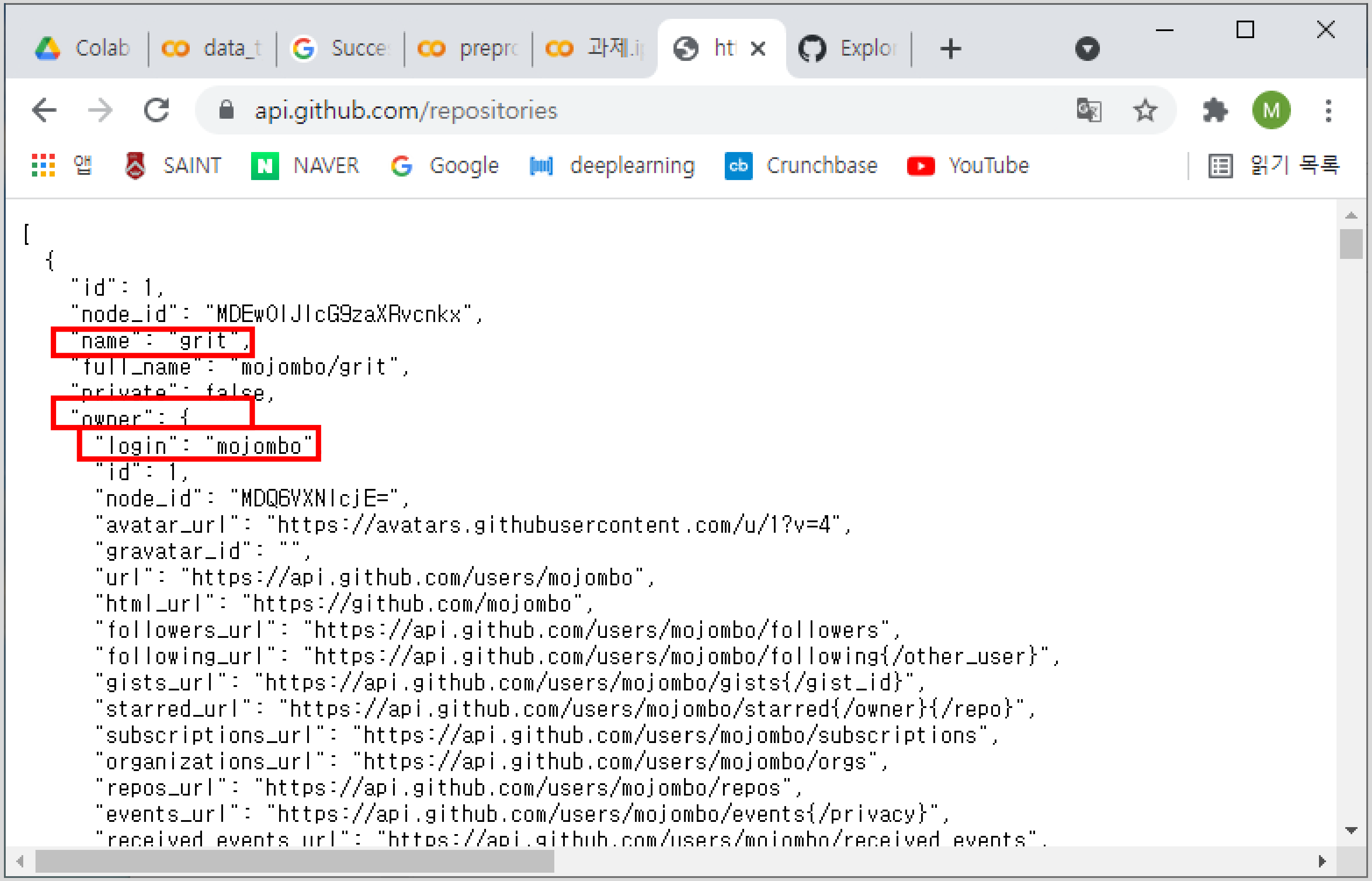
- 007 series summary table in JSON : Write it in [Wordpad] and write it as bondmovies.json save
- Format the JSON file : https://jsonformatter.curiousconcept.com Valid using check

```
{ bondmovies :
[
{
"title":" Goldfinger " , "director" : "Guy Hamilton" , "year" : 1964,
"genre" : ["Action" , "Adventure" , "Thriller" ],  "runtime" : "110 min" ,
"actor" : {"James Bond" : "Sean Connery" ,  "Bond Girl" : "Pussy Galore" },
"gross" : 51081062
},
{
"title" : "The Man with the Golden Gun" ,  "director" : "Guy Hamilton" , "year" : 1974,
"genre" : ["Action" , "Adventure" , "Thriller" ],  "runtime" : "123 min" ,
"actor" : {"James Bond" : "Roger Moore" ,  "Bond Girl" : "Britt Ekland" },
"gross" : 20972000
},
{
"title" : "License to Kill" ,  "director" : "John Glen" ,  "year" : 1989,
"genre" : ["Action" , "Adventure" ],  "runtime" : "133 min" ,
"actor" : {"James Bond" : "Timothy Dalton" , "Bond Girl" : "Carey Lowell" },
 "gross" : 51081062
}
]
}
```

Loading and saving JSON data

```
import urllib.request as req
import os.path, random
import json
# Preprocessing command to save file in colab ( set path )
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Download JSON data
url =" https://api.github.com/repositories "
savename = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/ repo .json "
if not os.path.exists ( savename ):
    req. urlretrieve ( url , savename )

# JSON interfaces
items = json.load (open( savename , "r", encoding="utf-8"))
# 또는
# s = open( savename , "r", encoding="utf-8").read()
#items = json.loads (s)

# 출력하기
for item in items:
print(item[" name "] + " - " + item[" owner "][" login "])
```

Loading and saving JSON data



```
[
  {
    "id": 1,
    "node_id": "MDEwOlJlcG9zaXRvcnkx",
    "name": "grit",
    "full_name": "mojombo/grit",
    "private": false,
    "owner": {
      "login": "mojombo",
      "id": 1,
      "node_id": "MDQ6VXNlcjE=",
      "avatar_url": "https://avatars.githubusercontent.com/u/1?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/mojombo",
      "html_url": "https://github.com/mojombo",
      "followers_url": "https://api.github.com/users/mojombo/followers",
      "following_url": "https://api.github.com/users/mojombo/following{/other_user}",
      "gists_url": "https://api.github.com/users/mojombo/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/mojombo/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/mojombo/subscriptions",
      "organizations_url": "https://api.github.com/users/mojombo/orgs",
      "repos_url": "https://api.github.com/users/mojombo/repos",
      "events_url": "https://api.github.com/users/mojombo/events{/privacy}",
      "received_events_url": "https://api.github.com/users/mojombo/received_events",
```

Loading and saving JSON data

```
# Create JSON file
#https://rfriend.tistory.com/474

student_data = {
"1.FirstName": " Gildong ",
"2.LastName": "Hong",
"3.Age": 20,
"4.University": " Yonsei University",
    "5.Courses": [
        {
            "Major": "Statistics",
            "Classes": ["Probability",
                        "Generalized Linear Model",
                        "Categorical Data Analysis"]
        },
        {
            "Minor": "ComputerScience",
            "Classes": ["Data Structure",
                        "Programming",
"Algorithms"]
}
]
}
```

Loading and saving JSON data

```
import json
st_json = json.dumps ( student_data )
print( st_json )

##### Save as JSON file
import json
# Preprocessing command to save file in colab ( set path )
from google.colab import drive
drive.mount ('/content/ gdrive ')

with open("/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/ student_file.json ", "w") as json_file :
    json.dump ( student_data , json_file )
```

Load and save CSV data

```
# Download from Kaggle : https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
from pandas as pd
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/IMDB Dataset.csv"
review = pd.read_csv ( file_name , engine="python")
review. head (10)

# Load all csv files in a folder
import os
from google.colab import drive
drive.mount ('/content/ gdrive ')

path = '/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/'
file_list = os.listdir (path)
file_list_py = [file for file in file_list if file.endswith ('.csv')] ## If the file name ends with .csv

## Import csv files into DataFrame and combine data ( concat )
import pandas as pd
df = pd.DataFrame ()
for i in file_list_py :
data = pd.read_csv (path + i )
    df = pd.concat ([ df,data ])

df = df.reset_index (drop = True)
Df
```

Load and save TXT data

```
# Sentiment classification using NRC terminology (NRC: classifying each word into 10 sentiments )
# Download NRC : from https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm
Click # (only the NRC Word-Emotion Association Lexicon)
Using NRC-Emotion-Lexicon-Wordlevel-v0.92.txt file

from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name1 = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/NRC-Emotion-Lexicon-Wordlevel-
v0.92.txt"

NRC = pd.read_csv (file_name1, engine="python", header=None, sep ="₩t")
NRC. head (20)

NRC = NRC[(NRC != 0).all(1)]
NRC. head (10)
# Column 0 : ( Applicable word ), Column 1 : (10 sentiments ), Column 2 : ( Applicable )
# Example ) abacus corresponds to the emotion of trust , and abadon corresponds to the three types of fear, negative,
and sadness .

# reset index number
NRC = NRC. reset_index (drop=True)
NRC. head (10)

list(NRC[0])
```