



## **SESSION #9** 자연어 처리

By Team 4  
정재근 빈다은 고병욱  
Date \_ 2017.12.30

# CONTENTS

---

**O1 N-gram & Part of Speech**

**O2 Word Cloud**

**O3 Journal**

**O4 Topic Modeling**

**O5 Topic Modeling +**

# # Terminology

- Natural Language Processing: 기계어(0,1)과 인간언어의 소통. 컴공 + 인공지능 + 전산언어학<sup>computational</sup> 관점에서 자연언어의 통계적인 모형과 논리적인 모형을 다루는 분야
- Data mining: 대규모 데이터 안에서 체계적이고 자동적으로 통계적 규칙이나 패턴을 찾아내는 것. Knowledge-discovery in database. 데이터 추출 -> 정제 -> 변경 -> 분석 -> 해석 ...
- Text mining: 비정형 데이터 마이닝 중 하나. 비/반정형 데이터에 대해 자연어 처리 기술/문서처리 기술을 적용하여 유용한 정보를 추출, 가공하기 위한 기술.
  - 텍스트로 이루어진 자료들을 분석하는 과정
  - 중요 부분, 주제, 숨은 의미와 관계 등을 (작가가 이를 표현하기 위해 사용한 문자 그대로를 모른 상태에서도) 알아내는 것이 목적

[https://www.ibm.com/support/knowledgecenter/en/SS3RA7\\_18.1.0/ta\\_guide\\_ddita/textmining/shared\\_entities/tm\\_intro\\_tm\\_defined.html](https://www.ibm.com/support/knowledgecenter/en/SS3RA7_18.1.0/ta_guide_ddita/textmining/shared_entities/tm_intro_tm_defined.html)

<https://cs.stackexchange.com/questions/7181/relation-and-difference-between-information-retrieval-and-information-extraction>

<http://iamdaisy.tistory.com/29> , <http://blog.embian.com/71>

# O1 NLP – N-gram model

- 언어에 대한 통계적 모델 중 하나, 텍스트 마이닝과 NLP에 자주 쓰임  
(그 다음 단어를 예측)
  - Google MSFT: (웹 상에서) 스펠링 체크, 텍스트 요약 / 음성인식
  - 장점: Simplicity 단순성, Scalability 확장성
- 
- <http://blog.ilkyu.kr/entry/%EC%96%B8%EC%96%B4-%EB%AA%A8%EB%8D%B8%EB%A7%81-ngram>
  - <http://www3.cs.stonybrook.edu/~ychoi/cse628/lecture/02-ngram.pdf>

2-gram / bi-gram	3-gram / tri-gram
<i>"The cow jumps over the moon".</i>	
The cow	The cow jumps
Cow jumps	Cow jumps over
Jumps over	Jumps over the
Over the	Over the moon

# O1 NLP – N-gram model

In [1]: `from __future__ import division`

`import math, random, re`

`from collections import defaultdict, Counter`

`from bs4 import BeautifulSoup`

`import requests`

In [2]: `def fix_unicode(text):`

`return text.replace(u"\u2019", "")`

텍스트 안의  
작은 따옴표들이  
사실 유니코드  
문자 u'\u2019'여서!

#Copy the code below

```
from __future__ import division
import math, random, re
from collections import defaultdict, Counter
from bs4 import BeautifulSoup
import requests
```

```
def fix_unicode(text):
    return text.replace(u"\u2019", "")
```

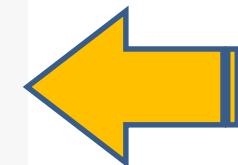
# O1 NLP – N-gram model

```
In [1]: from __future__ import division
import math, random, re
from collections import defaultdict, Counter
from bs4 import BeautifulSoup
import requests
```

```
In [2]: def fix_unicode(text):
    return text.replace(u"\u2019", "")
```

**Collection → Counter**

```
>>> # Tally occurrences of words in a list
>>> cnt = Counter()
>>> for word in ['r', 'b', 'r', 'g', 'b', 'b']:
...     cnt[word] += 1
>>> cnt
Counter({'b': 3, 'r': 2, 'g': 1})
>>> # Find the ten most common words in Hamlet
>>> import re
>>> words = re.findall(r'\w+', ('hamlet.txt').read().lower())
>>> Counter(words).most_common(10)
[('the', 1143), ('and', 966), ('to', 762), ('of', 669),
 ('i', 631), ('you', 554), ('a', 546), ('my', 514), ('hamlet', 471), ('in', 451)]
```



Counter는 나눌 수 있는 오브젝트를 세는 dictionary subclass이다. 각 원소가 key로, 그 원소의 카운트 횟수가 value로 이루어지는 사전형 집합을 만든다.

```
>>> c = Counter() # a new, empty counter
>>> c = Counter('gallahad') # a new counter from an iterable
>>> c = Counter({'red': 4, 'blue': 2}) # a new counter from a mapping
>>> c = Counter(cats=4, dogs=8)
```

<https://www.slideshare.net/dahlmoon/string-20160310>  
<https://docs.python.org/3/library/collections.html>  
<https://stackoverflow.com/questions/19212306/whats-the-difference-between-ascii-and-unicode>

# O1 NLP – N-gram model

```
In [1]: from __future__ import division
import math, random, re
from collections import defaultdict, Counter
from bs4 import BeautifulSoup
import requests
```

```
In [2]: def fix_unicode(text):
    return text.replace(u"\u2019", "'")
```

## Collections → defaultdict

### Defaultdict()

- 기본적으로 사전형 구조.
- 존재하지 않는 key 가 검색되면 자동적으로 default value를 반환 (오류내지않고)
- 사용할 때는( ) 안에 int, list 를 대입해야 한다. (함수가 아니라)

```
>>> from collections import defaultdict
>>> city_list = [('TX','Austin'), ('TX','Houston'), ('NY','Albany'), ('NY', 'Syracuse'), ('NY', 'Buffalo'), ('NY', 'Rochester'), ('TX', 'Dallas'), ('CA','Sacramento'), ('CA', 'Palo Alto'), ('GA', 'Atlanta')]
>>> cities_by_state = defaultdict(list)
>>> for state, city in city_list:
...     cities_by_state[state].append(city)
...
>>> for state, cities in cities_by_state.iteritems():
...     print state, ', '.join(cities) ...
```

NY Albany, Syracuse, Buffalo, Rochester  
 CA Sacramento, Palo Alto  
 GA Atlanta  
 TX Austin, Houston, Dallas

<https://www.slideshare.net/dahlmoon/string-20160310>  
<https://docs.python.org/3/library/collections.html>

<https://stackoverflow.com/questions/19212306/whats-the-difference-between-ascii-and-unicode>

# O1 NLP – N-gram model

```
In [1]: from __future__ import division
import math, random, re
from collections import defaultdict, Counter
from bs4 import BeautifulSoup
import requests
```

```
In [2]: def fix_unicode(text):
    return text.replace(u"\u2019", "")
```



## BeautifulSoup

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/#making-the-soup>

Beautiful Soup 4.4.0 documentation » index

**Table Of Contents**

- Beautiful Soup Documentation
  - Getting help
  - Quick Start
  - Installing Beautiful Soup
    - Problems after installation
    - Installing a parser
  - Making the soup
  - Kinds of objects
    - Tag
    - Name
    - Attributes
      - Multi-valued attributes
    - NavigableString

## Beautiful Soup Documentation

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

These instructions illustrate all major features of Beautiful Soup 4, with examples. I show you what the library is good for, how it works, how to use it, how to make it do what you want, and what to do when it violates your expectations.

The examples in this documentation should work the same way in Python 2.7 and Python 3.2.

You might be looking for the documentation for Beautiful Soup 3. If so, you should know that Beautiful Soup 3 is no longer being developed, and that Beautiful Soup 4 is recommended for all new projects. If you want to learn about the differences between Beautiful Soup 3 and Beautiful Soup 4, see [Porting code to BS4](#).

This documentation has been translated into other languages by Beautiful Soup users:

# O1 NLP – N-gram model

```
In [1]: from __future__ import division
import math, random, re
from collections import defaultdict, Counter
from bs4 import BeautifulSoup
import requests
```

```
In [2]: def fix_unicode(text):
    return text.replace(u"\u2019", "")
```



문자 집합 Character Set	인코딩 방식 Encoding Methods
ASCII, Unicode	UTF-8, UTF-16, UTF-32...

## fix\_unicode 배경 지식:

- 문자 인코딩 character encoding: 인코딩은 문자나 기호들의 집합을 컴퓨터에서 저장하거나 통신에 사용할 목적으로 부호화(e.g., byte sequence)하는 방법.  
(*You cannot save text to your hard drive as "Unicode". Unicode is an abstract representation of the text. You need to "encode" this abstract representation. That's where an encoding comes into play.*)
- 파이썬은 ASCII 인코딩을 기본으로 한다.
- UTF-8 방식이 대부분의 웹사이트 문자 인코딩 상태: 파이썬으로 가져올 때 **Unicode**에서 **ASCII**로 번역해주는 과정 때때로 필요
- <http://ssaemo.tistory.com/28>

# O1 NLP – N-gram model

In [3]: `def get_document():`

```
url = "https://www.oreilly.com/ideas/what-is-data-science"
html = requests.get(url).text
#print(html)
soup = BeautifulSoup(html, 'html.parser')
#html5lib or html.parser

content = soup.find("div", "article-body")
regex = r"[\w']+|[\.]"

document = []

for paragraph in content("p"):
    words = re.findall(regex, fix_unicode(paragraph.text))
    #print(words)
    document.extend(words)
#print(document)
return document
```

#url을 가져온다  
#html 코드를 요청해서 가져온 후,  
코드를 text로 변환  
# beautifulsoup 패키지를 이용해  
서 구조에 맞게 파싱한 뒤 저장.

#html 태그 중에서 내가 원하는  
글은 어느 태그에 속해있는가?  
#어떤 regular expression으로 정  
제할 수 있는가?

#빈 리스트를 형성해준다  
#리스트에 그 문단의 단어들을 다  
넣어준다. (각각 원소로)  
#리스트 반환

```
def get_document():

url = "https://www.oreilly.com/ideas/what-is-data-science"
html = requests.get(url).text
#print(html)
soup = BeautifulSoup(html, 'html.parser')
#html5lib or html.parser

content = soup.find("div", "article-body")          # find article-body div
regex = r"[\w']+|[\.]"                            # matches a word or a period

document = []

for paragraph in content("p"):
    words = re.findall(regex, fix_unicode(paragraph.text))
    #print(words)
    document.extend(words)
#print(document)
return document
```

# O1 NLP – N-gram model

```
In [3]: d
soup.title
# <title>The Dormouse's story</title>

soup.title.name
# u'title'

soup.title.string
# u'The Dormouse's story'

soup.title.parent.name
# u'head'

soup.p
# <p class="title"><b>The Dormouse's story</b></p>

soup.p['class']
# u'title'
```

{서 가져온 후,

|지를 이용해  
한 뒤 저장.내가 원하는  
해있는가?  
ssion으로 정|준다  
| 단어들을 다  
로)

```
def get_d
    soup.a
    url = I
    html = I
    #print(I
    soup = I
    #html5l:
    content
    regex =
    document
    for para
        words
        #print(words)
        document.extend(words)
    #print(document)
    return document
```

soup.find\_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
# <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

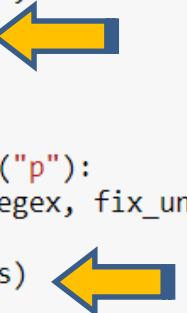
# O1 NLP – N-gram model

In [3]: `def get_document():`

```
url = "https://www.oreilly.com/ideas/what-is-data-science"
html = requests.get(url).text
#print(html)
soup = BeautifulSoup(html, 'html.parser')
#html5lib or html.parser

content = soup.find("div", "article-body")           # find article-body div
regex = r"[\w']+|[\.]"                            # matches a word or a period
document = []

for paragraph in content("p"):
    words = re.findall(regex, fix_unicode(paragraph.text))
    #print(words)
    document.extend(words)
#print(document)
return document
```



## REGEX

<https://docs.python.org/3/library/re.html#module-re>

₩w: alphanumeric chars 영숫자  
 [₩w'] = [a-zA-Z0-9\_] or wider...  
 [ ] → sets of characters

# # Regular Expression

/w

[Pages - Regular Expressions Tutorial](#)

[Back](#) | [Forward](#) | | [Previous](#) | [Next](#)

## Page 18

\w matches any word character ( alphanumeric plus "\_" ). In some languages these letter abbreviations are not recognized. Use character classes ("[A-zo-9\_]") instead (Case 5).

### Source

A1 B2 c3 d\_4 e:5 ffGG77--\_\_--

### Case 1

Regular Expression: \w  
 First match: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--  
 All matches: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--

### Case 2

Regular Expression: \w\*  
 First match: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--  
 All matches: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--

### Case 3

Regular Expression: [a-z]\w\*  
 First match: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--  
 All matches: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--

### Case 4

Regular Expression: \w{5}  
 First match: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--  
 All matches: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--

### Case 5

Regular Expression: [A-zo-9\_]  
 First match: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--  
 All matches: A1 B2 c3 d\_4 e:5 ffGG77--\_\_--

[Regular Expressions Tutorial](#)

keywords: [programming](#), [regexp](#), [regular expressions](#), [tutorial](#)

[Regular Expressions Tutorial](#)

keywords: [programming](#), [regexp](#), [regular expressions](#), [tutorial](#)

[http://zvon.org/comp/r/tut-Regexp.html#Pages~Page\\_10](http://zvon.org/comp/r/tut-Regexp.html#Pages~Page_10)

```
content = soup.find("div", "article-body")
regex = r"[\w]+|[\.]"
```

[Pages - Regular Expressions Tutorial](#)

[Back](#) | [Forward](#) | | [Previous](#) | [Next](#)

## Page 10

Alternating text can be enclosed in parentheses and alternatives separated with | .

### Source

Monday Tuesday Friday

### Case 2

Regular Expression: (on|ues|rida)  
 First match: Monday Tuesday Friday  
 All matches: Monday Tuesday Friday

### Case 2

Regular Expression: (Mon|Tues|Fri)day  
 First match: Monday Tuesday Friday  
 All matches: Monday Tuesday Friday

### Case 3

Regular Expression: ..(id|esd|nd)ay  
 First match: Monday Tuesday Friday  
 All matches: Monday Tuesday Friday

[Pages - Regular Expressions Tutorial](#)

[Back](#) | [Forward](#) | | [Previous](#) | [Next](#)

## Page 11

Quantifiers specify how many times a character can occur. Star \* (Case 1) matches zero or more times, plus + (Case 2) once or more times and question mark ? (Case 3) zero or once.

### Source

aabc abc bc

### Case 1

Regular Expression: a\*b  
 First match: aabc abc bc  
 All matches: aabc abc bc

### Case 2

Regular Expression: a+b  
 First match: aabc abc bc  
 All matches: aabc abc bc

### Case 3

Regular Expression: a?b  
 First match: aabc abc bc  
 All matches: aabc abc bc

[Regular Expressions Tutorial](#)

keywords: [programming](#), [regexp](#), [regular expressions](#), [tutorial](#)

## # Regular Expression

```

source = "Luke Skywalker 02-123-4567 luke@daum.net" # \w와 \w+의 차이

m1 = re.findall('\w', source) # 단어가 아니라 문자 단위로 출력
m2 = re.findall('\w+', source) # 단어 단위로 출력
print("m1 : ", m1)
print("m2 : ", m2)

>>> 출력결과
m1 :  ['L', 'u', 'k', 'e', 'S', 'k', 'y', 'w', 'a', 'r', 'k', 'e', 'r', '0', '2', '1', '2',&nbns
p; '3', '4', '5', '6', '7', 'l', 'u', 'k', 'e', 'd', 'a', 'u', 'm', 'n', 'e', 't']
m2 :  ['Luke', 'Skywalker', '02', '123', '4567', 'luke', 'daum', 'net']

```

정규패턴식을 보다보면

```
p = re.compile(r'\\section')
```

정규패턴식 앞에 **r**이 붙어 있는 경우가 많다. 파이썬 정규식에는 Raw string이라고 해서, 컴파일 해야 하는 정규식

이 Raw String(순수한 문자)임을 알려줄 수 있도록 문법이 있다.

만약 **p = re.compile('\\section')** 이라고 쓴다면 **\s**는 공백문자를 의미하는 **[ \t\n\r\f\v]**이 되어버려서 원하는 결과를 찾지 못한다. 그래서 #10번에서 배웠듯이 이스케이프 **\**를 활용해 **\\\\section**이라고 해주면 되지만, 파이썬은 특수하게 **r**을 사용하면 백슬래쉬를 1개만 써도 두개를 쓴 것과 같은 효과를 갖는다.

# O1 NLP – N-gram model

In [3]: `def get_document():`

```
url = "https://www.oreilly.com/ideas/what-is-data-science"
html = requests.get(url).text
#print(html)
soup = BeautifulSoup(html, 'html.parser')
#html5lib or html.parser

content = soup.find("div", "article-body")           # find article-body div
regex = r"[\w']+|[\.]"                            # matches a word or a period

document = []

for paragraph in content("p"):
    words = re.findall(regex, fix_unicode(paragraph.text))
    #print(words)
    document.extend(words)
#print(document)
return document
```



## extend / append 차이점

- `a = [1,2] a.append([3,4])`  
→ `[1, 2, [3, 4]]` # not `[1, 2, 3, 4]`
- `a = [1,2] a.extend([3,4])`  
→ `[1, 2, 3, 4]`

# O1 NLP – N-gram model

```
In [13]: document = get_document()
bigrams = zip(document, document[1:])
transitions= defaultdict(list)
for prev, current in bigrams: transitions[prev].append(current)
```

```
In [14]: def generate_using_bigrams(transitions):
    current = "." # this means the next word will start a sentence
    result = []
    while True:
        next_word_candidates = transitions[current] # bigrams (current, _)
        current = random.choice(next_word_candidates) # choose one at random
        result.append(current) # append it to results
        if current == ".": return " ".join(result) # if "." we're done
```



```
document = get_document()
bigrams = zip(document, document[1:])
transitions= defaultdict(list)
for prev, current in bigrams: transitions[prev].append(current)
```

```
def generate_using_bigrams(transitions):
    current = "." # this means the next word will start a sentence
    result = []
    while True:
        next_word_candidates = transitions[current] # bigrams (current, _)
        current = random.choice(next_word_candidates) # choose one at random
        result.append(current) # append it to results
        if current == ".": return " ".join(result) # if "." we're done
#print(generate_using_bigrams(transitions))
```

# O1 NLP – N-gram model

```
In [13]: document = get_document()
bigrams = zip(document, document[1:])
transitions= defaultdict(list)
for prev, current in bigrams: transitions[prev].append(current)
```



```
In [14]: def generate_using_bigrams(transitions):
    current = "." # this means the next word will start a sentence
    result = []
    while True:
        next_word_candidates = transitions[current] # bigrams (current, _)
        current = random.choice(next_word_candidates) # choose one at random
        result.append(current) # append it to results
        if current == ".": return " ".join(result) # if "." we're done
```

#zip은 입력되는 list 중에서 먼저 끝나는 쪽에 맞춰서 종료됨

```
#zip example (from stackoverflow)
numbers = [1,2,3]
letters = 'abcd'
zip(numbers, letters)
# [(1, 'a'), (2, 'b'), (3, 'c')]
```

# O1 NLP – N-gram model

In [13]:

```
document = get_document()
bigrams = zip(document, document[1:])
transitions= defaultdict(list)
for prev, current in bigrams: transitions[prev].append(current)
```

In [14]:

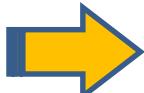
```
def generate_using_bigrams(transitions):
    current = "." # this means the next word will start a sentence
    result = []
    while True:
        next_word_candidates = transitions[current] # bigrams (current, _)
        current = random.choice(next_word_candidates) # choose one at random
        result.append(current) # append it to results
        if current == ".": return " ".join(result) # if ":" we're done
```

1. Document = ['a', 'b', 'c', ... ] 이런 형식으로 반환됨
  - 각 단어를 두개씩 쌍을 짜여주기 위해서 [0] 부터 시작하는 리스트와 [1]부터 시작하는 리스트의 원소들을 하나씩 뽑아서 묶어줌.
  - (0, 1) (1, 2) (2, 3) (3, 4) ....
2. Defaultdict(list)를 이용해서 빈 리스트를 생성 후, transitions에 저장. 즉 transitions는 빈 사전형 리스트.
3. Bigrams 를 for loop 을 돌려서 prev, current {0: [1], 1:[2], 2:[3] ... } 이렇게 넣어 줌.

# O1 NLP – N-gram model

```
In [13]: document = get_document()
bigrams = zip(document, document[1:])
transitions= defaultdict(list)
for prev, current in bigrams: transitions[prev].append(current)
```

```
In [14]: def generate_using_bigrams(transitions):
    current = "." # this means the next word will start a sentence
    result = []
    while True:
        next_word_candidates = transitions[current] # bigrams (current, _)
        current = random.choice(next_word_candidates) # choose one at random
        result.append(current) # append it to results
        if current == ".": return " ".join(result) # if "." we're done
```



1. Current = '.'으로 지정하는 이유 – 문장 첫 글자를 고르기 위함.  
 '.' : [~~~~~] 안에 있는 단어 중 하나를 골라서 그 단어를 current로 지정한다.  
 빈 리스트 result의 가장 첫 원소가 될 것.
2. Current가 다시 ":"이 될 때까지 계속 result를 다른 key : value [~~~] 안에서 뽑은 원소로  
 채워준다.  
 '.'에 도달하면 리스트 작성을 끝낸다.

# O1 NLP – N-gram model

```
In [75]: trigrams = zip(document, document[1:], document[2:])
trigram_transitions= defaultdict(list)
starts=[]
for prev, current, next in trigrams:
    if prev==".":
        starts.append(current)
    trigram_transitions[(prev, current)].append(next)
#print(trigram_transitions[(prev, current)])
```

```
In [76]: def generate_using_trigrams(starts, trigram_transitions):
    current = random.choice(starts) # choose a random starting word
    prev = "." # and precede it with a '.'
    result = [current]
    print(result)
    while True:
        next_word_candidates = trigram_transitions[(prev, current)]
        next = random.choice(next_word_candidates)

        prev, current = current, next
        result.append(current)

        if current == ".":
            return " ".join(result)
```

**trigram**

```
if current == ".":
    return " ".join(result)
```

```
#print(generate_using_trigrams(starts, trigram_transitions))
```

# O1 NLP – N-gram model

trigram

#Copy the code below

```
trigrams = zip(document, document[1:], document[2:])
trigram_transitions= defaultdict(list)
starts=[]
for prev, current, next in trigrams:
    if prev==".":
        starts.append(current)
    trigram_transitions[(prev, current)].append(next)
#print(trigram_transitions[(prev, current)])

def generate_using_trigrams(starts, trigram_transitions):
    current = random.choice(starts)      # choose a random starting word
    prev = "."                            # and precede it with a '.'
    result = [current]
    while True:
        next_word_candidates = trigram_transitions[(prev, current)]
        next = random.choice(next_word_candidates)

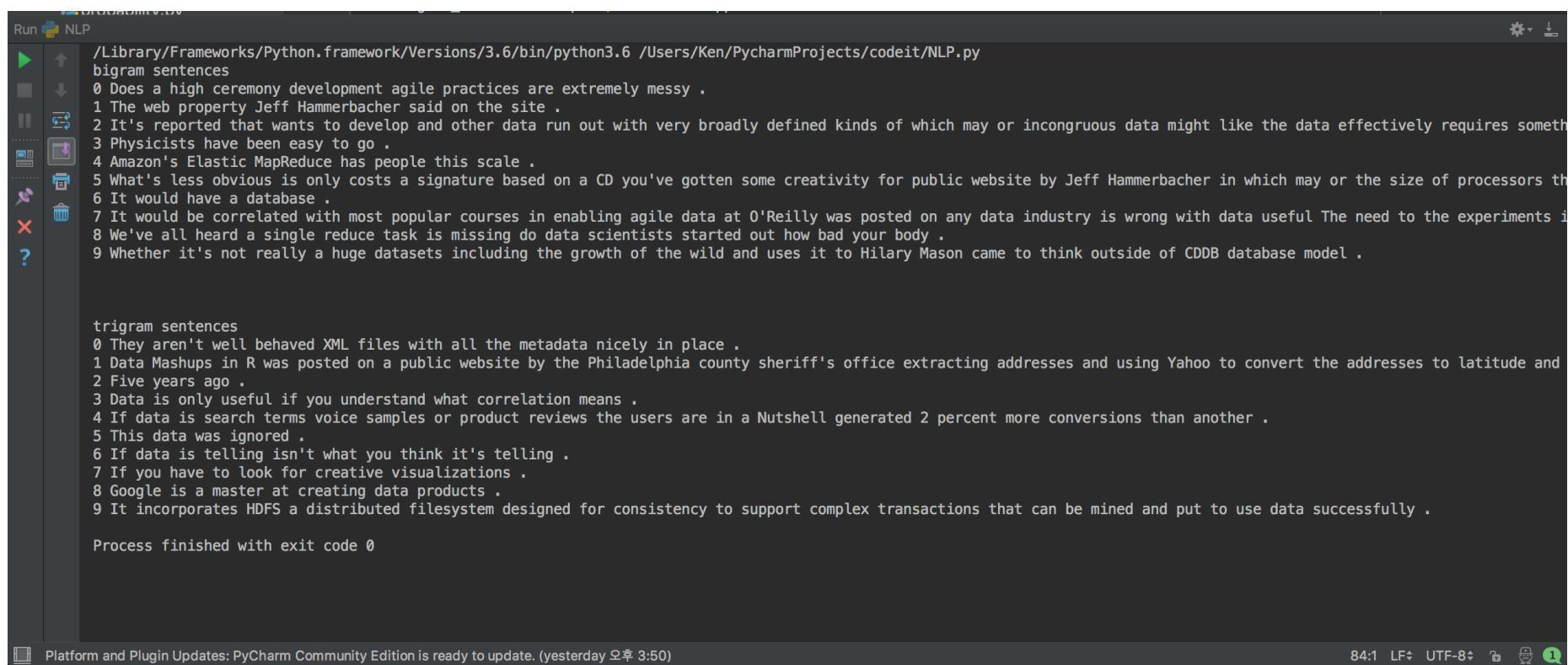
        prev, current = current, next
        result.append(current)

        if current == ".":
            return " ".join(result)

#print(generate_using_trigrams(starts, trigram_transitions))
```

# O1 NLP – N-gram model

result



The screenshot shows a PyCharm IDE window with a dark theme. The top bar displays the title 'SESSION #9 < NLP >' and the file name 'O1 NLP – N-gram model'. The main editor area contains two sections of processed text:

```
Run NLP
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/Ken/PycharmProjects/codeit/NLP.py
bigram sentences
0 Does a high ceremony development agile practices are extremely messy .
1 The web property Jeff Hammerbacher said on the site .
2 It's reported that wants to develop and other data run out with very broadly defined kinds of which may or incongruous data might like the data effectively requires someth
3 Physicists have been easy to go .
4 Amazon's Elastic MapReduce has people this scale .
5 What's less obvious is only costs a signature based on a CD you've gotten some creativity for public website by Jeff Hammerbacher in which may or the size of processors th
6 It would have a database .
7 It would be correlated with most popular courses in enabling agile data at O'Reilly was posted on any data industry is wrong with data useful The need to the experiments i
8 We've all heard a single reduce task is missing do data scientists started out how bad your body .
9 Whether it's not really a huge datasets including the growth of the wild and uses it to Hilary Mason came to think outside of CDBB database model .

trigram sentences
0 They aren't well behaved XML files with all the metadata nicely in place .
1 Data Mashups in R was posted on a public website by the Philadelphia county sheriff's office extracting addresses and using Yahoo to convert the addresses to latitude and
2 Five years ago .
3 Data is only useful if you understand what correlation means .
4 If data is search terms voice samples or product reviews the users are in a Nutshell generated 2 percent more conversions than another .
5 This data was ignored .
6 If data is telling isn't what you think it's telling .
7 If you have to look for creative visualizations .
8 Google is a master at creating data products .
9 It incorporates HDFS a distributed filesystem designed for consistency to support complex transactions that can be mined and put to use data successfully .

Process finished with exit code 0
```

The status bar at the bottom indicates: Platform and Plugin Updates: PyCharm Community Edition is ready to update. (yesterday 오후 3:50) 84:1 LF UTF-8 ⓘ

## 02 Wordcloud (Tag cloud, Weighted list)

## 단어 데이터 시각화 예시

- 단어의 출현 빈도수
  - 단어의 주제
  - 단어들끼리의 연관성
  - 해당 단어 다음에 나올 확률이 높은 다른 단어



## Word tree



## Ego-network



## Word cloud

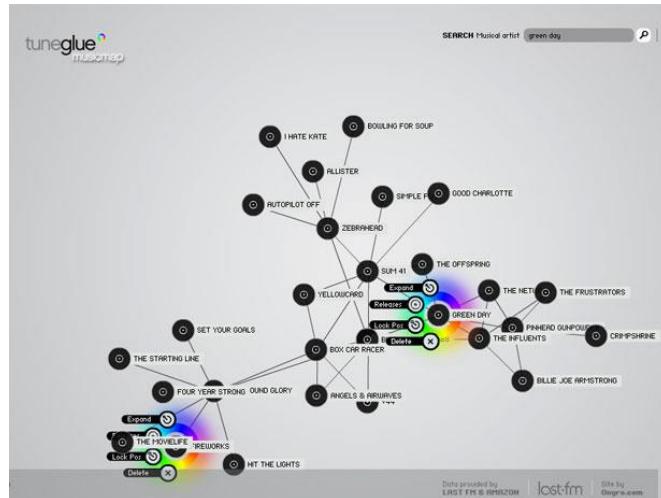
How many

the cannibals fly Before they're forever banned? The answer, my friend, is blowin' in the wind

times  
must  
years can  
can a man turn his head Pretending he just doesn't see? The answer, my friend, is blowin' in the wind

a man look up Before he can see the sky? Yes, 'n how many ears must a man have Before he can hear people cry? Yes, 'n how many deaths will it take till he knows That too many people have died? The answer, my friend, is blowin' in the wind

a mountain exist Before it's washed to the sea? Yes, 'n how many years can some people exist Before they're allowed to die? Yes, 'n how many times must a white dove sail Before seas must a white dove sail Before she sleeps in the sand? Yes, 'n how many times must the cannibals fly Before ears must one man have Before he can hear people cry? Yes, 'n how many deaths will it take till he knows That too many people have died? The answer, my friend, is blowin' in the wind



A word cloud visualization centered around the words "data" and "science". The word "data" is the largest word in the center, colored purple. Surrounding it are various other words related to research, education, and technology, such as "researchers", "project", "faculty", "support", "education", "bridge", "graduate", "research", "activity", "sciences", "fellow", "engineering", and "environment". The words are in different colors (purple, green, yellow, red, blue) and sizes, representing their frequency or importance in the context.

## 02 Wordcloud (Tag cloud, Weighted list)



## 02 Wordcloud (Tag cloud, Weighted list)

시각적 특징

Tag size: 크기가 큰 단어일수록 이목을 끔

Scanning: 보통 워드클라우드를 볼때 '스캔' 할 뿐 자세히 읽지 않음

Centering: 중앙에 위치한 단어일수록 이목을 끔

Position: 왼쪽 위에 위치한 단어일수록 이목을 끔

Exploration: 특정 단어를 찾기 용이한 구조는 아님

단점

단어의 위치는 임의로 결정되어, 좌표와 데이터 무관

-> 단어의 위치도 어떤 의미를 담는 Wordcloud 만들기

# 02 Wordcloud (Tag cloud, Weighted list)

## 단어의 위치에 의미가 담긴 Wordcloud

(책 예시)

X좌표:  
구인광고에 해당 단어  
가 등장하는 빈도수

Y좌표:  
이력서에 해당 단어가  
등장하는 빈도수

단어의 크기: 두 빈도  
수를 합친 것에 비례하  
게 설정

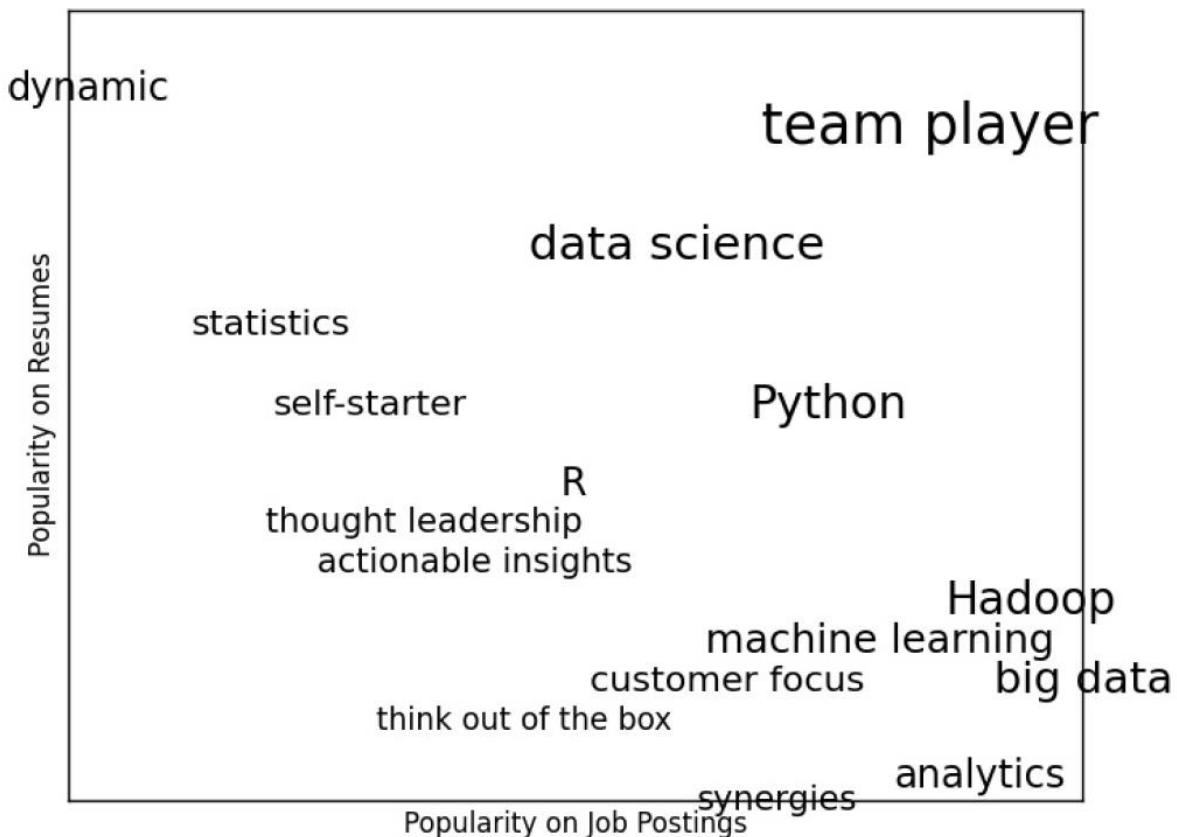


Figure 20-2. A more meaningful (if less attractive) word cloud

# 02 Wordcloud (Tag cloud, Weighted list)

## 단어의 위치에 의미가 담긴 Wordcloud

(책 예시)

X좌표:

구인광고에 해당 단어  
가 등장하는 빈도수

Y좌표:

이력서에 해당 단어가  
등장하는 빈도수

단어의 크기: 두 빈도  
수를 합친 것에 비례하  
게 설정

```
from __future__ import division
from matplotlib import pyplot as plt

data = [ ("big data", 100, 15), ("Hadoop", 95, 25), ("Python", 75, 50),
("R", 50, 40), ("machine learning", 80, 20), ("statistics", 20, 60),
("data science", 60, 70), ("analytics", 90, 3),
("team player", 85, 85), ("dynamic", 2, 90), ("synergies", 70, 0),
("actionable insights", 40, 30), ("think out of the box", 45, 10),
("self-starter", 30, 50), ("customer focus", 65, 15),
("thought leadership", 35, 35)]
```

```
def text_size(total):
    """equals 8 if total is 0, 28 if total is 200"""
    return 8 + total / 200 * 20
```

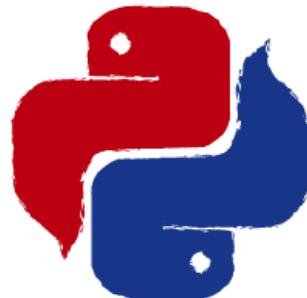
```
for word, job_popularity, resume_popularity in data:
    plt.text(job_popularity, resume_popularity, word,
            ha='center', va='center', size=text_size(job_popularity + resume_popularity))

plt.xlabel("Popularity on Job Postings")
plt.ylabel("Popularity on Resumes")
plt.axis([0, 100, 0, 100])
plt.show()
```

# 02 Wordcloud (Tag cloud, Weighted list)

## KoNLPy

- 한국어로 된 텍스트를 형태소 분석/품사 태깅하는 모듈들을 제공하는 패키지



## KoNLPy

Star 547

KoNLPy is a Python package  
for Korean natural languages

Note:

You are not using the most up to date version of the library. [0.4.4](#) is the newest version.

## KoNLPy: 파이썬 한국어 NLP

[build failing](#) [docs passing](#)

KoNLPy(“코엔엘파이”라고 읽습니다)는 한국어 정보처리를 위한 파이썬 패키지입니다. 설치  
법은 [이 곳을](#) 참고해주세요.

NLP를 처음 시작하시는 분들은 [시작하기](#)  
KoNLPy의 사용법 가이드는 [사용하기](#), 각  
다.

KoNLPy	Mecab-ko	Py th o n	GPL v 3	박은경(서울 대)
	한나눔			
	꼬꼬마			
	Komoran			
	Twitter			
UMorpheme		MIT	김경준(UNI ST)	shineware
Komoranpy				

## 02 Wordcloud (Tag cloud, Weighted list)

- 형태소 분석
  - 형태소를 비롯하여, 어근, 접두사/접미사, 품사(POS, part-of-speech) 등 다양한 언어적 속성의 구조를 파악하는 것.
  - 언어적 모호성을 해결하기 위해 통계 정보 활용

ex. “아버지가방에들어가신다”

“**나는** 눈을 떴다”, “하늘을 **나는** 새 ”

## 02 Wordcloud (Tag cloud, Weighted list)

- 품사 태깅 (POS) :

Hannanum	Kkma	Komoran	Mecab	Twitter
아버지가방에 들어가 / N	아버지 / NNG	아버지가방에 들어가신다 / NNP	아버지 / NNG	아버지 / Noun
이 / J	가방 / NNG		가 / JKS	가방 / Noun
시느다 / E	에 / JKM		방 / NNG	에 / Josa
	들어가 / VV		에 / JKB	들어가신 / Verb
	시 / EPH		들어가 / VV	다 / Eomi
	느다 / EFN		신다 / EP+EC	

# 02 Wordcloud (Tag cloud, Weighted list)

- 품사 태깅 (POS) :

Twitter Korean Text (ntags=19)		Komoran (ntags=42)		Mecab-ko (ntags=43)	
Tag	Description	Tag	Description	Tag	Description
Noun	명사 (Nouns, Pronouns, Company)	NNG	일반 명사	NNG	일반 명사
		NNP	고유 명사	NNP	고유 명사
		NNB	의존 명사	NNBC	단위를 나타내는 명사
		NR	수사	NR	수사
		NP	대명사	NP	대명사

## 02 Wordcloud (Tag cloud, Weighted list)

- 코드 예시

```
In [1]: from konlpy.tag import Kkma
kkma = Kkma()

In [2]: kkma.sentences('한국어 분석을 시작합니다 재미있어요~~')
Out[2]: ['한국어 분석을 시작합니다', '재미있어요~~']

In [3]: kkma.nouns('한국어 분석을 시작합니다 재미있어요~~')
Out[3]: ['한국어', '분석']

In [4]: kkma.pos('한국어 분석을 시작합니다 재미있어요~~')
Out[4]: [('한국어', 'NNG'),
          ('분석', 'NNG'),
          ('을', 'JKO'),
          ('시작하', 'VV'),
          ('ㅂ니다', 'EFN'),
          ('재미있', 'VA'),
          ('어요', 'EFN'),
          ('~~', 'SW')]
```

# 03 문법

## 유한문법 기반의 문장 생성

문법 : 언어 모델을 구축하는 또 다른 방법

```
grammar = {
    "_S" : ["_NP _VP"],
    "_NP" : ["_N",
              "_A _NP _P _A _N"],
    "_VP" : ["_V",
              "_V _NP"],
    "_N" : ["data science", "Python", "regression"],
    "_A" : ["big", "linear", "logistic"],
    "_P" : ["about", "near"],
    "_V" : ["learns", "trains", "tests", "is"]
}
```

\_S : 문장(Sentence)의 규칙  
 \_NP : 명사구(Noun Phrase)의 규칙  
 \_VP : 동사구(Verb Phrase)의 규칙

문법 규칙들은 재귀적이기 때문에,  
 무한히 많은 규칙 생성 가능

# 03 문법

## 유한문법 기반의 문장 생성

### 결과값

```
grammar = {
    "_S" : ["_NP _VP"],
    "_NP" : ["_N",
              "_A _NP _P _A _N"],
    "_VP" : ["_V",
              "_V _NP"],
    "_N" : ["data science", "Python", "regression"],
    "_A" : ["big", "linear", "logistic"],
    "_P" : ["about", "near"],
    "_V" : ["learns", "trains", "tests", "is"]
}
```

```
['_S']
['_NP', '_VP']
['_N', '_VP']
['Python', '_VP']
['Python', '_V', '_NP']
['Python', 'trains', '_NP']
['Python', 'trains', '_A', '_NP', '_P', '_A', '_N']
['Python', 'trains', 'logistic', '_NP', '_P', '_A', '_N']
['Python', 'trains', 'logistic', '_N', '_P', '_A', '_N']
['Python', 'trains', 'logistic', 'data science', '_P', '_A', '_N']
['Python', 'trains', 'logistic', 'data science', 'about', '_A', '_N']
['Python', 'trains', 'logistic', 'data science', 'about', 'logistic', '_N']
['Python', 'trains', 'logistic', 'data science', 'about', 'logistic', 'Python']
```

# 03 문법

## 유한문법 기반의 문장 생성

각 항목을 대체 가능한 다른 항목 또는 항목들로 변환시키는 함수.

함수의 규칙 항목 찾기

if cannot, 종결어로 구성되어 있음

if can, 대체할 수 있는 여러 항목 중 하나를 임의로 선택

```
def is_terminal(token):
    return token[0] != "_"
```

특정 항목이 종결어인지 아닌지  
판단하는 함수

```
def expand(grammar, tokens):
    for i, token in enumerate(tokens):
```

```
# ignore terminals
if is_terminal(token): continue
```

종결어 건너뛰기

```
# choose a replacement at random
replacement = random.choice(grammar[token])
```

종결어가 아닌 단어라면 대체 항목  
을 임의로 선택

# 03 문법

## 유한문법 기반의 문장 생성

```
if is_terminal(replacement):
    tokens[i] = replacement
else:
    tokens = tokens[:i] + replacement.split() + tokens[(i+1):]
return expand(grammar, tokens)
```

새로운 단어의 list에 expand 적용

```
# if we get here we had all terminals and are done
return tokens
```

모든 단어가 종결어일 때 종료

---

```
def generate_sentence(grammar):
    return expand(grammar, ["_S"])
```

문장 생성하는 함수

# 03 문법

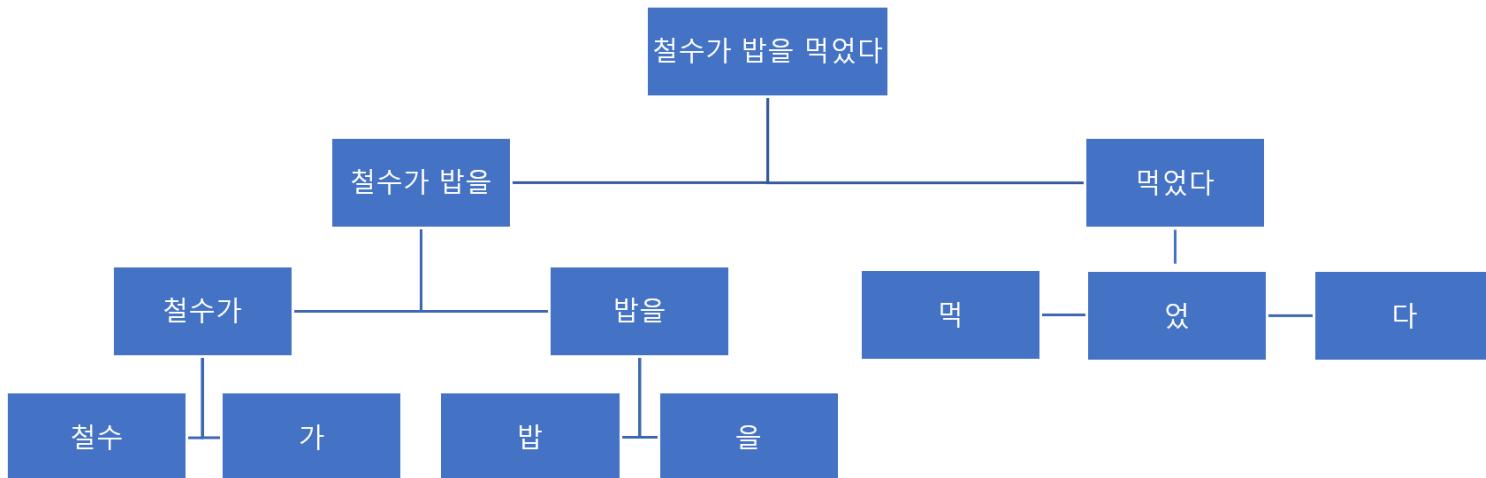
■ 문법을 기반으로 텍스트 이해하기  
- 한국어 기본문의 파싱(**Parsing**)

‘철수가 밥을 먹었다’

# 03 문법

## 문법을 기반으로 텍스트 이해하기 - 한국어 기본문의 파싱(Parsing)

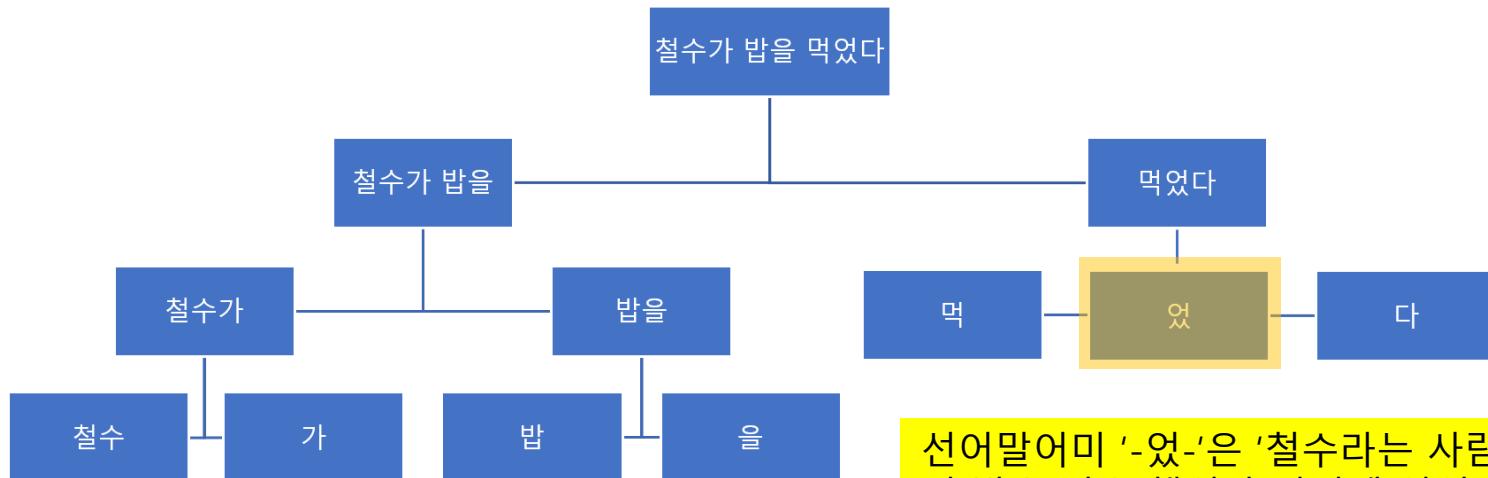
‘철수가 밥을 먹었다’



# 03 문법

## 문법을 기반으로 텍스트 이해하기 - 한국어 기본문의 파싱(Parsing)

‘철수가 밥을 먹었다’



선어말어미 ‘-었-’은 ‘철수라는 사람  
이 밥을 먹은 행위가 과거에 있었다’  
는 의미를 나타냄

→ ‘-었-’은 어간 ‘먹-’이 아니라 ‘철  
수가 밥을 먹-’이라는 큰 단위와 결  
합했다고 보는게 자연스러움

# 03 문법

## 문법을 기반으로 텍스트 이해하기 - 한국어 기본문의 파싱(Parsing)

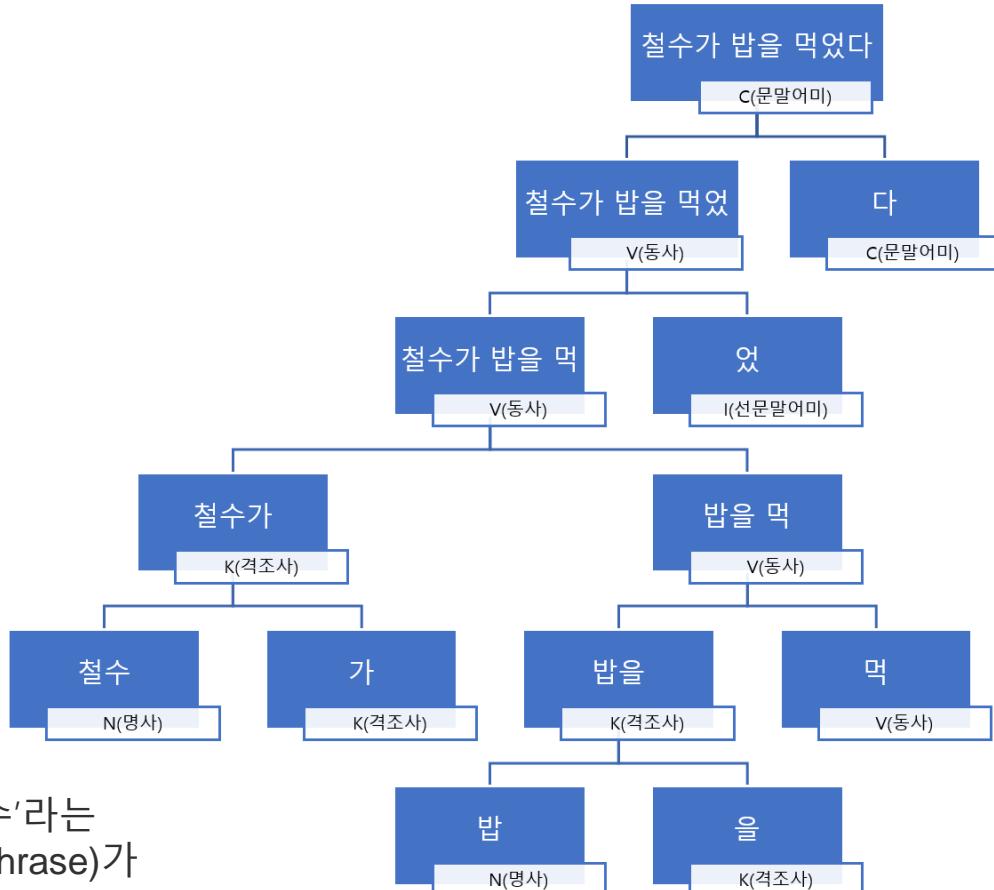
‘철수가 밥을 먹었다’

### 핵(head)

명사, 동사, 조사, 어미 등 개별 문법요소들은 위계적 구조를 가지고 결합

두 통사원자가 결합하여 이루는 상위 단위의 통사적(문법적) 성격은 **하위 요소의 통사적 성격에 의해 결정됨**. 이 때 둘 중 어느 한 요소가 중심적 역할을 맡게 되는데, 그 중심 역할을 하는 요소를 ‘핵’이라 함

Ex. 박진호(1994)에 따르면 위 문장에서 ‘철수’라는 명사는 격조사 ‘-가’와 결합해 (주)격조사구(phrase)가 됨 → 격조사 ‘-가’가 핵



# 03 문법

## 문법을 기반으로 텍스트 이해하기 - 한국어 기본문의 파싱(Parsing)

박진호(1994)는 한국어 통사원자의 범주를 8개로 나누고 그 기호를 아래와 같이 정의

명사	동사	격조사	문말어미	보조사	선문말어미	관형사	부사
N	V	K	C	D	I	ADN	ADV

- KoNLPy 는?
- 한국어로 된 텍스트를 형태소 분석/품사 태깅하는 모듈들을 제공하는 패키지
- 형태소를 비롯하여, 어근, 접두사/접미사, 품사(POS, part-of-speech) 등 다양한 언어적 속성의 구조를 파악하는 것
- 언어적 모호성을 해결하기 위해 통계 정보 활용

Ex. “아버지가 방에 들어가신다”

“나는 눈을 떴다”, “하늘을 나는 새 ”

# 03 문법

## 문법을 기반으로 텍스트 이해하기 - 한국어 기본문의 파싱(Parsing)

- 품사 태깅 (POS) :

		Hannanum	Kkma	Komoran	Mecab	Twitter
KoNLPy	Mecab-ko	아버지가방 에들어가 / N	아버지 / N NG	아버지가방 에들어가신 다 / NNP	아버지 / N NG	아버지 / N oun
	한나눔					
	꼬꼬마	이 / J	가방 / NN G		가 / JKS	가방 / Nou n
	Komoran					
	Twitter	시느다 / E	에 / JKM		방 / NNG	에 / Josa
UMorpheme			들어가 / V V		에 / JKB	들어가신 / Verb
Komoranpy			시 / EPH		들어가 / V V	다 / Eomi
			ㄴ다 / EFN		신다 / EP+ EC	

# 03 Journal – Topic Analysis(Modeling)

## Interest-based Customer Segmentation Methodology Using Topic Modeling (현윤진, 김나규, 조윤호, 2015)

토픽 분석을 활용한 관심 기반 고객 세분화 방법론

현윤진\* · 김남규\*\* · 조윤호\*\*\*

Interest-based Customer Segmentation Methodology  
Using Topic Modeling

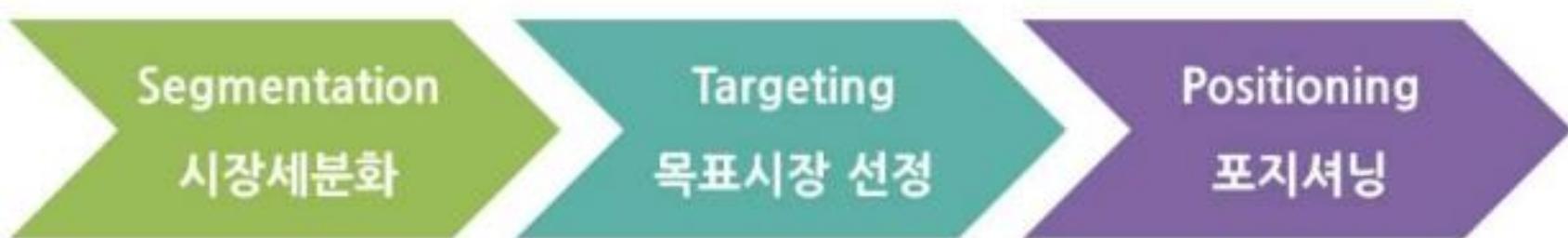
Yoonjin Hyun\* · Namgyu Kim\*\* · Yoonho Cho\*\*\*

### Abstract

As the range of the customer choice becomes more diverse, the average life span of companies' products and services is becoming shorter. Most companies are striving to maximize the revenue by understanding the customer's needs and providing customized products and services. However, companies had to bear a significant burden, in terms of the time and cost involved in the process of determining each individual customer's needs. Therefore, an alternative method is employed that involves grouping the customers into different categories based on certain criteria and establishing a marketing strategy tailored for each group. In this way, customer segmentation and customer clustering are performed using demographic information and behavioral information. Demographic information included sex, age, income level, and etc., while behavioral information was usually identified indirectly through customers' purchase history and search history. However, there is a limitation regarding companies' customer behavioral information, because the information is usually obtained through the limited data provided by a customer on a company's website. This is because the pattern indicated when a customer accesses a particular site might not be representative of the general tendency of that customer. Therefore, in this study, rather than the pattern indicated through a particular site, a customer's interest is identified using that customer's access record pertaining to external news. Hence, by utilizing this method, we proposed a methodology to perform customer segmentation. In addition, by extracting the main issues through a topic analysis covering approximately 3,000 Internet news articles, the actual experiment applying customer segmentation is performed and the applicability of the proposed methodology is analyzed.

# 03 Journal – Topic Analysis(Modeling)

## STP Strategy



- **STP 전략(모델)**
  - 일정한 기준에 의해 전체 시장을 나누고 (**Segmentation**)
  - 기업과 제품(혹은 서비스)에 가장 적당한 시장을 타겟 삼아 (**Targeting**)
  - 시장에서 어떠한 위치를 선정하여 소비자에게 다가가는(**Positioning**)
  - 마케팅 전략

# 03 Journal – Topic Analysis(Modeling)

## Market Segmentation

### Market Segmentation

- **시장 세분화** : 소비자들을 집단 내에서는 제품에 대한 욕구와 구매행동이 서로 유사하고 집단 간에는 상이하도록 군집화하는 과정
- **주요 세분화 가능 변수**
  - **지리적 변수** : 국가, 지역, 온라인, 오프라인
  - **인구 통계적 변수** : 나이, 성별, 연령, 직업, 소득, 가족상황
  - **심리 도식적 변수** : 라이프 스타일, 사회계층, 성격유형
    - 지리, 인구통계적 변수에 비해 추상적  
-> 측정이 어려움 + 세분화된 고객에 대한 접근 가능성이 낮음
    - 그러나 **소비자 행동을 근원적으로 설명해 줄 수 있는 변수임**
  - **인지 및 행동적 변수** : 제품 및 서비스에 대한 태도, 이용률, 충성도, 구매패턴
    - 고객의 구매 행동을 직접적으로 나타냄 + 실질적인 구매 행동과 밀접한 정보
    - **고객을 세분화 하는데 있어서 가장 효과적인 변수**

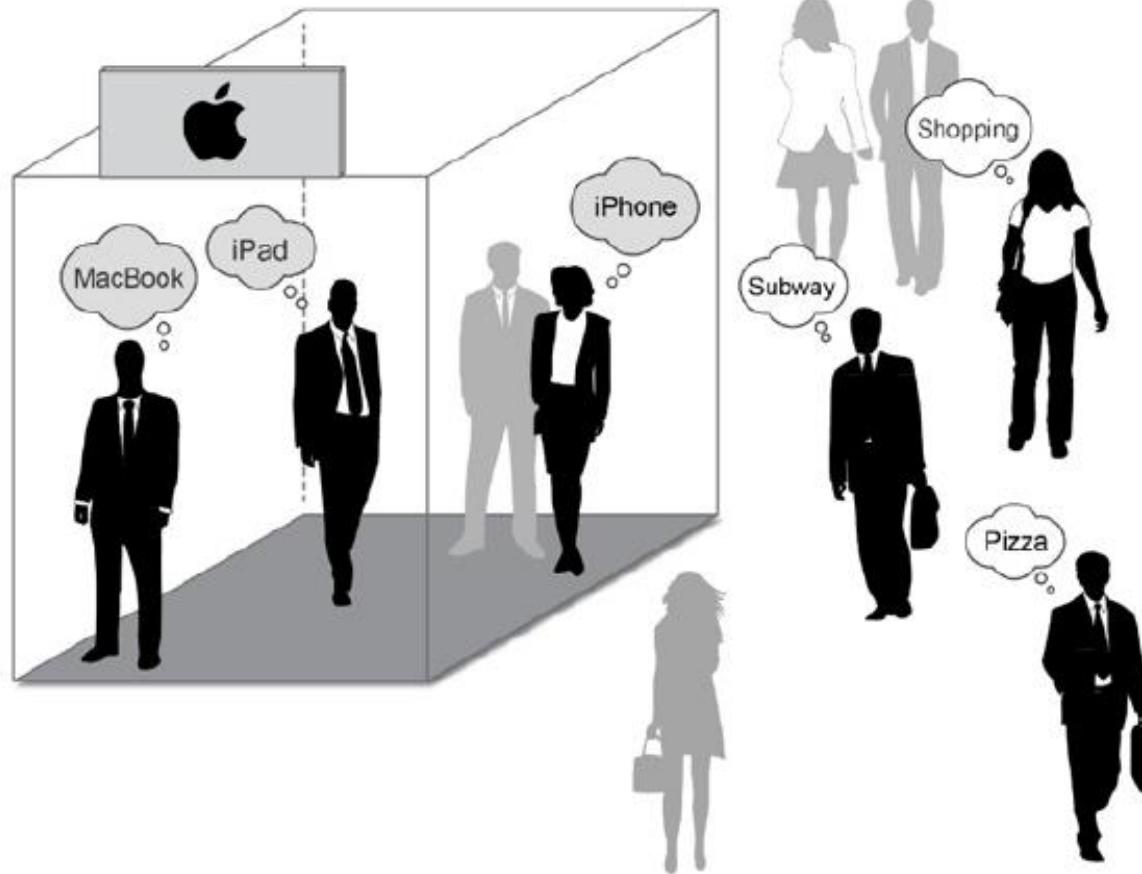
# 03 Journal – Topic Analysis(Modeling)

## Customer Segmentation

### Customer Segmentation

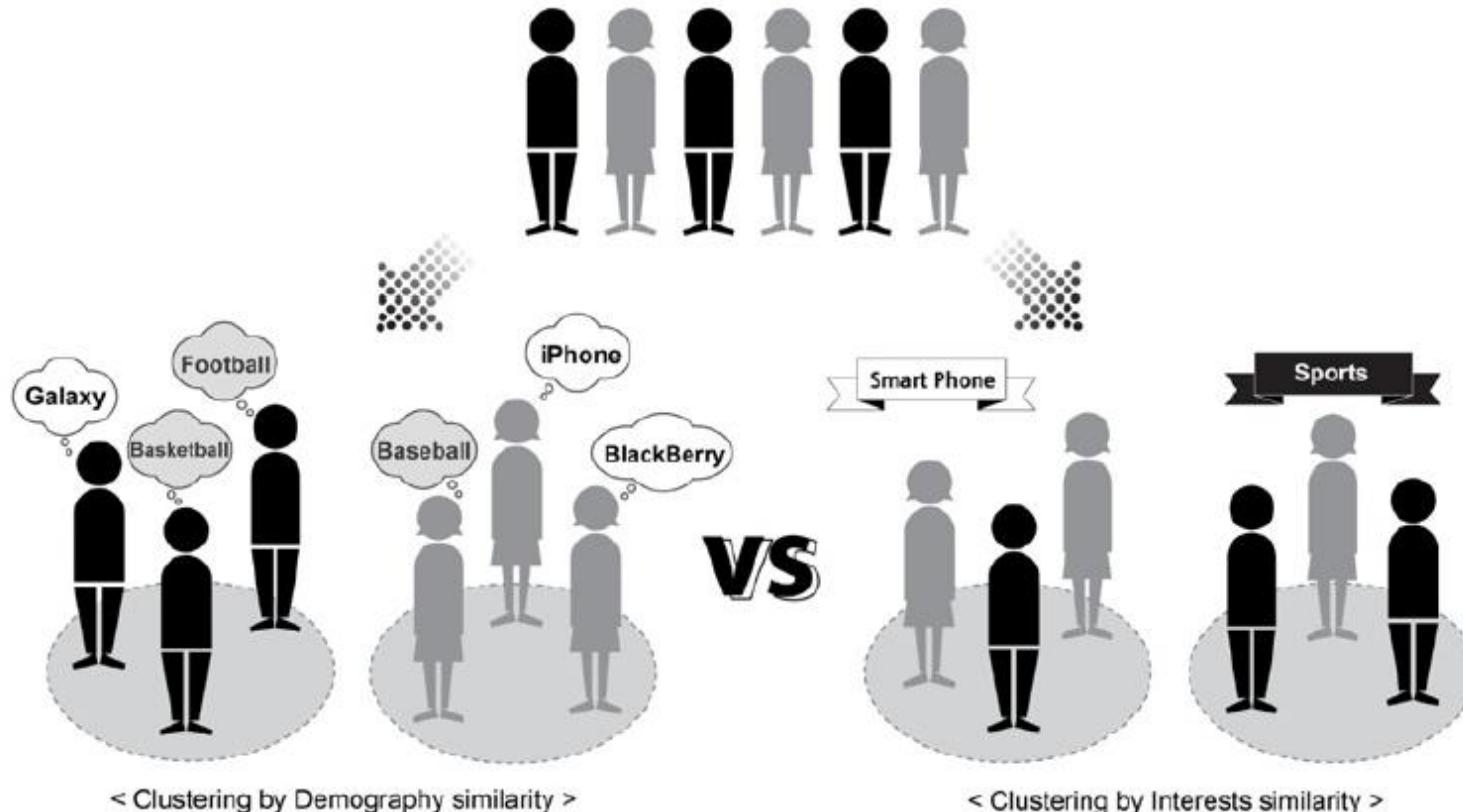
- 고객 세분화(고객 클러스터링)
  - 기술의 발전으로 인해 소비자 각각에 대한 데이터를 수집하고 해석하는 것이 가능해짐으로써 시장이 아닌 소비자들을 동질적인 특성을 가진 집단으로 나눌 수 있게 됨
  - 각 기업들이 보유한 각 고객의 구매 내역, 검색 내역 등의 고객 행태정보를 활용
- 단점
  - ① 개별 기업이 보유하고 있는 고객의 행태 정보는 자사의 사이트 / 서비스 내에서 보이는 구매 및 검색 내역에 국한되게 됨
    - > 고객의 실제 관심의 동질성이 아닌 해당 사이트 / 서비스 내에서 표시된 관심의 동질성에 기반하여 고객 클러스터링이 이루어짐

# 03 Journal – Topic Analysis(Modeling)



<그림 1> 실제 관심과 특정 사이트 내에서 표출되는 관심과의 차이

# 03 Journal – Topic Analysis(Modeling)



〈그림 2〉 클러스터링 기준에 따른 결과의 차이

# 03 Journal – Topic Analysis(Modeling)

## Topic Analysis(Modeling)

### Topic Analysis

#### – 토픽 분석(Topic Analysis / Modeling)

: 텍스트 마이닝(Text Mining)\*의 대표적인 분석기법.

각 문서에 포함된 용어의 빈도수에 근거하여 유사 문서를 그룹화한 뒤, 각 그룹을 대표하는 주요 용어들을 추출해 해당 그룹의 토픽 키워드 집합을 제시하는 방식

#### – \* 텍스트 마이닝

- 기술 발전 및 인터넷 확산으로 웹과 다양한 SNS를 통해 쌓인 방대한 텍스트 형태의 비정형 데이터를 분석하여 이전에는 찾을 수 없었던 새롭고 의미 있는 정보를 추출하는 과정

- **기법**

① 전통 : 연관관계 분석, 분류, 군집화

② 최근 : 자연어 처리, 정보검색, 전산 언어학, 토픽 추적, 텍스트 범주화 등의 종합적 활용

③ 특히 자연어 처리기술은 텍스터트 마이닝 분석 결과의 성패를 좌우하는 핵심기술

# 03 Journal – Topic Analysis(Modeling)

## Topic Analysis(Modeling)

### Topic Analysis

- 토픽 분석(Topic Analysis / Modeling)의 이론적 배경
  - ① 벡터 공간 모델
    - A. 텍스트를 표현하는 기본적인 방법
    - B. 분석 목적에 따라 행렬, 계층, 벡터 등의 다양한 형태로도 표현이 가능
  - ② TF-IDF(Term Frequency – Inverse Document Frequency)
    - A. 여러 문서에서 자주 출현하는 일반적인 단어는 가중치를 낮게 부여
    - B. 특정 문서에만 출현하는 비일반적인 단어의 가중치는 높게 부여하는 계산 방식
    - C. TF-IDF를 값으로 갖는 벡터로 표현되는데, 문서 내 존재하는 용어의 수가 너무 방대하기 때문에 SVD(Singular Value Decomposition) 등을 활용한 차원 축소가 이루어지게 됨
- 이러한 과정을 통해 비정형 텍스트 문서의 구조화가 완료되면 이후 과정에서 정형 데이터와 함께 텍스트 데이터에 대한 군집화, 예측 등의 작업이 가능해진다.

# 03 Journal – Topic Analysis(Modeling)

## Topic Analysis(Modeling)

### Topic Analysis

- 토픽 분석(Topic Analysis / Modeling)의 활용 예시

- ① 소셜 메트릭스

- A. 다음소프트의 소셜 미디어 분석 솔루션
  - B. 문맥 중심의 텍스트 마이닝 작업을 통해 각 데이터의 출현 원인 및 다른 데이터들과의 관계를 도식화하여 제공

- ② 버즈 인사이트 바이럴 지수(BVI) by 와이즈넛

- A. 트위터, 페이스북, 블로그, 카페 등에 올라온 대선 후보 관련 버즈 분석
  - B. 여러 문서에서 자주 출현하는 일반적인 단어는 가중치를 낮게

- ③ 브람스(Brams) by 다이퀘스트

- A. 실시간으로 사용자의 반응과 이슈 등을 분석 보고

- ④ 트루 스토리(True Story) by 솔트룩스

- A. 다양한 이슈나 트렌드 등에 관한 정보 제공

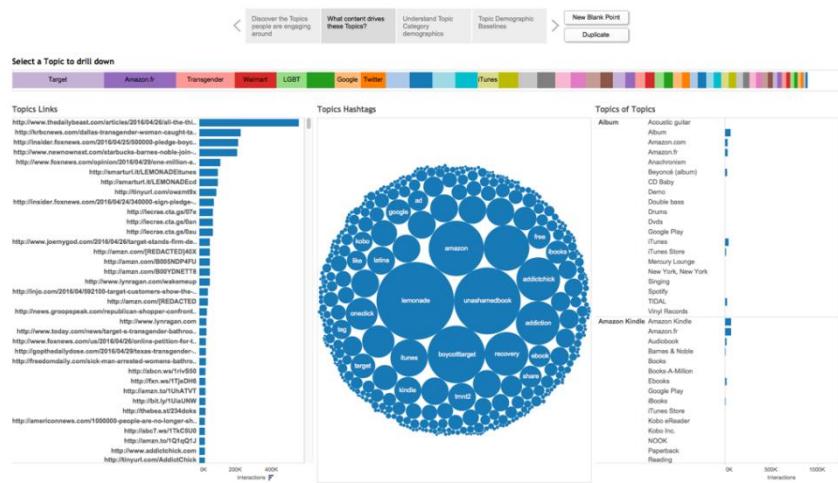
## 03 Journal - Topic Analysis(Modeling)

# Topic Analysis(Modeling)

# Topic Analysis

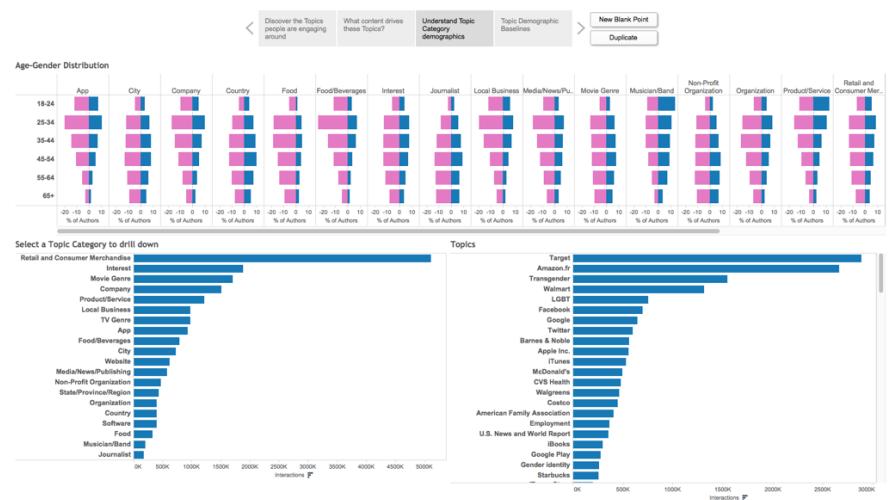
토픽 분석은 유사한 주제를 갖는 문서들을 묶어서 하나의 토픽으로 구성한다는 점에서는 전통적인 군집화와 유사한 특성을 갖지만, 하나의 문서가 다수의 토픽에 대응될 수 있다는 점에서 기존의 군집화와 차별성을 갖는다.

## Use Case - Topic Analysis



---

## Use Case - Topic Analysis



# 04 Topic modeling

## 깁스샘플링

- 일부 조건부 확률분포만 알고 있을 때 다차원 분포로부터 표본을 얻을 수 있는 방법

주사위를 두 개 던질 때 첫 번째 주사위의 눈을  $x$ , 두 주사위를 더한 값을  $y$ 일 때  $(x,y)$  쌍에 대한 표본을 구하고자 함. 우리는 조건부 확률을 알고 있음

$x$ 의 값을 알고 있다는 전제 하에  $y$ 는  
 $x+1, x+2, \dots x+6$  중 하나

$y$ 의 값을 알고 있다는 전제 하에  $x$ 는  
If  $y = 2 \rightarrow x = 1$   
If  $y = 3 \rightarrow x = 1 \text{ or } 2$   
If  $y = 11 \rightarrow x = 5 \text{ or } 6$

→깁스 샘플링은 임의의  $x$  or  $y$ 값에서 출발해서  $x$ 에 대한  $y$ 의 조건부 확률과  $y$ 에 대한  $x$ 의 조건부 확률 사이를 오가며 반복적으로 값을 선택하는 방법

# 04 Topic modeling

## 깁스샘플링

- 일부 조건부 확률분포만 알고 있을 때 다차원 분포로부터 표본을 얻을 수 있는 방법

주사위를 두 개 던질 때 첫 번째 주사위의 눈을  $x$ , 두 주사위를 더한 값을  $y$ 일 때  $(x,y)$  쌍에 대한 표본을 구하고자 함. 우리는 조건부 확률을 알고 있음

$x$ 의 값을 알고 있다는 전제 하에  $y$ 는  
 $x+1, x+2, \dots x+6$  중 하나

$y$ 의 값을 알고 있다는 전제 하에  $x$ 는  
 복잡함

$$\text{If } y = 2 \rightarrow x = 1$$

$$\text{If } y = 3 \rightarrow x = 1 \text{ or } 2$$

$$\text{If } y = 11 \rightarrow x = 5 \text{ or } 6$$

→ 이를 결합확률분포(joint distribution)이라고 하며, 원래 분포에서 샘플링했을 때와 유사한 결과값을 가져다 줌

# 04 Topic modeling

## 깁스샘플링

1. 각각의 단어에 임의의 주제를 배정한다
2. (잘못됐겠지만) 모든 단어에 주제가 배정되어있고, 그 결과 문헌별 주제 분포와 주제별 단어분포가 결정되었음.  
→ 이를깁스샘플링을 통해서 개선해 나가자.
3. n번째 단어를 골라 빼낸다. (나머지 단어들이 모두 적절하게 배정되어 있다고 가정하고)
4. 빼낸 단어를 제외하고, 그 단어가 속한 문헌의 주제분포( $P(\text{Topic} | \text{Document})$ )를 계산하고, 그 단어가 속한 주제의 단어분포( $P(\text{Word} | \text{Topic})$ )를 계산한다.  $P(\text{Word} | \text{Topic}) * P(\text{Topic} | \text{Document})$ 를 최대로하는 Topic을 찾아 그 단어에 배정한다.
5.  $n \leftarrow n+1$ 로 하고 3번으로 돌아간다.
6. 충분히 반복하다 보면 값들이 수렴하고 안정되게 된다.

# 04 Topic modeling

## 깁스샘플링

### 1. 각각의 단어의 임의의 주제를 배정한다

- A: Cute kitty
- B: Eat rice or cake
- C: Kitty and hamster
- D: Eat bread
- E: Rice, bread and cake
- F: Cute hamster eats bread and cake

W	cute	kit	eat	rice	cake	kit	ham	eat	bre	rice	bre	cake	cute	ham	eat	bre	cake
Z	#1	#2	#1	#2	#1	#2	#2	#1	#1	#1	#2	#1	#2	#2	#2	#1	#1

# 04 Topic modeling

## 깁스샘플링

2. 문헌별 주제 분포, 주제별 단어 분포를 계산한다

W	cute	kit	eat	rice	cake	kit	ham	eat	bre	rice	bre	cake	cute	ham	eat	bre	cake
Z	#1	#2	#1	#2	#1	#2	#2	#1	#1	#1	#2	#1	#2	#2	#2	#1	#1

$\theta$	A	B	C	D	E	F
#1	1.1	2.1	0.1	2.1	2.1	2.1
#2	1.1	1.1	2.1	0.1	1.1	3.1

$\varphi$	cute	kit	eat	rice	cake	ham	bre	SUM
#1	1.001	0.001	2.001	1.001	3.001	0.001	2.001	9.007
#2	1.001	2.001	1.001	1.001	0.001	2.001	1.001	8.007

# 04 Topic modeling

## 깁스샘플링

3. 첫번째 단어를 골라 빼낸다.

W	cute	kit	eat	rice	cake	kit	ham	eat	bre	rice	bre	cake	cute	ham	eat	bre	cake
Z	?	#2	#1	#2	#1	#2	#2	#1	#1	#1	#2	#1	#2	#2	#2	#1	#1

4. 새로운 주제를 배정한다

문헌 A에 속하는 단어 cute가 #1에 속할지 #2에 속할지 판단해야함  
문헌내 주제 분포를 따르면

(문헌 A 내에 #1이 있을 확률) =  $0.1 / (0.1+1.1) = 0.083\dots$

주제별 단어분포를 따르면

(#1 내의 단어가 cute일 확률) =  $0.001 / 8.007 = 0.00012\dots$

따라서 (A 안의 cute가 #1일 확률) =  $0.083 * 0.00012 \dots = 0.00008\dots$

→ 마지막 단어까지 업데이트가 끝나면 이것이 1회 갑스샘플링을 마무리한것이 됩니다.  
이 작업을 적절하게 여러번 반복하다보면 모든 단어가 적절하게 자기 주제를 찾아 배정되게 될것이라는 것이 갑스샘플링을 이용한 LDA의 가정

# O4 Topic modeling

## ■ 토픽 모델링

### 나이브 베이즈

$$P(S|w) = \frac{P(w|S)P(S)}{P(w)}$$

문서의 스팸 여부 파악.

- 문서가 스팸인지 아닌지 데이터셋에 이미 주어져 있었다.
- 각 단어의 스팸확률을 합하여 문서의 스팸여부를 결정한다.

### 토픽 모델링

$$P(T|w) = \frac{P(w|T)P(T)}{P(w)}$$

문서를 토픽별로 분류.

- 문서가 특정 토픽인지 아닌지는 주어지는 것이 아니다.
- 각 단어가 특정 토픽일 확률을 합하여 문서의 토픽을 정한다.

# 04 Topic modeling

## ■ 토픽 모델링

### 나이브 베이즈

$$P(S|w) = \frac{P(w|S)P(S)}{P(w)}$$

- $P(S)$ 는 이항 분포를 따른다.
- $P(S)$ 값이 0.5라고 가정해도 문제가 되지 않는다.

### 토픽 모델링

$$P(T|w) = \frac{P(w|T)P(T)}{P(w)}$$

- $P(T)$ 는 토픽의 개수  $K$ 개만큼의 확률변수를 가지는 다항 분포를 따른다.
- $P(T)$ 의 값을 정해버려서는 안된다.

# 04 Topic modeling

## ■ 토픽 모델링

나이브 베이즈(모델)

$$P(S|w) = \frac{P(w|S)P(S)}{P(w)}$$

토픽 모델링

~~$$P(T|w) = \frac{P(w|T)P(T)}{P(w)}$$~~



LDA 모델 이용

- $P(S)$ 는 이항 분포를 따른다.
- $P(S)$ 값이 0.5라고 가정해도 문제가 되지 않는다.

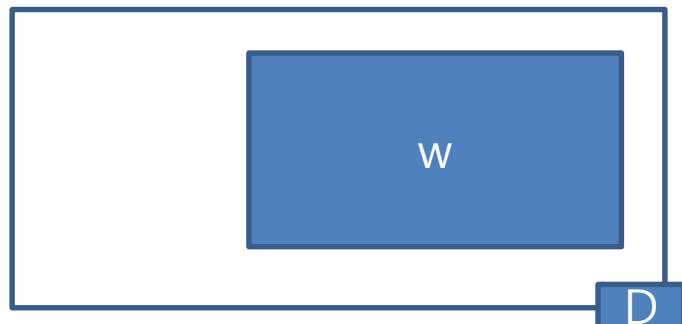
- $P(T)$ 는 토픽의 개수  $K$ 개만큼의 확률변수를 가지는 다항 분포를 따른다.
- $P(T)$ 의 값을 정해버려서는 안된다.

# 04 Topic Modeling

## LDA: 잠재 디리클레 할당(가정)

문서의 생성 과정을 가정한다.

문서 내의 각 단어는, 문서의 토픽 분포로부터 먼저 임의의 토픽이 선택된 뒤, 토픽의 단어 분포로부터 생성되었다고 가정한다.



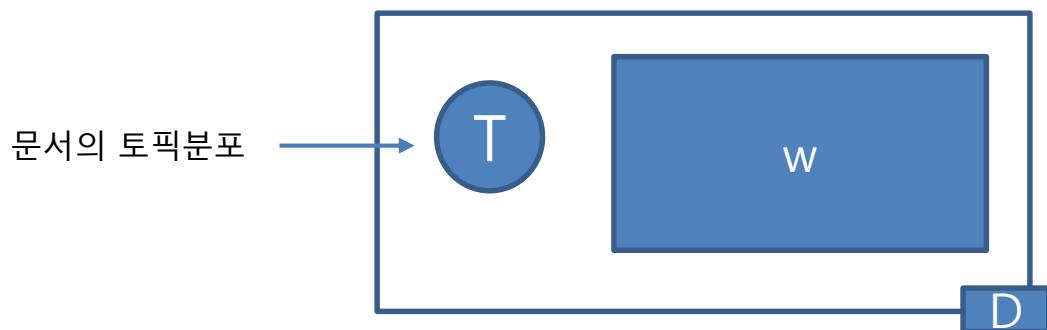
# 04 Topic Modeling

## LDA: 잠재 디리클레 할당(가정)

문서의 생성 과정을 가정한다.

문서 내의 각 단어는, 문서의 토픽 분포로부터 먼저 임의의 토픽이 선택된 뒤, 토픽의 단어 분포로부터 생성되었다고 가정한다.

1. 문서의 토픽분포가 존재한다고 가정한다.



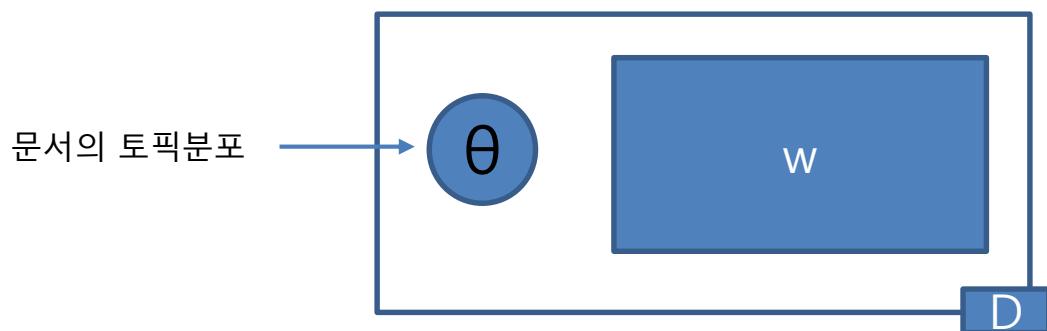
# 04 Topic Modeling

## LDA: 잠재 디리클레 할당(가정)

문서의 생성 과정을 가정한다.

문서 내의 각 단어는, 문서의 토픽 분포로부터 먼저 임의의 토픽이 선택된 뒤, 토픽의 단어 분포로부터 생성되었다고 가정한다.

2. 문서의 토픽분포로부터 임의의 토픽( $\theta$ )를 선택한다.



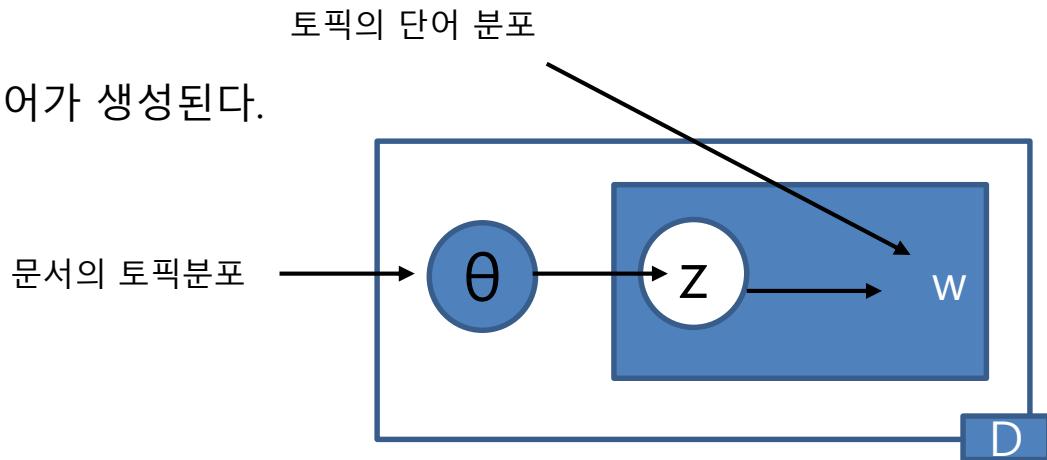
# 04 Topic Modeling

## LDA: 잠재 디리클레 할당(가정)

문서의 생성 과정을 가정한다.

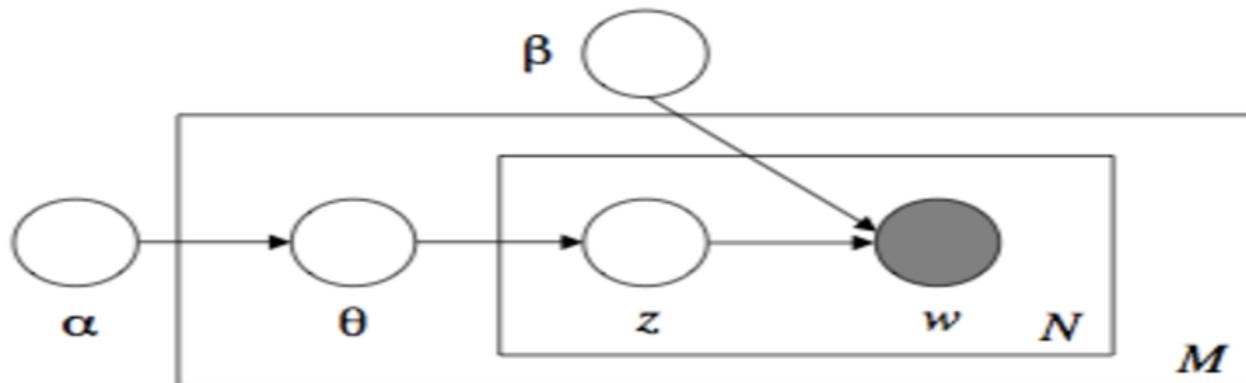
문서 내의 각 단어는, 문서의 토픽 분포로부터 먼저 임의의 토픽이 선택된 뒤, 토픽의 단어 분포로부터 생성되었다고 가정한다.

3. 토픽의 단어분포로부터 단어가 생성된다.



# 04 Topic Modeling

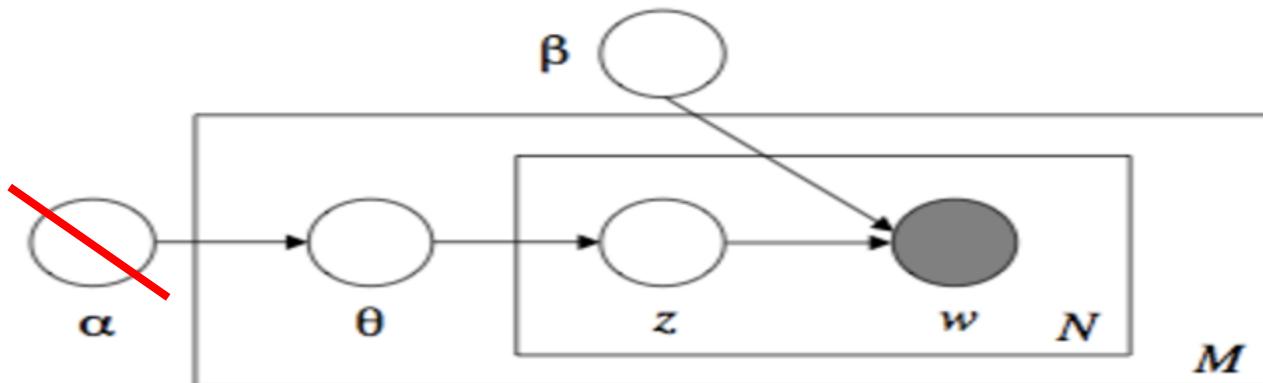
## LDA: 잠재 디리클레 할당



- $\alpha$ 는  $k$  차원 디리클레 분포의 매개변수이다.
- $\theta$ 는  $k$  차원 벡터이며,  $\theta^i$ 는 문서가  $i$ 번째 주제에 속할 확률 분포를 나타낸다.
- $z_n$ 는  $k$  차원 벡터이며,  $z_n^i$ 는 단어  $w_n$ 이  $i$ 번째 주제에 속할 확률 분포를 나타낸다.
- $\beta$ 는  $k \times V$  크기의 행렬 매개변수로,  $\beta_{ij}$ 는  $i$ 번째 주제가 단어집의  $j$ 번째 단어를 생성할 확률을 나타낸다.

# 04 Topic Modeling

## LDA: 잠재 디리클레 할당



- ~~$\alpha$ 는  $k$  차원 디리클레 분포의 매개변수이다.~~
  - $\theta$ 는  $k$  차원 벡터이며,  $\theta^i$ 는 문서가  $i$ 번째 주제에 속할 확률 분포를 나타낸다.
  - $z_n$ 는  $k$  차원 벡터이며,  $z_n^i$ 는 단어  $w_n$ 이  $i$ 번째 주제에 속할 확률 분포를 나타낸다.
  - $\beta$ 는  $N$  차원 벡터이며,  $\beta^i$ 는 단어  $w$ 가  $i$ 번째 주제에 속할 확률 분포를 나타낸다.
- $\alpha$ 는 책에서도 다루지 않고 있으며, 나타내려면 복잡한 수식이 필요하다.

# 04 Topic modeling

## LDA:(기브스 표집)

- 토픽을 가정했다고 없던 토픽 샘플이 생겨나지는 않는다.
- 문서당 토픽분포(세타)와 토픽당 단어분포(베타)를 먼저 구해줘야 하는데 토픽 샘플 없이는 이를 구할 수가 없다.

→ 어떻게 구할까?

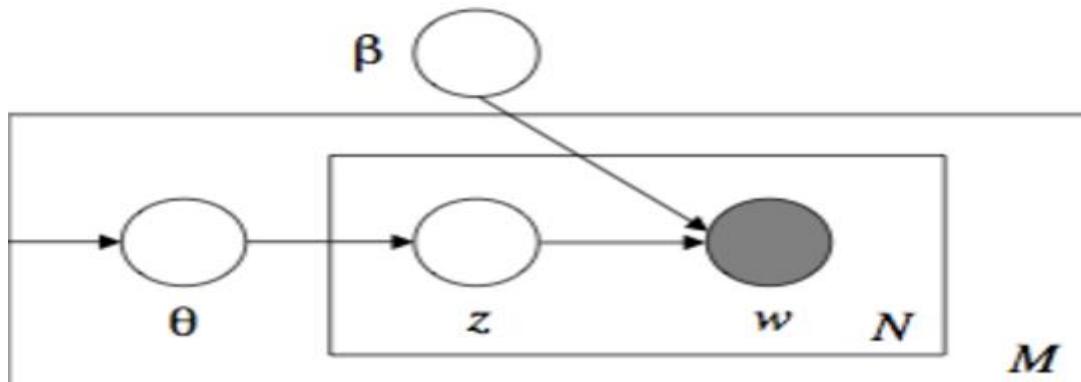
# 04 Topic modeling

## LDA:(기브스 표집)

- 토픽을 만드는 방법: 세타와 베타의 근사값을 구한다.(결합확률분포  $P(\Theta, \beta)$ 에서  $(\Theta, \beta)$ 를 표집 한다.)
  1. 볼록 기반 변분 알고리즘
  2. 기브스 표집

# 04 Topic modeling

## LDA:(기브스 표집)

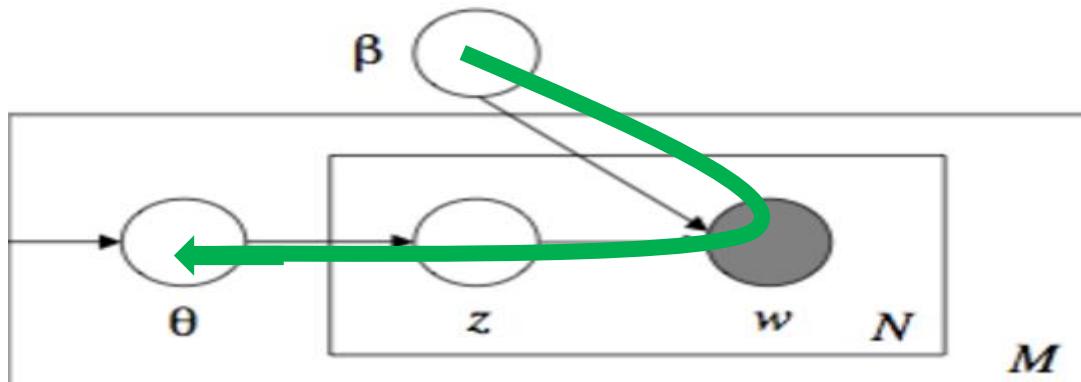


책에서는 하나가 주어졌을 때 다른 하나를 구해내는 방법을 반복하는(기브스 표집) 샘플링을 하였다.

기브스 표집은 일부 조건부 확률분포를 알고 있을 때 결합확률분포의 샘플을 얻어내는 방법이다.

# 04 Topic modeling

## LDA:(기브스 표집)



우리가 알고 있는 것: 문서당 단어 분포(일부 조건부 확률분포)

결합확률분포  $P(\Theta, \beta, z, w, d(\text{문서}))$ 에서  $(\Theta, \beta)$ 를 표집한다.

변수 간의 특징: 연쇄적, 가역적

조건부확률분포와 결합확률분포는 비례한다.

→ 반복( $\beta \rightarrow P(\Theta | \beta) \rightarrow \Theta \rightarrow P(\beta | \Theta)$ )

# 04 Topic Modeling

■ 잠재 디리클레 할당의 의미  
(여기서부터는 진짜 무시해도 됨)

L        D        A  
잠재 디리클레 할당  
:잠재 디리클레를 할당한다.

# 04 Topic Modeling

잠재 디리클레 할당의 의미

L            D            A  
잠재 디리클레 할당  
:잠재 디리클레를 할당한다.

책은 디리클레 분포 고려 안 했으니 그냥 LA아닌가?  
→ 맞기도 하고, 틀리기도 하다.

# 04 Topic Modeling

## ■ 잠재 디리클레 할당의 의미

실제 LDA와 책 LDA 차이

### 활용도가 다르다

토픽 모델링의 경우 비연속적인 자료이며 다항분포를 가정한다.

그러나 만약 주어지는 자료가 연속적이라면?(ex)소리)

잘 모르겠지만 연속확률분포인 디리클레 분포에 접근해야 할 필요가 있다.

# 04 Topic Modeling

## ■ 잠재 디리클레 할당의 확장

- 문서를 단어로 파악하는 방법 외에 앞서 나온 n-gram 모형을 사용하는 등의 변형이 가정하다.

앞선 설명은 결국 토픽모델링의 극히 일부분을 LDA로 제시한 것이며,  
LDA는 다양한 활용이 가능하다.(위키피디아 LDA 응용부분 참고하자)

# 04 Topic Modeling

## LDA

Topics	
gene	0.04
dna	0.02
genetic	0.01
...	
life	0.02
evolve	0.01
organism	0.01
...	
brain	0.04
neuron	0.02
nerve	0.01
...	
data	0.02
number	0.02
computer	0.01
...	

## Documents

### Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,\* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions

"are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Svante Paabo of the University of Berlin, who arrived at the SGN meeting. But coming up with a consensus answer may be more than just a genetic numbers game. "It's particularly more and more genomes" are completely sequenced and

sequenced. "It may be a way of organizing any newly sequenced genome," explains

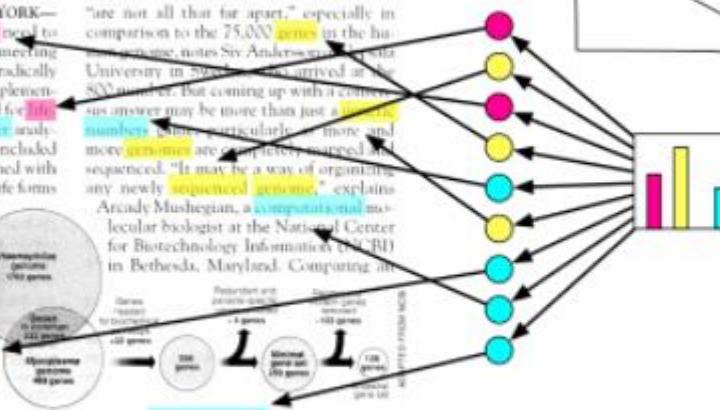
Ariandy Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing all

the genomes, he says, "you can see what genes are shared and what genes are unique."

\* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

## Topic proportions & assignments

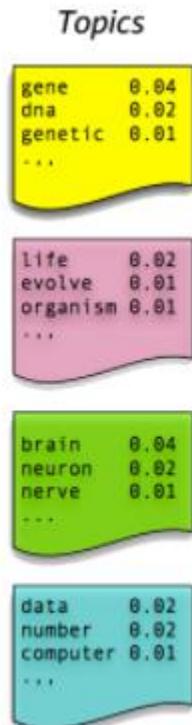


Stripping down, Computer analysis yields an estimate of the minimum modern and ancient genomes.

- 특정 토픽 (노란색) – 특정 단어 (gene) : 0.04

# 04 Topic Modeling

## LDA



*Documents*

### Seeking Life's Bare (Genetic) Necessities

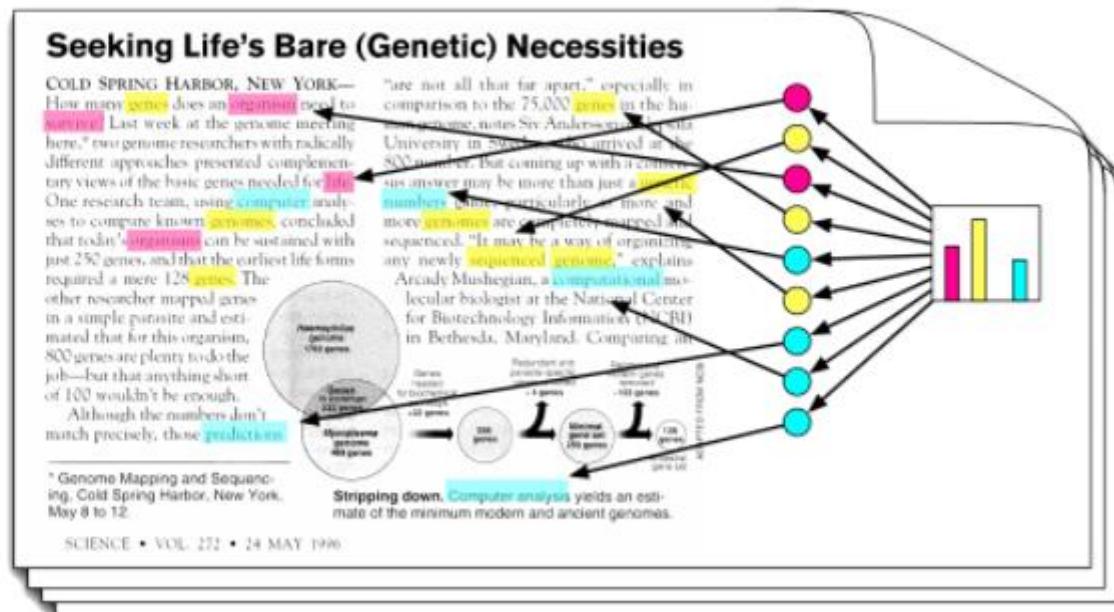
COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,\* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions

"are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Svante Paabo of the University of Berlin, who arrived at the SGN meeting. But coming up with a consensus answer may be more than just a genetic numbers game. As more and more genomes are completely sequenced and sequenced, "It may be a way of organizing any newly sequenced genome," explains Aronky Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing all

Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

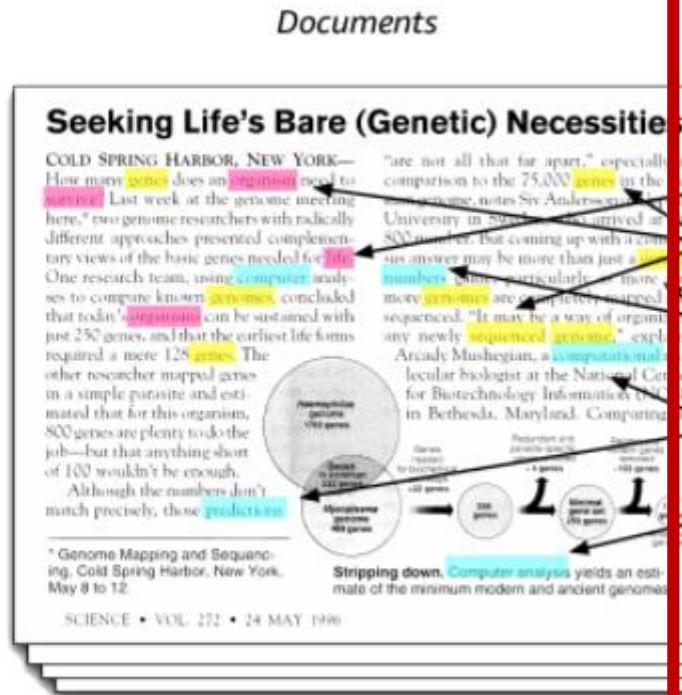
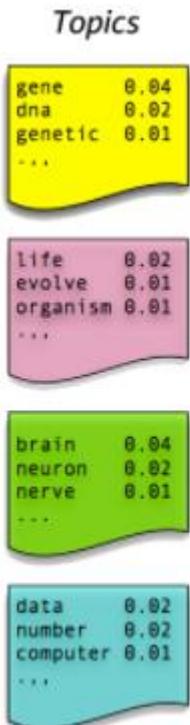
*Topic proportions & assignments*



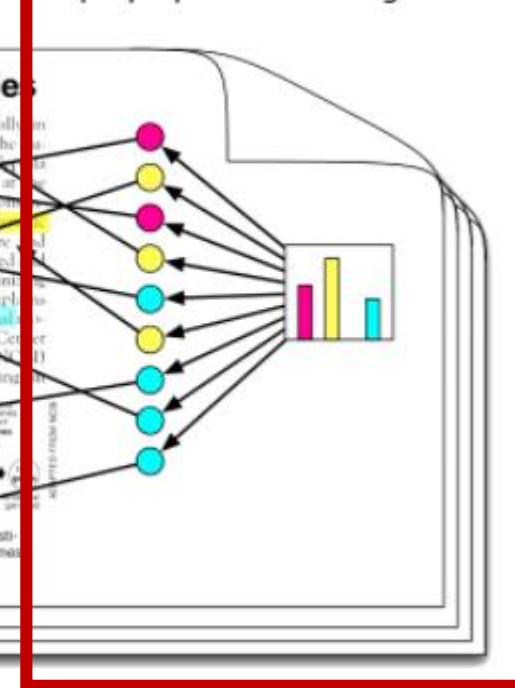
- 해당 문서 – 특정 토픽(노란색) : 最多 -> 유전 관련 문서

# 04 Topic Modeling

## LDA



*Topic proportions & assignments*



- 문서가 생성되는 과정을 확률 모형으로 모델링

# 05 Topic Modeling +

## LDA

- 모델에 대한 가정

- ① 토픽의 수는 K개로 고정
- ② 각 토픽을 단어에 대한 확률분포로 나타내주는 확률변수 존재  
토픽  $k$ 가 주어졌을 때 단어  $w$ 를 관찰할 수 있는 확률
- ③ 각 문서를 토픽에 대한 확률분포로 나타내주는 확률변수 존재  
문서  $d$ 에 어떤 토픽들이 섞여 있는지를 나타내는 분포
- ④ 문서 내의 각 단어는, 토픽 분포로부터 임의의 토픽이 선택된 뒤,  
토픽의 단어 분포로부터 생성되었다고 가정

# 05 Topic Modeling

## LDA

### 문서와 사용자들의 관심사

```
documents = [
    ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
    ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
    ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
    ["R", "Python", "statistics", "regression", "probability"],
    ["machine learning", "regression", "decision trees", "libsvm"],
    ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
    ["statistics", "probability", "mathematics", "theory"],
    ["machine learning", "scikit-learn", "Mahout", "neural networks"],
    ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
    ["Hadoop", "Java", "MapReduce", "Big Data"],
    ["statistics", "R", "statsmodels"],
    ["C++", "deep learning", "artificial intelligence", "probability"],
    ["pandas", "R", "Python"],
    ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
    ["libsvm", "regression", "support vector machines"]
]
```

# 05 Topic Modeling

## LDA

- Document\_topic을 생성하는 갑스 샘플링
  - ① 모든 문서의 모든 단어에 임의의 토픽을 부여
  - ② 토픽-단어 분포 & 문서-토픽 분포 ~> 각 토픽에 weight 할당
  - ③ Weight를 사용하여, 해당 단어에 알맞은 새로운 토픽 할당
  - ④ 이 과정의 반복
  - ⑤ 토픽-단어 분포 & 문서-토픽 분포의 결합확률로부터 나오는 표본 획득

# 05 Topic Modeling

## LDA

### 토픽의 총 개수

K = 4

### 조건부 확률분포 정의를 위한 준비

# 각 토픽이 각 문서에 할당되는 횟수 :

# Counter로 구성된 list

# 각각의 Counter는 각 문서를 의미함

```
document_topic_counts = [Counter() for _ in documents]
```

# 각 단어가 각 토픽에 할당되는 횟수 :

# Counter로 구성된 list

# 각각의 Counter는 각 토픽을 의미함

```
topic_word_counts = [Counter() for _ in range(K)]
```

# 각 토픽에 할당되는 총 단어 수 :

# 숫자로 구성된 list

# 각각의 숫자는 각 토픽을 의미함

```
topic_counts = [0 for _ in range(K)]
```

[0, 0, 0, 0]

[19, 12, 30, 6]

[Counter(), Counter(), Counter()]

[Counter({0: 4, 2: 2, 3: 1}), Counter({2: 3, 1: 2, 3: 0}), Counter({0: 3, 2: 3, 1: 0}), Counter({2: 5, 0: 0, 3: 0}), Counter({2: 3, 3: 1, 1: 0}), Counter({0: 4, 2: 2, 3: 0}), Counter({2: 3, 1: 1, 0: 0, 3: 0}), Counter({1: 3, 0: 1, 2: 0}), Counter({1: 2, 0: 2, 3: 0}), Counter({0: 3, 2: 1, 3: 0}), Counter({2: 2, 0: 1}), Counter({3: 4, 2: 0, 1: 0}), Counter({2: 2, 0: 1, 3: 0}), Counter({1: 4, 2: 1}), Counter({2: 3, 0: 0, 3: 0})]

4

document\_topic\_counts[2][0]

> 문서 3에서 토픽 1과 관련 있는 단어 수

[Counter(), Counter(), Counter()]

[Counter({'Java': 3, 'pandas': 2, 'Big Data': 2, 'statsmodels': 2, 'scikit-learn': 1, 'R': 1, 'Haskell': 1, 'artificial intelligence': 1, 'Hadoop': 1, 'Spark': 1, 'Storm': 1, 'programming languages': 1, 'machine learning': 1, 'MapReduce': 1, 'HBase': 0, 'regression': 0, 'C++': 0, 'statistics': 0, 'libsvm': 0}), Counter({'neural networks': 2, 'Postgres': 2, 'MongoDB': 2, 'Mahout': 1, 'deep learning': 1, 'databases': 1, 'MySQL': 1, 'probability': 1, 'scikit-learn': 1}])

topic\_word\_counts[0]['Java']

> 'Java'라는 단어와 토픽1이 연관 지어 나오는 단어 수

# 05 Topic Modeling

## LDA

### 토픽의 총 개수

K = 4

# 각 문서에 포함되는 총 단어 수 :

# 숫자로 구성된 *List*

# 각각의 순자는 각 문서를 의미함

```
document_lengths = [len(d) for d in documents]
```

# 단어 종류의 수 :

```
distinct_words = set(word for document in documents for word in document)
```

```
W = len(distinct_words)
```

# 총 문서의 수:

```
D = len(documents)
```

```
[7, 5, 6, 5, 4, 6, 4, 4, 4, 3, 4, 3, 5, 3]
['scipy', 'C++', 'R', 'Cassandra', 'artificial
intelligence', 'programming languages', 'deep
learning', 'neural networks', 'Python', 'support
vector machines', 'statistics', 'databases', 't
heory', 'numpy', 'Postgres', 'Java', 'MySQL',
'libsvm', 'Big Data', 'mathematics', 'regression
', 'decision trees', 'MongoDB', 'probability',
'scikit-learn', 'Mahout', 'Spark', 'HBase', 'pa
ndas', 'machine learning', 'MapReduce', 'Hadoop
', 'Haskell', 'statsmodels', 'NoSQL', 'Storm']
36 15
```

# 05 Topic Modeling

## LDA

```
### 각 단어를 임의의 토픽에 배정하고, 필요한 수치를 계산

random.seed(0)

document_topics = [[random.randrange(K) for word in document] for document in documents]

for d in range(D):
    for word, topic in zip(documents[d], document_topics[d]):
        document_topic_counts[d][topic] += 1
        topic_word_counts[topic][word] += 1
        topic_counts[topic] += 1
```

```
[[3, 3, 0, 2, 3, 3, 2], [3, 2, 1, 1, 2], [1, 0,
2, 1, 2, 0], [0, 2, 3, 0, 2], [3, 2, 1, 3], [3,
2, 0, 0, 0, 3], [0, 3, 2, 1], [2, 0, 1, 1], [1,
1, 3, 0], [0, 2, 3, 0], [2, 2, 0], [2, 1, 2, 3],
[0, 3, 2], [1, 2, 1, 1, 1], [0, 2, 3]]
Hadoop 3
Big Data 3
HBase 0
Java 2
Spark 3
```

# 05 Topic Modeling

## LDA

### 모든 문서가 0 이상의 확률을 가지도록 *smoothing*

```

def p_topic_given_document(topic, d, alpha=0.1):
    """문서 d의 모든 단어 중에서 topic에 속하는 단어의 비율 (smoothing을 더한 비율)"""
    return ((document_topic_counts[d][topic] + alpha) / (document_lengths[d] + K * alpha))

def p_word_given_topic(word, topic, beta=0.1):
    """topic에 속한 단어 중에서 word의 비율 (smoothing을 더한 비율)"""
    return ((topic_word_counts[topic][word] + beta) / (topic_counts[topic] + W * beta))

def topic_weight(d, word, k):
    """문서와 문서의 단어가 주어지면, k번째 토픽의 weight를 반환"""
    return p_word_given_topic(word, k) * p_topic_given_document(k, d)

def choose_new_topic(d, word):
    return sample_from([topic_weight(d, word, k) for k in range(K)])

```

```

print(p_topic_given_document(2,0))
print(p_word_given_topic('Java',2))
print(topic_weight(0,'Java',2))

0.28878378378378377
0.08898305084745763
0.02525194686211635

```

# 05 Topic Modeling

## LDA

```
### 랜덤으로 생성된 weight들을 이용해 인덱스를 생성

def sample_from(weights):
    """i는 weights[i] / sum(weights)의 확률로 반환"""
    total = sum(weights)
    rnd = total * random.random()  # 0과 total 사이를 균일하게 선택
    for i, w in enumerate(weights):
        rnd -= w  # 밑의 주석의 식을 만족하는 가장 작은 i를 반환
        if rnd <= 0: return i  # weights[0] + ... + weights[i] >= rnd
```

# 05 Topic Modeling

## LDA

```

### 조건부 확률분포를 이용해서 갑스 샘플링 실행

for iter in range(1000):
    for d in range(D):
        for i, (word, topic) in enumerate(zip(documents[d],
                                              document_topics[d])):
            # weight에 영향이 없도록
            # word와 topic을 제외하고 세어봄
            document_topic_counts[d][topic] -= 1
            topic_word_counts[topic][word] -= 1
            topic_counts[topic] -= 1
            document_lengths[d] -= 1

            # weight를 기준으로 새로운 topic을 선택
            new_topic = choose_new_topic(d, word)
            document_topics[d][i] = new_topic

            # 다시 세어 봄
            document_topic_counts[d][new_topic] += 1
            topic_word_counts[new_topic][word] += 1
            topic_counts[new_topic] += 1
            document_lengths[d] += 1

```

```

0 ('Hadoop', 3)
1 ('Big Data', 3)
2 ('HBase', 0)
3 ('Java', 2)
4 ('Spark', 3)
5 ('Storm', 3)
6 ('Cassandra', 2)
0 ('NoSQL', 3)
1 ('MongoDB', 2)
2 ('Cassandra', 1)
3 ('HBase', 1)
4 ('Postgres', 2)
0 ('Hadoop', 3)
0
1 ('Big Data', 3)
3
2 ('HBase', 0)
2
3 ('Java', 2)
0
4 ('Spark', 3)
0
5 ('Storm', 3)
0
6 ('Cassandra', 2)

```

# 05 Topic Modeling

## LDA

```
[Counter(), Counter(), Counter(), Counter(), Counter(), Counter(), Counter(),
Counter(), Counter(), Counter(), Counter(), Counter(), Counter(), Counter(),
Counter(), Counter(), Counter(), Counter()]
[Counter(), Counter(), Counter(), Counter()]
[0, 0, 0, 0]
36 15
[Counter({0: 4, 2: 2, 3: 1}), Counter({2: 3, 1: 2, 3: 0}), Counter({0: 3, 2: 3, 1: 0}),
}, Counter({2: 5, 0: 0, 3: 0}), Counter({2: 3, 3: 1, 1: 0}), Counter({0: 4, 2: 2, 3:
0}), Counter({2: 3, 1: 1, 0: 0, 3: 0}), Counter({1: 3, 0: 1, 2: 0}), Counter({1: 2,
0: 2, 3: 0}), Counter({0: 3, 2: 1, 3: 0}), Counter({2: 2, 0: 1}), Counter({3: 4, 2:
0, 1: 0}), Counter({2: 2, 0: 1, 3: 0}), Counter({1: 4, 2: 1}), Counter({2: 3, 0: 0,
3: 0})]
[Counter({'Java': 3, 'pandas': 2, 'Big Data': 2, 'statsmodels': 2, 'scikit-learn': 1,
'R': 1, 'Haskell': 1, 'artificial intelligence': 1, 'Hadoop': 1, 'Spark': 1, 'Storm':
1, 'programming languages': 1, 'machine learning': 1, 'MapReduce': 1, 'HBase': 0,
'regression': 0, 'C++': 0, 'statistics': 0, 'libsvm': 0}), Counter({'neural networks': 2,
'Postgres': 2, 'MongoDB': 2, 'Mahout': 1, 'deep learning': 1, 'databases': 1,
'MYSQL': 1, 'probability': 1, 'scikit-learn': 1, 'Cassandra': 0, 'HBase': 0, 'Python':
0, 'numpy': 0, 'decision trees': 0, 'theory': 0}), Counter({'Python': 4, 'regression': 3,
'R': 3, 'statistics': 3, 'HBase': 3, 'Cassandra': 2, 'libsvm': 2, 'scipy': 1,
'probability': 1, 'mathematics': 1, 'machine learning': 1, 'C++': 1, 'NoSQL': 1,
'numpy': 1, 'theory': 1, 'Hadoop': 1, 'support vector machines': 1, 'Java': 0, 'MongoDB':
0, 'Postgres': 0, 'statsmodels': 0, 'artificial intelligence': 0}), Counter(
{'Big Data': 1, 'probability': 1, 'decision trees': 1, 'C++': 1, 'deep learning': 1,
'artificial intelligence': 1, 'Hadoop': 0, 'Spark': 0, 'Storm': 0, 'NoSQL': 0, 'statistics':
0, 'machine learning': 0, 'libsvm': 0, 'Python': 0, 'programming languages': 0,
'MapReduce': 0, 'R': 0, 'support vector machines': 0})]
[19, 12, 30, 6]
```

# 05 Topic Modeling

## LDA

```
### 각 토픽에 가장 영향력이 높은 (weight값이 큰) 단어들 탐색
```

```
for k, word_counts in enumerate(topic_word_counts):
    for word, count in word_counts.most_common():
        if count > 0: print(k, word, count)
```

# 05 Topic Modeling

## LDA

```
### 각 토픽에 가장 영향력이 높은 (weight값이 큰) 단어들 탐색
```

```
for k, word_counts in enumerate(topic_word_counts):
    for word, count in word_counts.most_common():
        if count > 0: print(k, word, count)
0 Java 3
0 Big Data 3
0 Hadoop 2
0 HBase 1
0 C++ 1
0 Spark 1
0 Storm 1
0 programming languages 1
0 MapReduce 1
0 Cassandra 1
0 deep learning 1
1 HBase 2
1 neural networks 2
1 Postgres 2
1 MongoDB 2
1 machine learning 2
1 Cassandra 1
1 numpy 1
1 decision trees 1
1 deep learning 1
1 databases 1
1 MySQL 1
1 NoSQL 1
1 artificial intelligence 1
1 scipy 1
2 regression 3
2 Python 2
2 R 2
2 libsvm 2
2 scikit-learn 2
2 mathematics 1
2 support vector machines 1
2 Haskell 1
2 Mahout 1
3 statistics 3
3 probability 3
3 Python 2
3 R 2
3 pandas 2
3 statsmodels 2
3 C++ 1
3 artificial intelligence 1
3 theory 1
```

# 05 Topic Modeling

## LDA

Topic 0 : Big Data and Programming Languages

Topic 1 : databases

Topic 2 : Machine learning

Topic 3 : Python and statistics



Topic 0	Topic 1	Topic 2	Topic 3
Java (3)	HBase (2)	Regression (3)	Statistics (3)
Big Data (3)	Nnet(2)	Python (2)	Probability (3)
Hadoop (1)	Postgres (2)	R (2)	Python (2)
C++ (1)	MongoDB (2)	Libsvm (2)	R (2)
Spark (1)	ML (2)	ScikitLearn (2)	Pandas (2)

# 05 Topic Modeling

## LDA

```
topic_names = ["Big Data and programming languages",
               "Python and statistics",
               "databases",
               "machine learning"]

for document, topic_counts in zip(documents, document_topic_counts):
    print (document)
    for topic, count in topic_counts.most_common():
        if count > 0:
            print (topic_names[topic], count)
print()
```

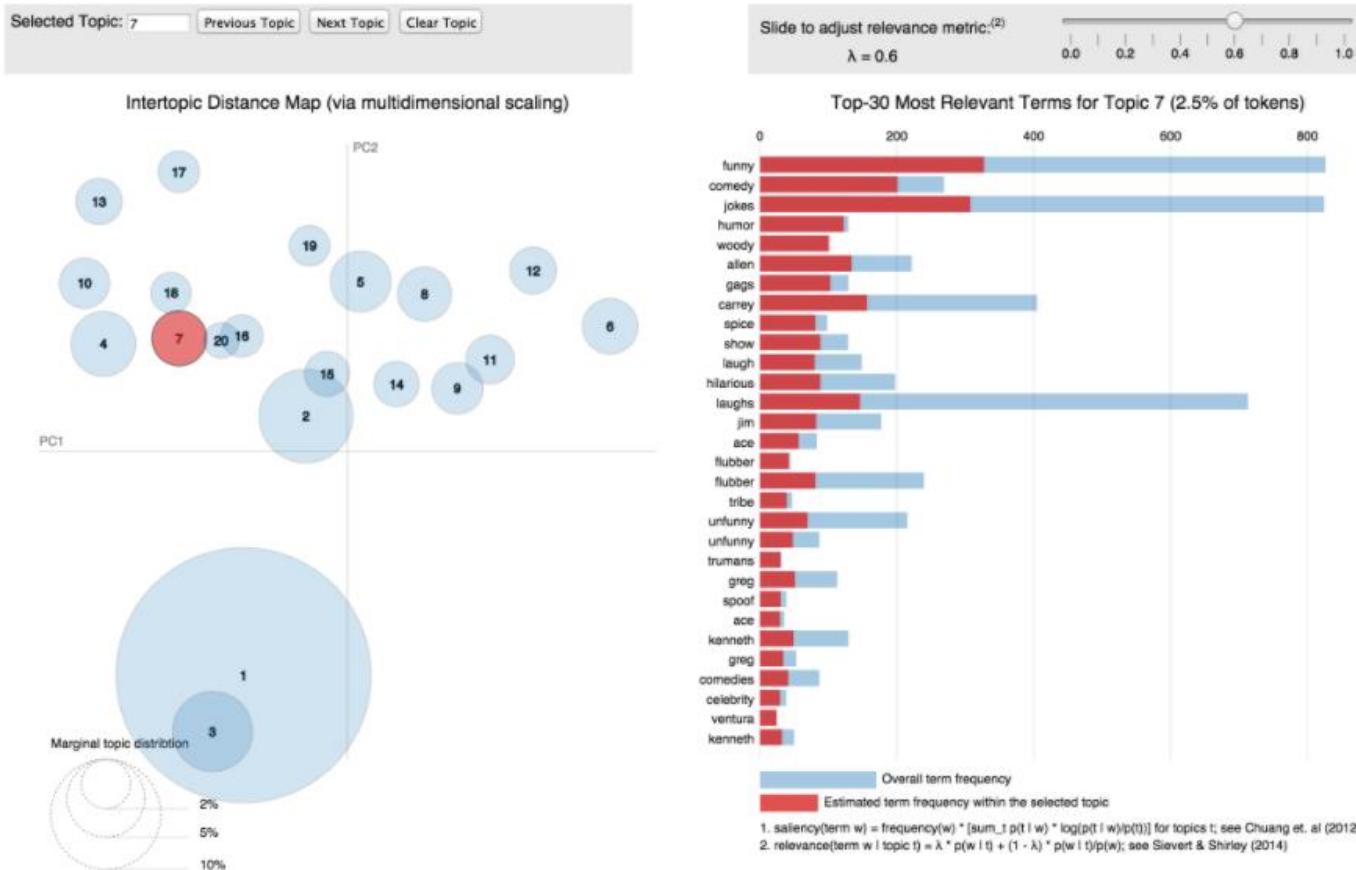
# 05 Topic Modeling

## LDA

```
[['Hadoop', 'Big Data', 'HBase', 'Java', 'Spark', 'Storm', 'Cassandra']  
Big Data and programming languages 7  
  
[['NoSQL', 'MongoDB', 'Cassandra', 'HBase', 'Postgres']  
Python and statistics 5  
  
[['Python', 'scikit-learn', 'scipy', 'numpy', 'statsmodels', 'pandas']  
Python and statistics 2  
databases 2  
machine learning 2  
  
[['R', 'Python', 'statistics', 'regression', 'probability']  
machine learning 3  
databases 2  
  
[['machine learning', 'regression', 'decision trees', 'libsvm']  
databases 2  
Python and statistics 2  
  
[['Python', 'R', 'Java', 'C++', 'Haskell', 'programming languages']  
databases 3  
Big Data and programming languages 3  
  
[['statistics', 'probability', 'mathematics', 'theory']  
machine learning 3  
databases 1  
  
[['machine learning', 'scikit-learn', 'Mahout', 'neural networks']  
databases 2  
Python and statistics 2  
  
[['neural networks', 'deep learning', 'Big Data', 'artificial intelligence']  
Python and statistics 3  
Big Data and programming languages 1  
|  
[['Hadoop', 'Java', 'MapReduce', 'Big Data']]  
Big Data and programming languages 4
```

# 05 Topic Modeling

## LDAvis



<https://github.com/cpsievert/LDAvis>

# 06 References

- KoNLPy, <http://konlpy.org/ko/latest/>
- KoNLPy\_GitHub, <https://github.com/konlpy/konlpy>
- NLTK, <https://datascienceschool.net/view-notebook/118731eec74b4ad3bdd2f89bab077e1b/>
- Text Mining, <https://www.lucypark.kr/courses/2015-dm/text-mining.html>
- Topic Modeling, <https://www.youtube.com/watch?v=BuMu-bdoVrU>
- <https://datascienceschool.net/view-notebook/118731eec74b4ad3bdd2f89bab077e1b/>
- <https://datascienceschool.net/view-notebook/70ce46db4ced4a999c6ec349df0f4eb0/>

# THANK YOU !