# 텍스트 마이닝

## 1. 텍스트 마이닝 개요

텍스트 마이닝은 단어의 출현 빈도, 단어 간 관계성 등을 파악하여 유의미한 정보를 추출하는 것이다. 이는 '자연어 처리 기술'을 기반으로 하고 있다.

텍스트마이닝(Text mining)은 데이터마이닝의 일부라고 볼 수 있다. 데이터마이닝이 수치데이터와 범주형데이터를 집중적으로 보는 반면에 텍스트마이닝은 텍스트데이터를 중점적으로 다룬다.

텍스트데이터를 다루는 것은 수치데이터를 다루는 것과 프로세싱이나 처리방법에서 많이 다르다.  텍스트마이닝을 잘 하기 위해서는 형태소분석기나 구문분석기와 같은 자연어처리 도구를 잘 사용할 수 있어야 하며 그 외에 다루는 언어에 대해서도 잘 알고 있어야 한다.

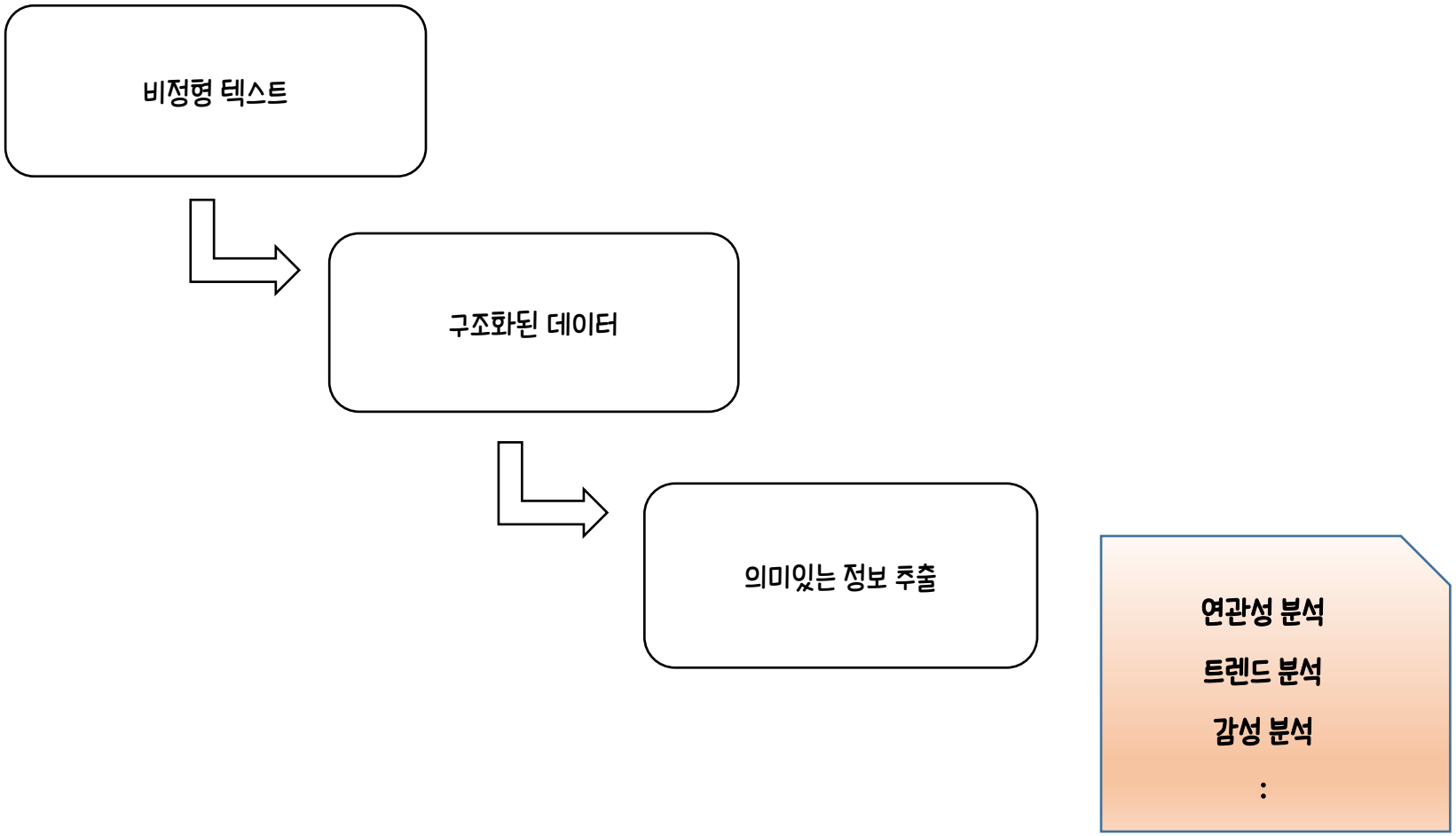
텍스트 마이닝은 많은 양의 비정형 데이터에 접근하고 활용하는데 사용되고 있고, 가치나 인사이트를 찾을 수 있을 뿐 아니라 비정형 데이터를 관리함으로써 실질적인 ROI(Return On Investment)를 이끌어 낼 수도 있다.

소셜 미디어는 시장 및 고객 정보를 파악하는데 있어서 점차 그 중요성이 높아지고 있으며 텍스트 마이닝은 많은 양의 비정형 데이터를 분석함으로써 해당 브랜드나 제품에 대한 다양한 의견과 감성반응을 살펴볼 수 있다.

텍스트 마이닝을 통해 문서들을 자동으로 분류할 수 있고, 문서나 단어들간의 연관성을 분석할 수 있으며 텍스트에 담겨있는 감성(즉, 평가, 성향)을 예측할 수 있고 시간의 흐름에 따른 이슈들의 변환과정을 추적할 수 있다.

# 텍스트 마이닝

## 1. 텍스트 마이닝 개요



## 2. 자연어 처리

'자연어'가 무엇일까? 우리가 일상생활에서 사용하는 말, 언어이다.

자연어 처리 기술을 바탕으로 사람들이 작성한 텍스트를 컴퓨터가 분석하여 중요한 단어나 문장들을 추출할 수 있다. 이를 잘 활용하면 매력적인 뉴스 기사의 헤드라인 작성, 제품에 대한 고객 반응 분석, 우리 브랜드에 대한 고객 생각 등을 알아낼 수 있다.

한국어는 어순이나 조사 등을 영어처럼 명확하게 끊어지지 않는 부분이 있다.
'한국어'라는 단어를 분석해 보면 '한국어는', '한국어의', '한국어를' 등의 표현들은 모두 '한국어'라는 핵심 단어를 가지고 있다. 하지만 기계적으로 이 표현들을 각각 다르게 인식할 수도 있으므로 한국어로 된 텍스트를 쪼개는 과정이 영어보다 더 많이 필요하다. 현재 나와 있는 기술만으로도 텍스트 마이닝을 통하여 충분히 유의미한 인사이트를 도출해낼 수 있다.

저연어 처리는 인간이 사용하는 언어를 컴퓨터가 사용할 수 있게 처리하는 것을 말한다. 애플의 시리(Siri)와 같은 음성인식이나 광학문자판독을 통해서 책에서 글자를 읽어 들이거나 웹 페이지에서 사람이 작성한 글을 로봇을 이용해서 크롤링 하고 해석한 후 어떤 것이 핵심어이고 어떤 것이 주제어인가 등을 알아내기도 하며 글쓴이의 감정이나 상태 등을 알아내기도 한다.

## 2. 자연어 처리

자언어 처리는 형태소분석기, 구문분석기와 같은 사람이 작성한 글이나 대화를 컴퓨터를 통해 해석할 수 있게 하는 소프트웨어를 개발하거나 연구하고 그런 것들을 이용해서 실제로 작업하는 것을 말한다.

- 형태소분석기

형태소를 구분하고 무엇인지 알려주는 것이다.

- 구문분석기

주어, 목적어, 서술어와 같은 형태로 품사보다는 단위가 더 높은 논리적 레벨까지를 처리해주는 것이다.

- 잘 알려진 형태소분석기 종류

Hannanum: KAIST의 한나눔 형태소 분석기와 NLP_HUB 구문분석기

KKMA: 서울대의 꼬꼬마 형태소/구문 분석기 v2..1

KOMORAN: Junsoo Shin님의 코모란 v3.3.3

Twitter: OpenKoreanText의 오픈 소스 한국어 처리기 v2..2..0 (구 Twitter 한국어 분석기)1-1

Eunjeon: 은전한닢 프로젝트의 SEunjeon(S은전)

Arirang: 이수명님의 Arirang Morpheme Analyzer 1-2.

RHINO: 최석재님의 RHINO v2..5.4

# 텍스트 마이닝

## 2. 자연어 처리

형태소 분석으로 어절들의 품사를 파악한 후 명사,동사, 형용사 등 의미를 지닌 품사의 단어를 추출해 각 단어가 얼마나 많이 등장했는지 확인한다.

install.packages("KoNLP")

library(KoNLP)

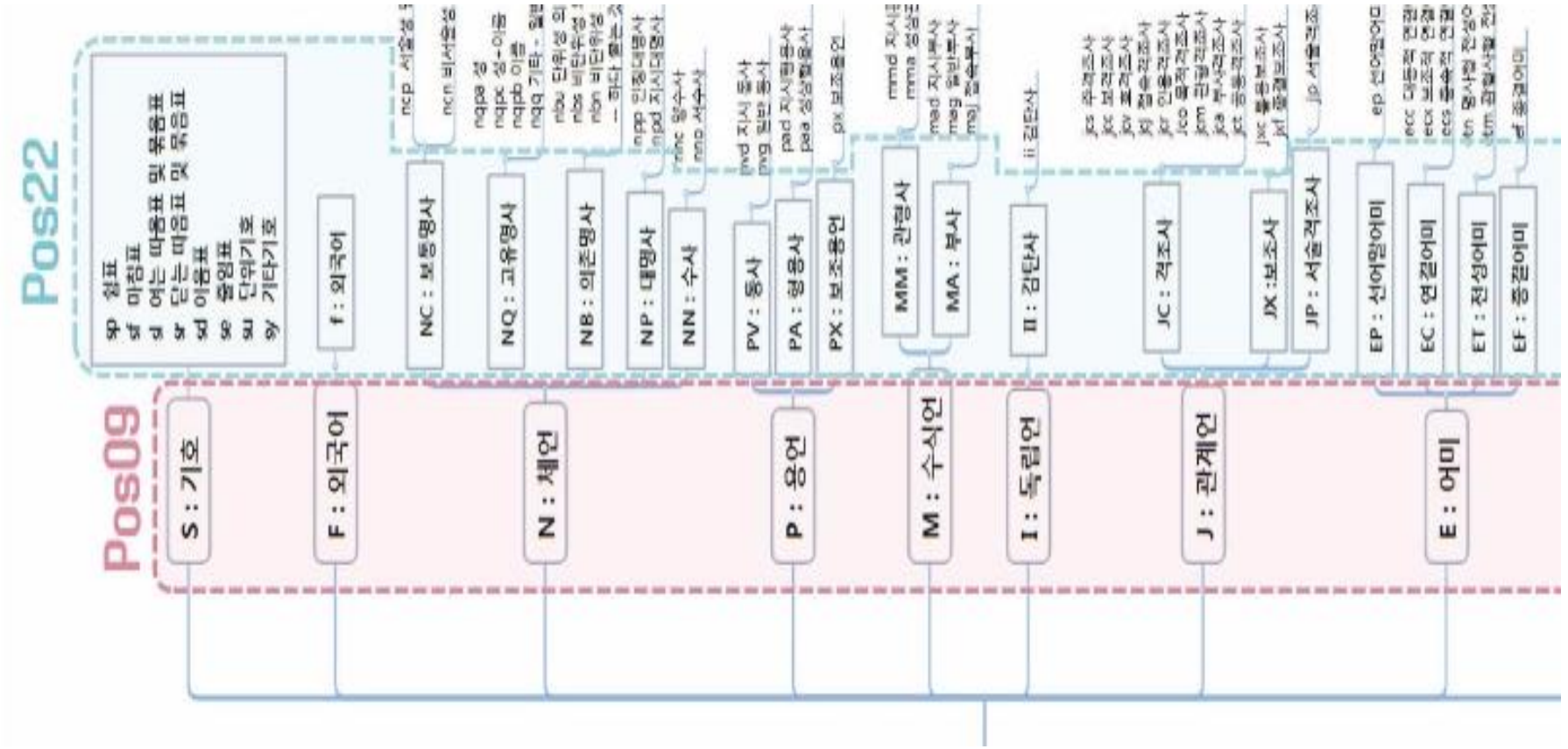
```
> extractNoun("대한민국의 영토는 한반도와 그 부속도서로 한다")
[1] "대한민국" "영토"     "한반도"   "부속도서" "한"
> SimplePos22("대한민국의 영토는 한반도와 그 부속도서로 한다")
$`대한민국의`
[1] "대한민국/NC+의/JC"

$영토는
[1] "영토/NC+는/JX"

$한반도와
[1] "한반도/NC+와/JC"

$그
[1] "그/MM"

$부속도서로
[1] "부속도서/NC+로/JC"

$한
[1] "한/NN"

$다
[1] "다/MA"
```

```
> SimplePos09("대한민국의 영토는 한반도와 그 부속도서로 한다")
$`대한민국의`
[1] "대한민국/N+의/J"

$영토는
[1] "영토/N+는/J"

$한반도와
[1] "한반도/N+와/J"

$그
[1] "그/M"

$부속도서로
[1] "부속도서/N+로/J"

$한
[1] "한/N"

$다
[1] "다/M"
```

S: 기호
F: 외국어
N: 체언(명사, 대명사, 수사)
P: 용언(동사, 형용사)
M: 수식언(관형사, 부사)
I: 독립언(감탄사)
J: 관계언(조사)
E: 어미
X: 접사

## 2. 자연어 처리

한나눔에서 사용하는 카이스트 품사 태그 셋



## 2. 자연어 처리

```
> extractNoun("대한민국의 영토는 한반도와 그 부속도서로 한다")
[1] "대한민국" "영토"     "한반도"   "부속도서" "한"
> sapply("대한민국의 영토는 한반도와 그 부속도서로 한다", extractNoun, USE.NAMES=F)
     [,1]
[1,] "대한민국"
[2,] "영토"
[3,] "한반도"
[4,] "부속도서"
[5,] "한"
```


```
setup_twitter_oauth(api_key,api_secret, access_token,access_token_secret)
key <- "미세먼지"
key <- enc2.utf8(key)
result <- searchTwitter(key, n=100)
DF <- twListToDF(result)
content <- DF$text
content <- gsub("[[:lower:][:upper:][:digit:][:punct:][:cntrl:]]", "", content)
content <- gsub("미세먼지", "", content)
content
word <- extractNoun(content)
cdata <- unlist(word)
cdata
cdata <- Filter(function(x) {nchar(x) < 6 & nchar(x) >= 2} ,cdata)
wordcount <- table(cdata)
wordcount <- head(sort(wordcount, decreasing=T),30)
library(wordcloud)
wordcloud(names(wordcount),freq=wordcount,scale=c(5,1),rot.per=0.35,min.freq=2.,
random.order=F,random.color=T,colors=rainbow(20))
```

## 3. 텍스트 전처리와 tm 패키지

**[ 정규 표현식 활용 ]**

```
word <- "JAVA javascript 가나다 12.3 %^&*"
gsub("A", "", word)
gsub("a", "", word)
gsub("[Aa]", "", word)
gsub("[가-힣]", "", word)
gsub("[^가-힣]", "", word)
gsub("[&^%*]", "", word)
gsub("[[:punct:]]", "", word)
gsub("[[:alnum:]]", "", word)
gsub("[12.34567890]", "", word)
gsub("[[:digit:]]", "", word)
gsub("[^[:alnum:]]", "", word)
gsub("[[:space:]]", "", word)
gsub("[[:space:][:punct:]]", "", word)
```

| | |
|---|---|
| . | Any character except \n |
| \| | Or, e.g. (a\|b) |
| [...] | List permitted characters, e.g. [abc] |
| [a-z] | Specify character ranges |
| [^...] | List excluded characters |
| (...) | Grouping, enables back referencing using \\N where N is an integer |

| | |
|---|---|
| * | Matches at least 0 times |
| + | Matches at least 1 time |
| ? | Matches at most 1 time; optional string |
| {n} | Matches exactly n times |
| {n,} | Matches at least n times |
| {,n} | Matches at most n times |
| {n,m} | Matches between n and m times |

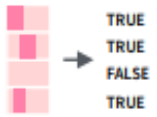
| | |
|---|---|
| [[:digit:]] or \\d | Digits; [0-9] |
| \\D | Non-digits; [^0-9] |
| [[:lower:]] | Lower-case letters; [a-z] |
| [[:upper:]] | Upper-case letters; [A-Z] |
| [[:alpha:]] | Alphabetic characters; [A-z] |
| [[:alnum:]] | Alphanumeric characters [A-z0-9] |
| \\w | Word characters; [A-z0-9_] |
| \\W | Non-word characters |
| [[:xdigit:]] or \\x | Hexadec. digits; [0-9A-Fa-f] |
| [[:blank:]] | Space and tab |
| [[:space:]] or \\s | Space, tab, vertical tab, newline, form feed, carriage return |
| \\S | Not space; [^[:space:]] |
| [[:punct:]] | Punctuation characters; !"#$%&'()*+,-./:;<=>?@[]^_`{\|}~ |
| [[:graph:]] | Graphical char.; [[:alnum:][:punct:]] |
| [[:print:]] | Printable characters; [[:alnum:][:punct:]\\s] |
| [[:cntrl:]] or \\c | Control characters; \n, \r etc. |

## 3. 텍스트 전처리와 tm 패키지

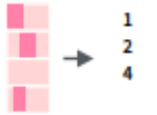
[ stringr 패키지 활용 ]

```
install.packages("stringr")
library(stringr)
```

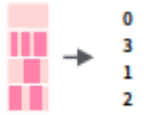
# Detect Matches



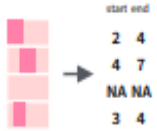

**str_detect**(string, **pattern**) Detect the presence of a pattern match in a string.
*str_detect(fruit, "a")*




**str_which**(string, **pattern**) Find the indexes of strings that contain a pattern match.
*str_which(fruit, "a")*




**str_count**(string, **pattern**) Count the number of matches in a string.
*str_count(fruit, "a")*




**str_locate**(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

# Subset Strings

**str_sub**(string, start = 1L, end = -1L) Extract substrings from a character vector.
*str_sub(fruit, 1, 3); str_sub(fruit, -2)*

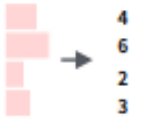
**str_subset**(string, **pattern**) Return only the strings that contain a pattern match.
*str_subset(fruit, "b")*


**str_extract**(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*


**str_match**(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**.
*str_match(sentences, "(a|the) ([^ ]+)")*

### 3. 텍스트 전처리와 tm 패키지

## Manage Lengths




**str_length**(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

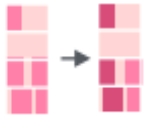
**str_pad**(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

**str_trunc**(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

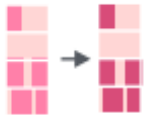
**str_trim**(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

## Mutate Strings

**str_sub**() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*



**str_replace**(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*



**str_replace_all**(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*


**str_to_lower**(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*


**str_to_upper**(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*


**str_to_title**(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

### 3. 텍스트 전처리와 tm 패키지

## Join and Split

**str_c**(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

**str_c**(..., sep = "", **collapse = NULL**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

**str_dup**(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

**str_split_fixed**(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*


**str_glue**(…, .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

**str_glue_data**(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

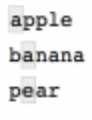
## Order Strings


**str_order**(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

**str_sort**(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

## Helpers

**str_conv**(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*




**str_view**(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*


**str_view_all**(string, **pattern**, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

**str_wrap**(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

## 3. 텍스트 전처리와 tm 패키지

### [ tm 패키지를 이용한 텍스트 전처리 : tm_map() 함수 사용 ]

텍스트는 기본적으로 비정형 데이터로서 분석에 불필요한 때로는 분석에 방해가 되는 요소들이 포함되어 있어서 데이터 정제(data cleaning) 작업은 필수이다. tm 패키지에는 텍스트 데이터의 정제작업을 지원하는 다양한 변환함수를 제공한다. getTransformations()이라는 함수를 수행시키면 사용 가능한 변환함수의 리스트를 확인할 수 있으며 이 함수들은 tm_map() 함수에 인수로 전달하여 변환작업을 처리할 수 있다. 문서에서 문장 부호를 제거하거나, 문자를 모두 소문자로 바꾸거나, 단어의 어간을 추출해주는 스테밍(stemming)을 적용할 수 있다.

```
tm_map(
  x,  # 코퍼스
  FUN  # 변환에 사용할 함수
)
```

```
> getTransformations()
[1] "removeNumbers"     "removePunctuation" "removeWords"
[4] "stemDocument"      "stripWhitespace"
```

```
corp2. <- tm_map(corp1,stripWhitespace) # 여러 개의 공백을 하나의 공백으로 변환한다.
    corp2. <- tm_map(corp2.,removeNumbers) # 숫자를 제거한다.
corp2. <- tm_map(myCorpus, content_transformer(tolower)) # 영문 대문자를 소문자로 변환한다.
corp2. <- tm_map(corp2.,removePunctuation) # 마침표,콤마,세미콜론,콜론 등 문자 제거한다.
corp2. <- tm_map(corp2.,PlainTextDocument)
stopword2. <- c(stopwords('en'),"and","but") # 기본 불용어 외에 불용어로 쓸 단어 추가
corp2. <- tm_map(corp2.,removeWords,stopword2.) # 불용어 제거하기 (전치사 , 관사 등)
```

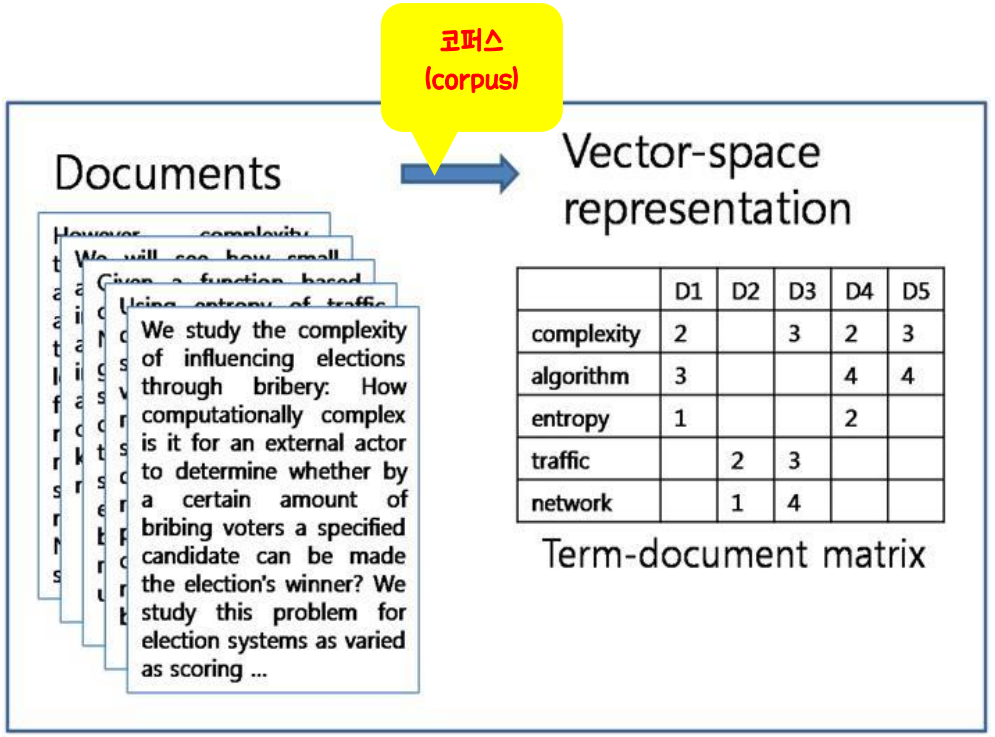
## 3. 텍스트 전처리와 tm 패키지

### 코퍼스란

코퍼스(corpus; 말뭉치): 언어학에서 구조를 이루고 있는 텍스트 집합으로 통계 분석 및 가설 검증, 언어 규칙의 검사 등에 사용된다. 텍스트 마이닝 패키지인 tm에서 문서를 관리하는 기본구조를 Corpus라 부르며, 이는 텍스트 문서들의 집합을 의미한다.

텍스트 마이닝을 위해 수행해야 할 첫 번째 작업은 비 구조화된 텍스트 즉, **비정형의 텍스트를 구조화된 데이터로 변환하는 것**이다. 텍스트를 구조화하는 방법 가운데 하나로 코퍼스(단어주머니:bag-of-words)접근법이 일반적으로 많이 사용된다.

코퍼스 접근법은 **분석 대상이 되는 개별 텍스트 즉 문서(document)를 단어의 집합(주머니)으로 단순화시킨 표현 방법으로서 단어의 순서나 문법은 무시하고 단어의 출현 빈도만을 이용하여 텍스트를 매트릭스로 표현**한다. 이때 생성되는 매트릭스를 term-document-matrix(TDM)또는 document-term-matrix(DTM) 라고 한다.




| | D1 | D2 | D3 | D4 | D5 |
|---|---|---|---|---|---|
| complexity | 2 | | 3 | 2 | 3 |
| algorithm | 3 | | | 4 | 4 |
| entropy | 1 | | | 2 | |
| traffic | | 2 | 3 | | |
| network | | 1 | 4 | | |

## 3. 텍스트 전처리와 tm 패키지

분석해야 할 텍스트를 문서들의 집합인 코퍼스 형식으로 변환해야 한다. tm 패키지의 Corpus() 함수를 사용한다. 이 함수는 다양한 소스로부터 읽어 들인 텍스트를 텍스트 마이닝을 위한 Corpus 객체로 변환한다. getSources() 함수를 사용하면 사용 가능한 소스객체의 종류를 파악할 수 있다.

```
> getSources()
[1] "DataframeSource" "DirSource"       "URISource"       "VectorSource"    "XMLSource"
[6] "ZipSource"
```

[ tm 패키지를 이용한 텍스트 마이닝 예제 1 ]

```
lunch <- c("커피 파스타 치킨 샐러드 아이스크림",
           "커피 우동 소고기김밥 귤",
           "참치김밥 커피 오뎅",
           "샐러드 피자 파스타 콜라",
           "티라무슈 햄버거 콜라",
           "파스타 샐러드 커피"
          )
```

## 3. 텍스트 전처리와 tm 패키지

```
cps <- VCorpus(VectorSource(lunch))
tdm <- TermDocumentMatrix(cps)
tdm
(m <- as.matrix(tdm))
```

```
> (m <- as.matrix(tdm))
          Docs
Terms      1 2 3 4 5 6
  샐러드   1 0 0 1 0 1
  소고기김밥 0 1 0 0 0 0
  아이스크림 1 0 0 0 0 0
  참치김밥   0 0 1 0 0 0
  티라무슈   0 0 0 0 1 0
  파스타     1 0 0 1 0 1
  햄버거     0 0 0 0 1 0
```

```
cps <- VCorpus(VectorSource(lunch))
tdm <- TermDocumentMatrix(cps,
            control=list(wordLengths = c(1, Inf)))
tdm
(m <- as.matrix(tdm))
```

```
> (m <- as.matrix(tdm))
          Docs
Terms      1 2 3 4 5 6
  귤       0 1 0 0 0 0
  샐러드   1 0 0 1 0 1
  소고기김밥 0 1 0 0 0 0
  아이스크림 1 0 0 0 0 0
  오뎅     0 0 1 0 0 0
  우동     0 1 0 0 0 0
  참치김밥   0 0 1 0 0 0
  치킨     1 0 0 0 0 0
  커피     1 1 1 0 0 1
  콜라     0 0 0 1 1 0
  티라무슈   0 0 0 0 1 0
  파스타     1 0 0 1 0 1
  피자     0 0 0 1 0 0
  햄버거     0 0 0 0 1 0
```

# 텍스트 마이닝

## 3. 텍스트 전처리와 tm 패키지

rowSums(m)

colSums(m)

```
> rowSums(m)
      귤 샐러드 소고기김밥 아이스크림      오뎅      우동
        1         3         1         1         1         1
참치김밥      치킨      커피      콜라 티라무슈 파스타
        1         1         4         2         1         3
    피자 햄버거
        1         1
> colSums(m)
1 2 3 4 5 6
5 4 3 4 3 3
```

```
> m
                Docs
Terms            1 2 3 4 5 6
  귤             0 1 0 0 0 0
  샐러드         1 0 0 1 0 1
  소고기김밥     0 1 0 0 0 0
  아이스크림     1 0 0 0 0 0
  오뎅           0 0 1 0 0 0
  우동           0 1 0 0 0 0
  참치김밥       0 0 1 0 0 0
  치킨           1 0 0 0 0 0
  커피           1 1 1 0 0 1
  콜라           0 0 0 1 1 0
  티라무슈       0 0 0 0 1 0
  파스타         1 0 0 1 0 1
  피자           0 0 0 1 0 0
  햄버거         0 0 0 0 1 0
```

```
> t(m)
    Terms
Docs 귤 샐러드 소고기김밥 아이스크림 오뎅 우동 참치김밥 치킨 커피 콜라 티라무슈 파스타 피자 햄버거
   1  0      1         0          1    0    0        0    1    1    0         0      1    0      0
   2  1      0         1          0    0    1        0    0    1    0         0      0    0      0
   3  0      0         0          0    1    0        1    0    1    0         0      0    0      0
   4  0      1         0          0    0    0        0    0    0    1         0      1    1      0
   5  0      0         0          0    0    0        0    0    0    1         1      0    0      1
   6  0      1         0          0    0    0        0    0    1    0         0      1    0      0
```

com <- m %*% t(m) # 동시출현(Co-occurrence)이란 한 문장, 문단 또는 텍스트 단위에서 같이 출현한 단어를 가리킨다.

```
> com
          Terms
Terms      귤 샐러드 소고기김밥 아이스크림 오뎅 우동 참치김밥 치킨 커피 콜라 티라무슈 파스타 피자 햄버거
  귤         1      0          1          0    0    1        0    0    1    0         0      0    0      0
  샐러드     0      3          0          1    0    0        0    1    2    1         0      3    1      0
  소고기김밥 1      0          1          0    0    1        0    0    1    0         0      0    0      0
  아이스크림 0      1          0          1    0    0        0    1    1    0         0      1    0      0
  오뎅       0      0          0          0    1    0        1    0    1    0         0      0    0      0
  우동       1      0          1          0    0    1        0    0    1    0         0      0    0      0
  참치김밥   0      0          0          0    1    0        1    0    1    0         0      0    0      0
  치킨       0      1          0          1    0    0        0    1    1    0         0      1    0      0
  커피       1      2          1          1    1    1        1    1    4    0         0      2    0      0
  콜라       0      1          0          0    0    0        0    0    0    2         1      1    1      1
  티라무슈   0      0          0          0    0    0        0    0    0    1         1      0    0      1
  파스타     0      3          0          1    0    0        0    1    2    1         0      3    1      0
  피자       0      1          0          0    0    0        0    0    0    1         0      1    1      0
  햄버거     0      0          0          0    0    0        0    0    0    1         1      0    0      1
```

## 3. 텍스트 전처리와 tm 패키지

[ tm 패키지를 이용한 텍스트 마이닝 예제 2. ]

```
library(tm)
A <- c('포도 바나나 딸기 맥주 비빔밥 여행 낚시 떡볶이 분홍색 듀크 귤')
B <- c('사과 와인 스테이크 배 포도 여행 등산 짜장면 냉면 삼겹살 파란색 듀크 귤 귤')
C <- c('백숙 바나나 맥주 여행 피자 콜라 햄버거 비빔밥 파란색 듀크 귤')
D <- c('귤 와인 스테이크 배 포도 햄버거 등산 갈비 냉면 삼겹살 녹색 듀크')
data <- c(A,B,C,D)
cps <- Corpus(VectorSource(data))
tdm <- TermDocumentMatrix(cps)
inspect(tdm)
m <- as.matrix(tdm)
v <- sort(rowSums(m), decreasing=T)
```

```
> data
[1] "포도 바나나 딸기 맥주 비빔밥 여행 낚시 떡볶이 분홍색 듀크 귤"
[2] "사과 와인 스테이크 배 포도 여행 등산 짜장면 냉면 삼겹살 파란색 듀크 귤 귤"
[3] "백숙 바나나 맥주 여행 피자 콜라 햄버거 비빔밥 파란색 듀크 귤"
[4] "귤 와인 스테이크 배 포도 햄버거 등산 갈비 냉면 삼겹살 녹색 듀크"
> cps
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 4
> tdm
<<TermDocumentMatrix (terms: 24, documents: 4)>>
Non-/sparse entries: 41/55
Sparsity           : 57%
Maximal term length: 4
Weighting          : term frequency (tf)
```

## 3. 텍스트 전처리와 tm 패키지

```
inspect(tdm)

m <- as.matrix(tdm)

v <- sort(rowSums(m), decreasing=T)
```

```
> inspect(tdm)
<<TermDocumentMatrix (terms: 24, documents: 4)>>
Non-/sparse entries: 41/55
Sparsity           : 57%
Maximal term length: 4
Weighting          : term frequency (tf)
Sample             :
          Docs
Terms      1 2 3 4
  냉면     0 1 0 1
  듀크     1 1 1 1
  등산     0 1 0 1
  맥주     1 0 1 0
  바나나   1 0 1 0
  비빔밥   1 0 1 0
  삼겹살   0 1 0 1
  스테이크 0 1 0 1
  여행     1 1 1 0
  포도     1 1 0 1
```

```
> m
          Docs
Terms      1 2 3 4
  낚시     1 0 0 0
  듀크     1 1 1 1
  딸기     1 0 0 0
  떡볶이   1 0 0 0
  맥주     1 0 1 0
  바나나   1 0 1 0
  분홍색   1 0 0 0
  비빔밥   1 0 1 0
  여행     1 1 1 0
  포도     1 1 0 1
  냉면     0 1 0 1
  등산     0 1 0 1
  사과     0 1 0 0
  삼겹살   0 1 0 1
  스테이크 0 1 0 1
  와인     0 1 0 1
  짜장면   0 1 0 0
  파란색   0 1 1 0
  백숙     0 0 1 0
  콜라     0 0 1 0
  피자     0 0 1 0
  햄버거   0 0 1 1
  갈비     0 0 0 1
  녹색     0 0 0 1
```

```
> v
  듀크     여행     포도     맥주   바나나   비빔밥     냉면     등산   삼겹살 스테이크
     4        3        3        2        2        2        2        2        2        2
  와인   파란색   햄버거     낚시     딸기   떡볶이   분홍색     사과   짜장면     백숙
     2        2        2        1        1        1        1        1        1        1
  콜라     피자     갈비     녹색
     1        1        1        1
```

## 3. 텍스트 전처리와 tm 패키지

```
m1 <- as.matrix(weightTf(tdm))
m2. <- as.matrix(weightTfIdf(tdm))
m1;m2.
```

단어 가중치 : 문서에서 어떤 단어의 중요도를 평가하기 위해 사용되는 통계적인 수치

TF : Term Frequency(단어빈도)

IDF : Inverse Document Frequency(역문서빈도)

DF : Document Frequency(문서빈도)

TFIDF : TF X IDF

특정 문서 내에서 단어 빈도가 높을 수록, 전체 문서들엔 그 단어를 포함한 문서가 적을 수록 TFIDF 값이 높아지게 된다.

즉, 문서 내에서 해당 단어의 중요도는 커지게 된다.

```
> m1
        Docs
Terms    1 2 3 4
  낚시   1 0 0 0
  듀크   1 1 1 1
  딸기   1 0 0 0
  떡볶이 1 0 0 0
  맥주   1 0 1 0
  바나나 1 0 1 0
  분홍색 1 0 0 0
  비빔밥 1 0 1 0
  여행   1 1 1 0
  포도   1 1 0 1
  넹면   0 1 0 1
  등산   0 1 0 1
  사과   0 1 0 0
  삼겹살 0 1 0 1
  스테이크 0 1 0 1
  와인   0 1 0 1
  짜장면 0 1 0 0
  파란색 0 1 1 0
  백숙   0 0 1 0
  콜라   0 0 1 0
  피자   0 0 1 0
  햄버거 0 0 1 1
  갈비   0 0 0 1
  녹색   0 0 0 1
```

```
> m2
        Docs
Terms          1          2          3          4
  낚시   0.20000000 0.00000000 0.00000000 0.00000000
  듀크   0.00000000 0.00000000 0.00000000 0.00000000
  딸기   0.20000000 0.00000000 0.00000000 0.00000000
  떡볶이 0.20000000 0.00000000 0.00000000 0.00000000
  맥주   0.10000000 0.00000000 0.10000000 0.00000000
  바나나 0.10000000 0.00000000 0.10000000 0.00000000
  분홍색 0.20000000 0.00000000 0.00000000 0.00000000
  비빔밥 0.10000000 0.00000000 0.10000000 0.00000000
  여행   0.04150375 0.03773068 0.04150375 0.00000000
  포도   0.04150375 0.03773068 0.00000000 0.04150375
  넹면   0.00000000 0.09090909 0.00000000 0.10000000
  등산   0.00000000 0.09090909 0.00000000 0.10000000
  사과   0.00000000 0.18181818 0.00000000 0.00000000
  삼겹살 0.00000000 0.09090909 0.00000000 0.10000000
  스테이크 0.00000000 0.09090909 0.00000000 0.10000000
  와인   0.00000000 0.09090909 0.00000000 0.10000000
  짜장면 0.00000000 0.18181818 0.00000000 0.00000000
  파란색 0.00000000 0.09090909 0.10000000 0.00000000
  백숙   0.00000000 0.00000000 0.20000000 0.00000000
  콜라   0.00000000 0.00000000 0.20000000 0.00000000
  피자   0.00000000 0.00000000 0.20000000 0.00000000
  햄버거 0.00000000 0.00000000 0.10000000 0.10000000
  갈비   0.00000000 0.00000000 0.00000000 0.20000000
  녹색   0.00000000 0.00000000 0.00000000 0.20000000
```

## 3. 텍스트 전처리와 tm 패키지

[ tm 패키지를 이용한 텍스트 마이닝 예제 3 ]

```
html.parsed <- htmlParse("TextofSteveJobs.html")

text <- xpathSApply(html.parsed,

path="//p", xmlValue)

text


text <- text[4:30]

text

docs <- VCorpus(VectorSource(text))

docs
```

```
> text
 [1] "I am honored to be with you today at your commencement from one of the f$
 [2] "The first story is about connecting the dots."                          $
 [3] "I dropped out of Reed College after the first 6 months, but then stayed $
 [4] "It started before I was born. My biological mother was a young, unwed co$
 [5] "And 17 years later I did go to college. But I naively chose a college th$
 [6] "It wasn't all romantic. I didn't have a dorm room, so I slept on the f$
 [7] "Reed College at that time offered perhaps the best calligraphy instructi$
 [8] "None of this had even a hope of any practical application in my life. Bu$
 [9] "Again, you can't connect the dots looking forward; you can only connect$
[10] "My second story is about love and loss."                                $
[11] "I was lucky — I found what I loved to do early in life. Woz and I starte$
[12] "I really didn't know what to do for a few months. I felt that I had let$
[13] "I didn't see it then, but it turned out that getting fired from Apple w$
[14] "During the next five years, I started a company named NeXT, another comp$
[15] "I'm pretty sure none of this would have happened if I hadn't been fire$
[16] "My third story is about death."                                         $
[17] "When I was 17, I read a quote that went something like: "If you live ea$
[18] "Remembering that I'll be dead soon is the most important tool I've eve$
[19] "About a year ago I was diagnosed with cancer. I had a scan at 7:30 in th$
[20] "I lived with that diagnosis all day. Later that evening I had a biopsy, $
[21] "This was the closest I've been to facing death, and I hope it's the cl$
[22] "No one wants to die. Even people who want to go to heaven don't want to$
[23] "Your time is limited, so don't waste it living someone else's life. Do$
[24] "When I was young, there was an amazing publication called The Whole Eart$
[25] "Stewart and his team put out several issues of The Whole Earth Catalog, $
[26] "Stay Hungry. Stay Foolish."                                             $
[27] "Thank you all very much."                                              $
> docs <- VCorpus(VectorSource(text))
> docs
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 27
```

## 3. 텍스트 전처리와 tm 패키지

```
toSpace <- content_transformer(function(x, pattern){return(gsub(pattern, " ", x))})
docs <- tm_map(docs, toSpace, ":")
docs <- tm_map(docs, toSpace, ";")
docs <- tm_map(docs, toSpace, "'")
```

docs[[17]]

docs[[19]]

docs[[17]]$content

docs[[19]]$content

```
> docs[[17]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 460
> docs[[19]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 679
> docs[[17]]$content
[1] "When I was 17, I read a quote that went something like   "If you live eac$
> docs[[19]]$content
[1] "About a year ago I was diagnosed with cancer. I had a scan at 7 30 in the$
```

```
docs <- tm_map(docs, removePunctuation)
text[17]
docs[[17]]$content
```

```
> text[17]
[1] "When I was 17, I read a quote that went something like:  "If you live eac$
> docs[[17]]$content
[1] "When I was 17 I read a quote that went something like   "If you live each$
```

## 3. 텍스트 전처리와 tm 패키지

```
docs <- tm_map(docs, content_transformer(tolower))
docs[[17]]$content
docs <- tm_map(docs, removeNumbers)
docs[[17]]$content
docs <- tm_map(docs, removeWords, stopwords("english"))
docs[[17]]$content
docs <- tm_map(docs,
  stripWhitespace)
docs[[17]]$content
docs <- tm_map(docs,
  stemDocument)
docs[[17]]$content
```

```
> docs <- tm_map(docs, content_transformer(tolower))
> docs[[17]]$content
[1] "     read  quote  went something like  “  live  day      last someday ’ $
> docs <- tm_map(docs, removeNumbers)
> docs[[17]]$content
[1] "     read  quote  went something like  “  live  day      last someday ’ $
> docs <- tm_map(docs, removeWords, stopwords("english"))
> docs[[17]]$content
[1] "     read  quote  went something like  “  live  day      last someday ’ $
> docs <- tm_map(docs, stripWhitespace)
> docs[[17]]$content
[1] " read quote went something like  “ live day last someday ’ ll certainly r$
> docs <- tm_map(docs, stemDocument)
> docs[[17]]$content
[1] "read quot went someth like  “ live day last someday ’ ll certain right”  $
```

## 3. 텍스트 전처리와 tm 패키지

```
tdm <- TermDocumentMatrix(docs)
tdm

inspect(tdm[50:60, 1:5])

termFreq <- rowSums(as.matrix(tdm))
head(termFreq)
termFreq[head(order(termFreq, decreasing=T))]
```

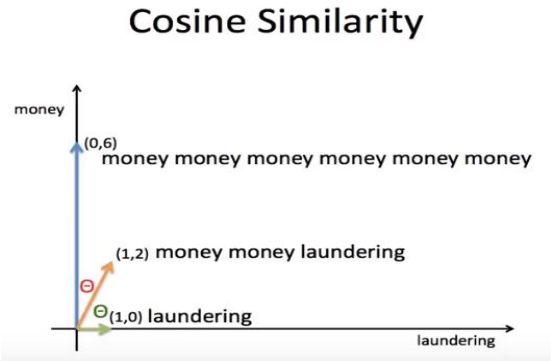
```
> tdm
<<TermDocumentMatrix (terms: 534, documents: 27)>>
Non-/sparse entries: 865/13553
Sparsity           : 94%
Maximal term length: 12
Weighting          : term frequency (tf)
> inspect(tdm[50:60, 1:5])
<<TermDocumentMatrix (terms: 11, documents: 5)>>
Non-/sparse entries: 4/51
Sparsity           : 93%
Maximal term length: 6
Weighting          : term frequency (tf)
Sample             :
        Docs
Terms    1 2 3 4 5
  biolog 0 0 0 2 0
  biopsi 0 0 0 0 0
  birth  0 0 0 1 0
  bit    0 0 0 0 0
  board  0 0 0 0 0
  bob    0 0 0 0 0
  born   0 0 0 1 0
  bottl  0 0 0 0 0
  bought 0 0 0 0 0
  boy    0 0 0 1 0
>
> termFreq <- rowSums(as.matrix(tdm))
> head(termFreq)
   'll     've   "stay across   adopt   adult
    3       4       1       1       4       1
> termFreq[head(order(termFreq, decreasing=T))]
  life colleg   year   drop    one   look
    16     14     12     10     10      9
```

## 4. 문서간 유사도 분석

문서들간에 동일한 단어 또는 비슷한 단어가 얼마나 공통으로 많이 사용 되었나에 따라서 문서간 유사도 분석을 할 수 있다.

(1) 문서의 각 단어들을 수치화 하여 표현한다. – DTM

(2) 문서간 단어들의 차이를 계산한다 – 코사인 유사도, 유클리드 거리

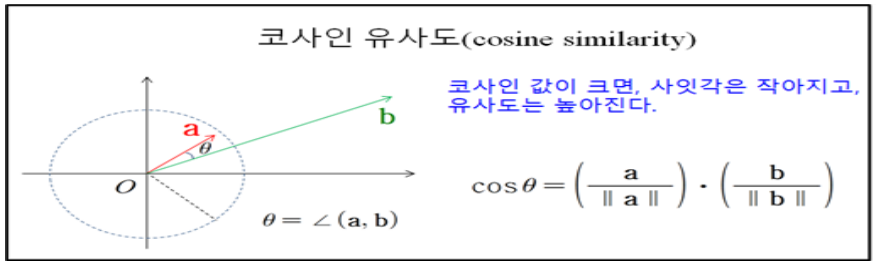


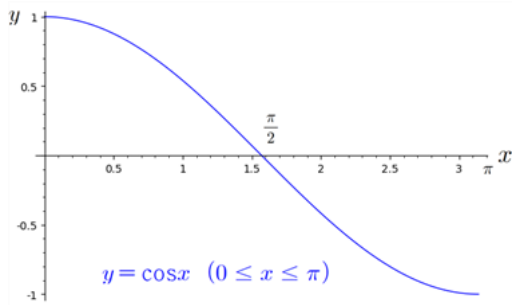

코사인 유사도(Cosine Similarity)

두 벡터 간의 코사인 각도를 이용하여 유사도를 측정한다.

데이터의 코사인 유사도를 계산하여, 만일 코사인 값이 크면, 코사인 함수의 성질에 의해 사잇각은 작아지게 되고,

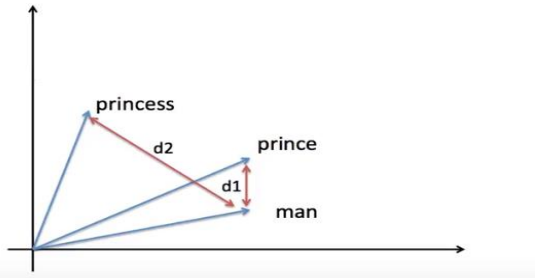
그에 따라 유사도는 높아지게 된다. 이런 방식으로 데이터 사이의 패턴을 분석할 수 있게 된다.




d1 is shorter than d2

man is similar to prince than princess




유클리드 거리(Euclidean distance)

두 점 사이의 유클리드 거리 공식은 피타고라스의 정리를 통해 두 점 사이의 거리를 구하는 것과 동일하다.

$$p = (p_1, p_2, p_3, \ldots, p_n)\text{과 } q = (q_1, q_2, q_3, \ldots, q_n)$$

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

## 4. 문서간 유사도 분석

코사인 거리(Cosine Distance)는 '1 – 코사인 유사도(Cosine Similarity)' 로 계산

**코사인 유사도 (Cosine Similarity) vs. 코사인 거리 (Cosine Distance)**

Cosine similarity $$cosine\ similarity = \frac{X \cdot Y}{\|X\|_2 \cdot \|Y\|_2}$$

Cosine distance $$d_{cosine}(X,Y) = 1 - \frac{X \cdot Y}{\|X\|_2 \cdot \|Y\|_2}$$ , where $\|X\|_2$ is the L2 norm

→ 문서 분류, 군집화 (Text Classification, Clustering) 에 활용

| | | | |
|---|---|---|---|
| Document 1 | 1 | 0 | 5 |
| Document 2 | 4 | 7 | 3 |

$$d_{cosine}(Document_1, Document_2)$$

$$= 1 - \frac{[1,0,5] \cdot [4,7,3]}{\|[1,0,5]\|_2 \cdot \|[4,7,3]\|_2}$$

$$= 1 - \frac{1 \times 4 + 0 \times 7 + 5 \times 3}{\sqrt{1^2+0^2+5^2} \cdot \sqrt{4^2+7^2+3^2}}$$

$$= 1 - \frac{19}{\sqrt{26} \cdot \sqrt{74}}$$

$$= 1 - \frac{19}{5.1 \times 8.6} \approx 1 - 0.433 \approx 0.567$$

## 4. 문서간 유사도 분석

```
install.packages("proxy")
library(proxy)
dd <- NULL
d1 <- c("aaa bbb ccc")
d2 <- c("aaa bbb ddd")
d3 <- c("aaa bbb ccc")
d4 <- c("xxx yyy zzz")
dd <- c(d1, d2, d3, d4)
cps <- Corpus(VectorSource(dd))
dtm <- DocumentTermMatrix(cps)
(m <- as.matrix(dtm))
com <- m %*% t(m)
com
dist(com, method = "cosine") # 코사인 거리(Cosine Distance) : '1 – 코사인 유사도(Cosine Similarity)'
dist(com, method = "Euclidean") # 유클리드 거리
```

```
> (m <- as.matrix(dtm))
     Terms
Docs aaa bbb ccc ddd xxx yyy zzz
   1   1   1   1   0   0   0   0
   2   1   1   0   1   0   0   0
   3   1   1   1   0   0   0   0
   4   0   0   0   0   1   1   1
```

```
> dist(com, method = "cosine") # 코사인 거리(Cosine Distance) : 1 - 코사인 유사도
arity)'
           1          2         3
2 0.06924216
3 0.00000000 0.06924216
4 1.00000000 1.00000000 1.00000000
```

```
> dist(com, method = "Euclidean") # 유클리드 거리
         1        2        3
2 1.732051
3 0.000000 1.732051
4 5.567764 5.099020 5.567764
```

## 5. 텍스트 마이닝의 결과 시각화

텍스트 마이닝의 결과를 시각화 할 때 가장 많이 사용되는 것은 워드 클라우드이다.

워드 클라우드는 단어의 개수를 세어 객수의 크기 값에 따라서 단어의 크기를 차등적으로 출력하여 키워드가 되는 단어를 좀 더 강조하여 출력하는 시각화이다.

```
install.packages("wordcloud")
library(wordcloud)
wordcloud(words,freq,scale=c(4,.5),min.freq=3,max.words=Inf, random.order=TRUE, random.color=FALSE, rot.per=.1,
colors="black",ordered.colors=FALSE,use.r.layout=FALSE, fixed.asp=TRUE, ...)
```

- scale : 빈도가 가장 큰 단어와 가장 빈도가 작은 단어 폰트 사이 크기, scale=c(5,0.2.)
- rot.per=0.1 : 90도 회전해서 보여줄 단어 비율
- min.freq=3, max.words=100 : 빈도 3이상, 100미만 단어 표현
- random.order=F : True(랜덤배치) / False(빈도수가 큰단어를 중앙에 배치)
- random.color=T : True(색상랜덤) / False(빈도수순으로 색상표현)
- colors=색상이름
- family : 폰트

savePlot(szWordCloudImageFile, type="png") : WordCloud 결과를 이미지 파일로 저장

# 텍스트 마이닝

## 5. 텍스트 마이닝의 결과 시각화

```
library(wordcloud)

v <- sort(rowSums(m), decreasing=T)
v
```

```
> v
  듀크   여행   포도   맥주 바나나 비빔밥   냉면   등산 삼겹살 스테이크
     4      3      3      2      2      2      2      2      2      2
 와인 파란색 햄버거   낚시   딸기 떡볶이 분홍색   사과 짜장면   백숙
     2      2      2      1      1      1      1      1      1      1
 콜라   피자   갈비   녹색
     1      1      1      1
>
```

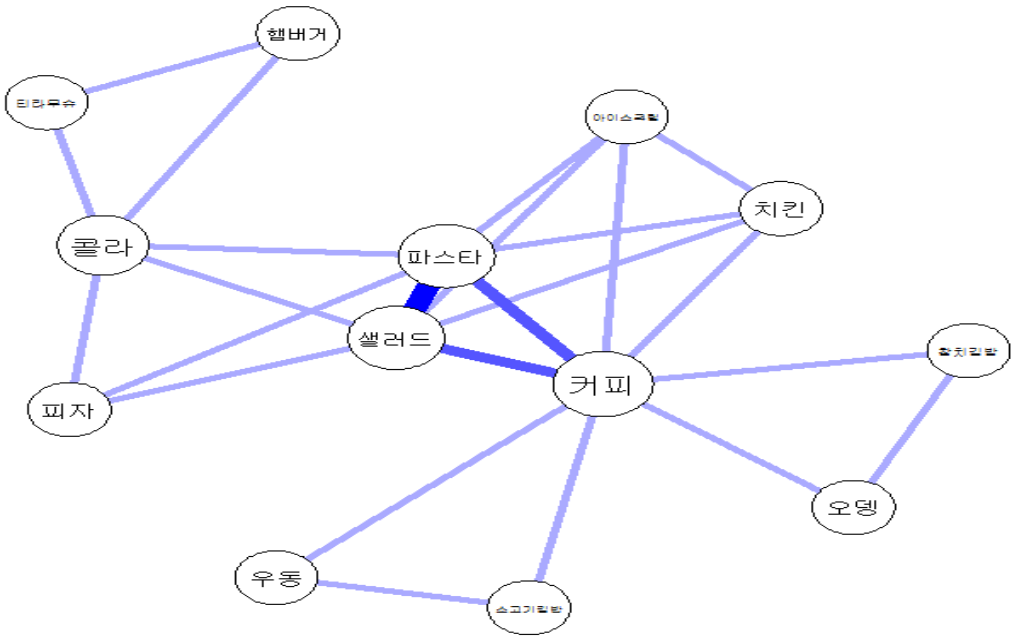
```
wordcloud(names(v), v, min.freq = 2., random.order = FALSE, rot.per = 0.1, scale = c(4, 1),
colors = c("pink", "blue", "green", "red"))
```

## 4. 텍스트 마이닝의 결과 시각화

동시출현(Co-occurrence)이란 한 문장, 문단 또는 텍스트 단위에서 같이 출현한 단어를 가리킵니다. 단어의 연결성(collocation)을 찾는 데 활용된다. 이 개념에서 출발한 동시출현 네트워크(Co-occurrence networks)는 특정 텍스트 단위에서 공동으로 출현한 단어의 집합적 상호 연결을 표현하는 방식이다.

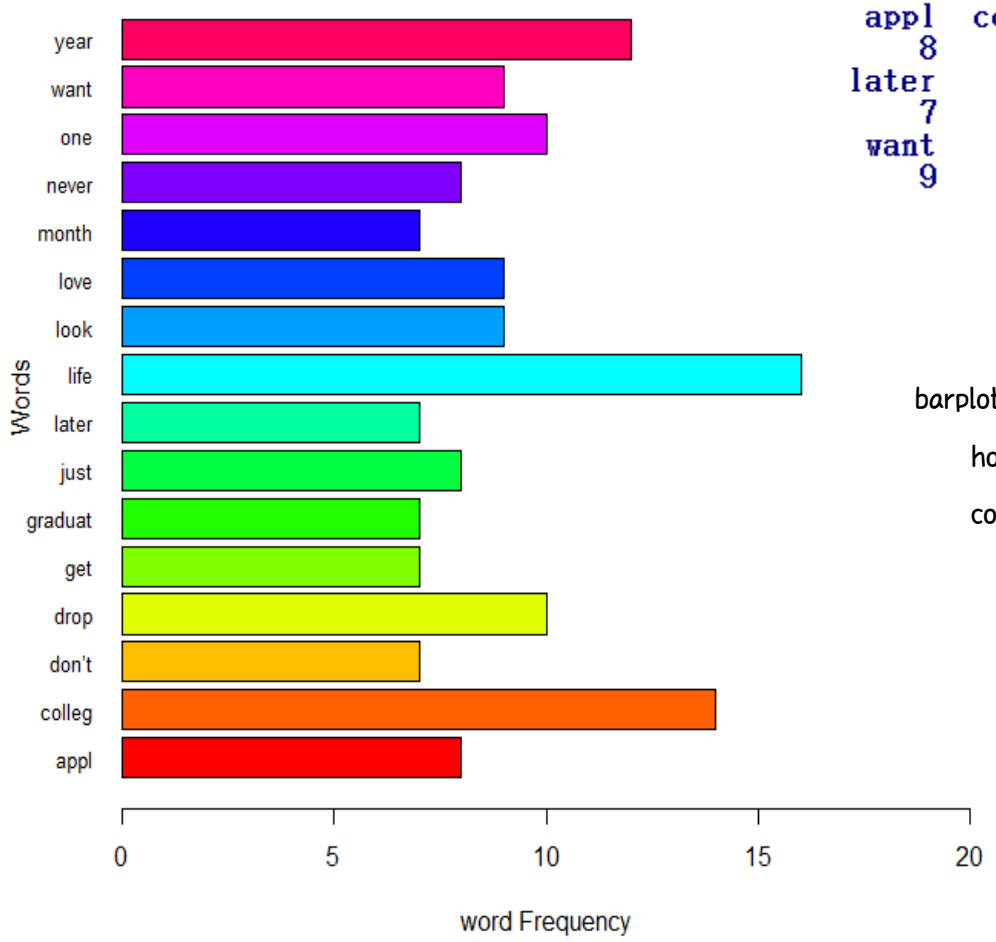
나타나는 단어를 모두 표시한 뒤, 둘 사이를 선으로 연결해 나가다 보면 단어의 네트워크를 만들 수 있다. qgraph 패키지의 qgraph() 함수를 사용한다.

```
install.packages("qgraph")
library(qgraph)

qgraph(com, labels=rownames(com), diag=F,
layout='spring',  edge.color='blue',
vsize=log(diag(com)*800))
```



## 4. 텍스트 마이닝의 결과 시각화

단어의 출현횟수를 바 그래프로 그린다.




```
> termFreq[termFreq>=7]
   appl   colleg   don't    drop     get graduat    just
      8      14       7      10       7       7       8
  later    life    look    love   month   never     one
      7      16       9       9       7       8      10
   want    year
      9      12
```

```
barplot(termFreq[termFreq >= 7],
        horiz=T, las=1, cex.names=0.8,
        col=rainbow(16), xlab="word Frequency", ylab="Words")
```