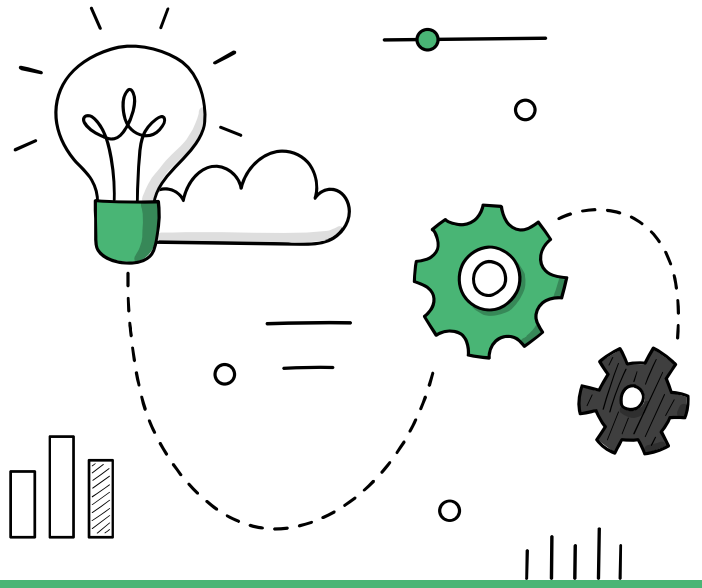
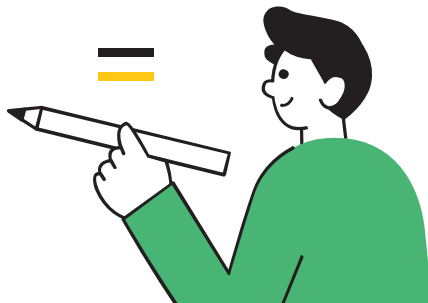
디지털 아카데미 빅데이터 분석

비정형 데이터(텍스트) 분석 _Text Analysis



CONTENTS

모듈	내용
M1.텍스트 분석(Text Analysis) 개요	- 텍스트 분석 개념 이해 / 텍스트 분석의 배경 / 텍스트 분석의 과정
M2.텍스트 전처리(Pre-processing)	- 토큰화 / 정제 및 정규화
M3.키워드 분석(Keyword Analysis)	- 텍스트 전처리 / 형태소 분석 / 빈도 분석
M4.텍스트 벡터화(Text Vectorization)	- 원-핫 인코딩 / TF-IDF / 단어 임베딩
M5.군집 분석(Cluster Analysis)	- 텍스트 유사도 / 군집화 데이터 전처리 / Word2Vec 생성 / Scikit-learn을 이용한 군집화
M6.감정분석(Sentiment Analysis)	- afinn 감정 어휘 사전 및 Scikit-learn을 이용한 기계학습 감적분석



1. 텍스트 분석 개요

- 텍스트 분석의 배경
- 텍스트 분석의 과정

텍스트 분석(Text Analysis) 개요

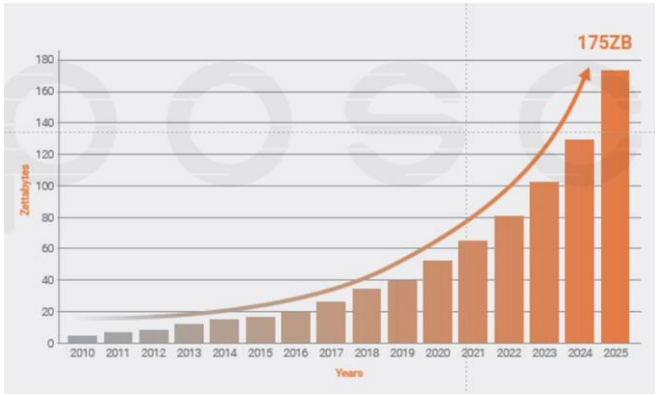
● 텍스트 분석이란?

- 텍스트 분석은 비정형 데이터 마이닝의 유형 중 하나
- 비정형 및 반정형 데이터에 대하여 자연어 처리 기술과 문서처리 기술을 적용하여 유용한 정보를 추출, 가공하는 것

● 텍스트 분석의 배경

- 최근 비정형 데이터가 폭발적으로 증가
- IDC(International Data Corporation)에서는 세계에서 생성, 수집, 소비되는 데이터 양이 2020년 50ZB(제타바이트)에서 2025년에는 175ZB까지 폭증할 것이라 전망

- 데이터 사용 전망

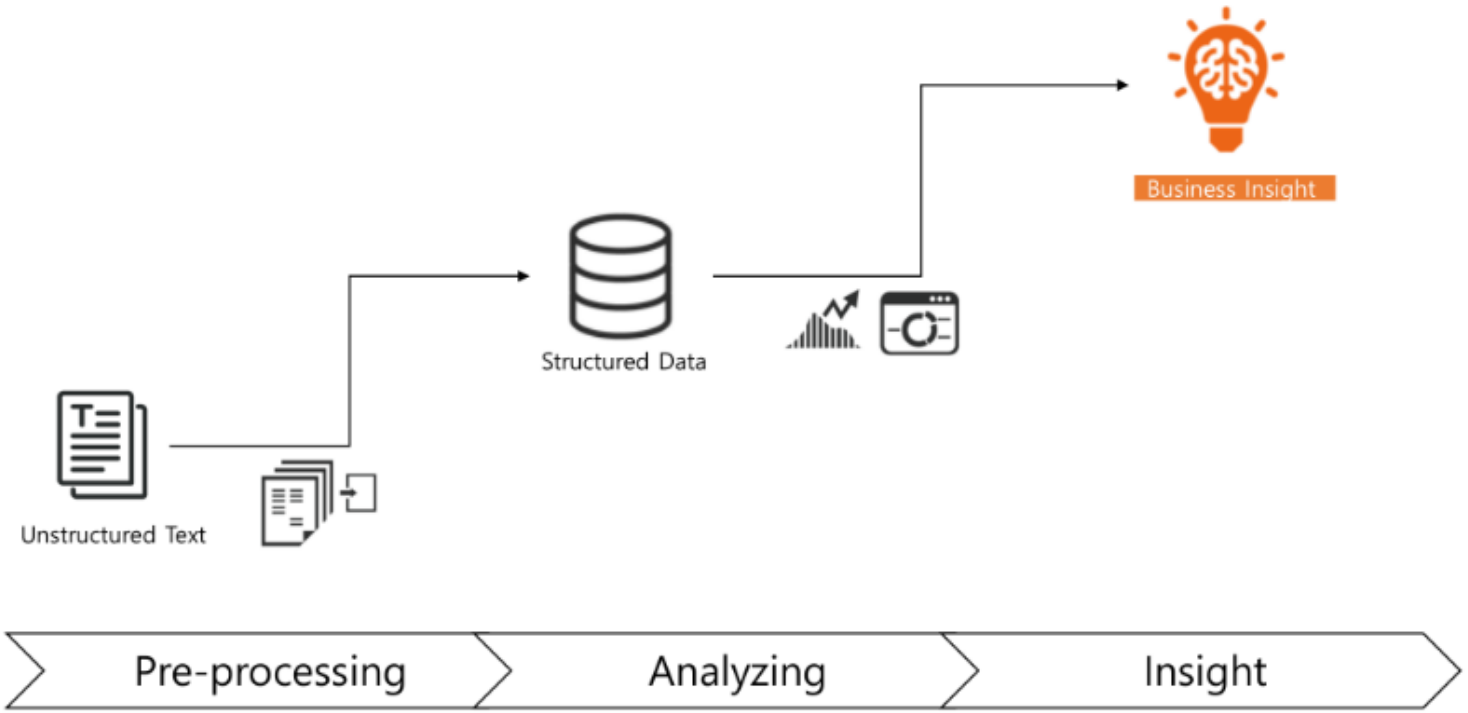


1ZB = 1TB HDD 10억 7,374만개 / 약 80%는 비정형 데이터
[출처] https://blog.his21.co.kr/582

텍스트 분석(Text Analysis) 개요

● 텍스트 분석의 과정

- **텍스트 분석은 일반적으로 3단계의 프로세스로 진행**



[출처] https://www.smu.ac.kr/flexer/index.jsp?ftype=pdf&attachNo=495514

2. 텍스트 전처리

- 토큰화
- 정제 및 정규화
- 표제어 추출
- 어간 추출

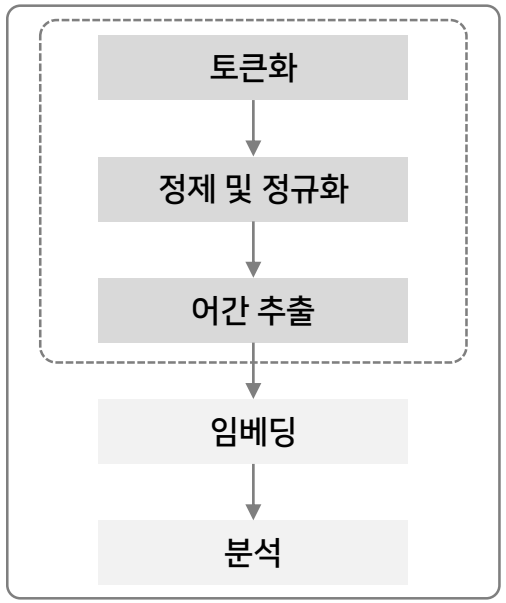
텍스트 전처리(Pre-processing)란?

● 텍스트 전처리 이해

- 텍스트 분석을 진행하기 전에 분석 목적에 맞게 변형,가공하는 과정을 의미하는 것
- 적용 패키지 : 파이썬 내장 함수와 더불어 한글은 KoNLPy, 영어는 NLTK 등으로 전처리

● 전처리 절차

- 말뭉치(코퍼스)의 형태나 분석하고자 하는 목적에 따라 유연하게 적용



한국어 KoNLPy [실습] Kiwipiepy

한국어 자연어 처리를 위한 패키지로 자주 사용되며 다양한 형태 분석기를 모아놓아 활용도가 높다.

영어 NLTK

다양한 기능 및 예제를 가지고 있으며 실무 및 연구에서도 많이 사용, 한국어를 지원하지 않는 영문 모듈이다.

토큰화(Tokenization)

● 토큰화 이해

▪ 토큰화의 개념 및 처리 방법

- 토큰화 : 말뭉치(corpus)에서 텍스트를 토큰(token)이라 불리는 의미있는 단위로 나누는 것
  (단어 또는 문장 단위로 토큰화를 진행)

· 말뭉치(corpus) : 특정한 목적을 가지고 수집한 텍스트 데이터

· 토큰(token) : 더 이상 분해될 수 없는 최소한의 의미를 갖는 단위

- 특수문자에 대한 처리

· 단어에 일반적으로 사용되는 알파벳, 숫자와는 다르게 특수문자는 별도의 처리가 필요

· 일괄적으로 단어의 특수문자를 제거하는 방법도 있지만 특수문자가 단어에 특별한 의미를 가질 때 이를 학습에 반영시키지 못할 수도 있음

· 특수문자에 대한 일괄적인 제거보다는 데이터의 특성을 파악하고, 처리를 하는 것이 중요

토큰화(Tokenization)

▪ 토큰화의 개념 및 처리 방법

- 특정 단어에 대한 토큰 분리 방법

· 한 단어지만 토큰으로 분리할 때 판단되는 문자들로 이루어진 we’re, United Kingdom 등의 단어는 어떻게 분리해야 할지 선택이 필요

· we’re 는 한 단어이나 분리해도 단어의 의미에 별 영향을 끼치진 않지만 United Kingdom은 두 단어가 모여 특정 의미를 가리켜 분리해서는 안됨

· 사용자가 단어의 특성을 고려해 토큰을 분리하는 것이 학습에 유리

토큰화(Tokenization)

▪ 한국어 토큰화의 특징

- 한국어는 어간에 접사가 붙어 의미와 문법적 기능이 변화하는 '교착어'로 인공지능 모델링이 어려움

1) 접사 추가에 따른 의미 발생
- 사과(어간) + 를(접사) → '사과' 목적어
- 사과(어간) + 가(접사) → '사과' 주어

접사에 따라 의미가 달라짐

2) 단어의 순서를 바꾸어도 맥락을 이해하는데 문제 없음
- 사과(어간) + 를(접사) → '사과' 목적어
- 사과(어간) + 가(접사) → '사과' 주어

컴퓨터가 서로 다른 문장을
동일한 뜻으로 받아들이기 어려움

3) 주어가 생략되어도 이해하는데 큰 문제 없음
- 영어 : Did you have lunch?
- 한국어 : 점심 먹었어?

컴퓨터가 맥락을 파악하는 것이 까다로움

4) 띄어쓰기를 지키지 않아도 맥락을 이해할 수 있음
- 나는밥을먹으러간다.
- 점심먹었어?

컴퓨터가 문장을 정확하게 인식하기 어려움

※ 언어 분류

분류	언어	특징
교착어	한국어, 일본어, 몽골어	어간에 접사가 붙어, 단어를 이루고 의미가 문법적 기능이 정해짐
굴절어	라틴어, 독일어, 러시아어	단어의 형태가 변함으로써 문법적 기능이 정해짐
고립어	영어, 중국어	어순에 따라 단어의 문법적 기능이 정해짐

토큰화(Tokenization)

- 한국어 토큰화의 특징

  - 한국어 처리를 위한 패키지

    · KoNLPy : 한글을 쉽고 간결하게 처리할 수 있도록 만들어진 오픈소스 패키지 -> https://konlpy.org/ko/latest/

    · kiwipiepy : 한국어 형태소 분석기인 Kiwi(Korean Intelligent Word Identifier)의 Python 모듈 -> https://bab2min.github.io/kiwipiepy/

토큰화(Tokenization)

● 단어 토큰화(Word Tokenization)

▪ 파이썬 내장 함수로 텍스트 처리

```
sentence = ‘When you have faults, do not fear to abandon them.’
#허물이 있다면, 버리기를 두려워 말라. -공자-
```

- 공백을 기준으로 단어를 분리

```
tokens = sentence.split() #디폴트는 공백
print(tokens)
```

```
[‘When’, ‘you’, ‘have’, ‘faults,’, ‘do’, ‘not’, ‘fear’, ‘to’, ‘abandon’, ‘them.’]
```

토큰화(Tokenization)

- **NLTK 영어 토큰화**
  - 'nltk' 패키지의 tokenize 모듈을 사용해 손쉽게 구현 가능
  - 단어 토큰화는 word_tokenize() 함수를 사용해 구현 가능

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
```

  - 토큰화

```
tokens = word_tokenize(sentence)
print(tokens)
```

```
['When', 'you', 'have', 'faults,', 'do', 'not', 'fear', 'to', 'abandon', 'them.']
```

토큰화(Tokenization)

- NLTK 영어 토큰화

  - PoS(Parts-of-Speech) 태깅: PoS는 품사를 의미하며, PoS 태깅은 문장 내에서 단어에 해당하는 각 품사를 태깅

```
nltk.download('averaged_perceptron_tagger')
p_tag = nltk.pos_tag(tokens)
print(p_tag)
```

```
[('When', 'WRB'), ('you', 'PRP'), ('have', 'VBP'), ('faults', 'NNS'), (',', ','), ('do', 'VBP'),
('not', 'RB'), ('fear', 'VB'), ('to', 'TO'), ('abandon', 'VB'), ('them', 'PRP'), ('.', '.')]
```

토큰화(Tokenization)

▪ NLTK PoS 태그 리스트

No	Tag	Description	설명
1	CC	Coordinating conjunction	
2	CD	Cardinal number	
3	DT	Determiner	한정사
4	EX	Existential there	
5	FW	Foreign word	외래어
6	IN	Preposition	전치사
7	JJ	Adjective	형용사
8	JJR	Adjective, comparative	헝용사, 비교급
9	JJS	Adjective, superlative	형용사, 최상급
10	LS	List item marker	
11	MD	Modal	
12	NN	Noun, singular or mass	명사, 단수형
13	NNS	Noun, plural	명사, 복수형
14	NNP	Proper noun, singular	고유명사, 단수형
15	NNPS	Proper noun, plural	고유명사, 복수형
16	PDT	Predeterminer	전치한정사
17	POS	Possessive ending	소유형용사
18	PRP	Personal pronoun	인칭 대명사

No	Tag	Description	설명
19	PRP$	Possessive pronoun	소유 대명사
20	RB	Adverb	
21	RBR	Adverb, comparative	부사, 비교급
22	RBS	Adverb, superlative	부사, 최상급
23	RP	Particle	
24	SYM	Symbol	기호
25	TO	to	
26	UH	Interjection	감탄사
27	VB	Verb, base form	동사, 원형
28	VBD	Verb, past tense	동사, 과거형
29	VBG	Verb, gerund or present participle	동사, 현재분사
30	VBN	Verb, past participle	동사, 과거분사
31	VBP	Verb, non-3rd person singular present	동사, 비3인칭
32	VBZ	Verb, 3rd person singular present	동사, 3인칭
33	WDT	Wh-determiner	
34	WP	Wh-pronoun	
35	WP$	Possessive wh-pronoun	
36	WRB	Wh-adverb	

토큰화(Tokenization)

- **Kiwipiepy 한국어 형태소 분석**
  - 한국어는 공백으로 단어를 분리해도 조사, 접속사 등이 남아 분석에 어려움
  - 이를 해결해주는 한국어 토큰화는 조사, 접속사를 분리해주거나 제거
  - 형태소 분석(Morphological Analysis) : 더 이상 분해될 수 없는 최소한의 의미를 갖는 단위인 형태소를 추출
    - 형태소에는 어간(stem)과 접사(affix)가 있음
    - 어간 : 단어의 의미를 담고 있는 단어의 핵심 부분
    - 접사 : 단어에 추가적인 의미를 주는 부분

```
s = '자신감 있는 표정을 지으면 자신감이 생긴다.'
```

  - kiwi 객체 생성

```
from kiwipiepy import Kiwi
kiwi = Kiwi()
```

토큰화(Tokenization)

- Kiwipiepy 한국어 형태소 분석

  - 형태소 분석

```
kiwi.tokenize(s)
```

```
[Token(form='자신감', tag='NNG', start=0, len=3), Token(form='있', tag='VA', start=4, len=1),
Token(form='는', tag='ETM', start=5, len=1), Token(form='표정', tag='NNG', start=7, len=2),
Token(form='을', tag='JKO', start=9, len=1), Token(form='짓', tag='VV-I', start=11, len=1),
Token(form='으면', tag='EC', start=12, len=2), Token(form='자신감', tag='NNG', start=15, len=3),
Token(form='이', tag='JKS', start=18, len=1),Token(form='생기', tag='VV', start=20, len=2),
Token(form='ㄴ다', tag='EF', start=21, len=2), Token(form='.', tag='SF', start=23, len=1)]
```

```
clean_token = []

tokens = kiwi.tokenize(s)
for t in tokens:
    clean_token.append(t.form) #형태소만 추출
print(clean_token)
```

```
['자신감', '있', '는', '표정', '을', '짓', '으면', '자신감', '이', '생기', 'ㄴ다', '.']
```

토큰화(Tokenization)

- Kiwipiepy 한국어 형태소 분석

  - 명사 추출(조사, 접속사 등을 제거)

```
clean_token = []

for t in tokens:
    if t.tag[0] == 'N': #명사만 추출
        clean_token.append(t.form)

print(clean_token)
```

```
['자신감', '표정', '자신감']
```

토큰화(Tokenization)

▪ Kiwipiepy 한국어 형태소 분석

대분류	Tag	설명
체언(N)	NNG	일반 명사
	NNP	고유 명사
	NNB	의존 명사
	NR	수사
	NP	대명사
용언(V)	VV	동사
	VA	형용사
	VX	보조 용언
	VCP	긍정 지시사(이다)
	VCN	부정 지시사(아니다)
관형사	MM	관형사
부사(MA)	MAG	일반 부사
	MAJ	접속 부사
감탄사	IC	감탄사

대분류	Tag	설명
조사(J)	JKS	주격 조사
	JKC	보격 조사
	JKG	관형격 조사
	JKO	목적격 조사
	JKB	부사격 조사
	JKV	호격 조사
	JKQ	인용격 조사
	JX	보조사
	JC	접속 조사
어미(E)	EP	선어말 어미
	EF	종결 어미
	EC	연결 어미
어미(E)	ETN	명사형 전성 어미
	ETM	관형형 전성 어미
접두사	XPN	체언 접두사
접미사(XS)	XSN	명사 파생 접미사
	XSV	동사 파생 접미사
	XSA	형용사 파생 접미사
	XSM	부사 파생 접미사*

대분류	Tag	설명
어근	XR	어근
부호, 외국어, 특수문자(S)	SF	종결 부호(. ! ?)
	SP	구분 부호(, / : ;)
	SS	인용 부호 및 괄호 (' " ( ) [ ] < > { } — ‘ ’ “ ” ≪ ≫ 등)
	SSO	SS 중 여는 부호*
	SSC	SS 중 닫는 부호*
	SE	줄임표(…)
	SO	붙임표(- ~)
	SW	기타 특수 문자
	SL	알파벳(A-Z a-z)
	SH	한자
	SN	숫자(0-9)
분석 불능	UN	분석 불능*
웹(W)	W_URL	URL 주소*
	W_EMAIL	이메일 주소*
	W_HASHTAG	해시태그(#abcd)*
	W_MENTION	멘션(@abcd)*
	W_SERIAL	일련번호(전화번호, 통장번호, IP주소 등)*
기타	Z_CODA	덧붙은 받침*

토큰화(Tokenization)

● 문장 토큰화(Setence Tokenization)

▪ 줄바꿈 문자(‘₩n’)를 기준으로 문장을 분리

```
s = ‘Life is like a box of chocolates.\nYou never know what you’re gonna get.’
#인생은 마치 초콜릿 상자와도 같다. 당신이 무엇을 얻게 될지 모르기 때문이다.

tokens = s.split(‘\n’)
print(tokens)
```

```
[‘Life is like a box of chocolates.’, ‘You never know what you’re gonna get.’]
```

▪ nltk 이용

```
from nltk.tokenize import sent_tokenize

tokens = sent_tokenize(s)
print(tokens)
```

```
[‘Life is like a box of chocolates.’, ‘You never know what you’re gonna get.’]
```

토큰화(Tokenization)

- **Kiwipiepy 이용(한국어)**

```
text = '한 가지 생각을 선택하라. 그 생각을 당신의 삶으로 만들어라. 그걸 생각하고, 꿈꾸고, 그에 기반해서 살아가라. 당신의 몸의 모든 부분, 뇌, 근육, 신경을 그 생각으로 가득 채우고 다른 생각은 다 내버려둬라. 이것이 성공하는 방법이다.'

sent_kr = kiwi.split_into_sents(text)
sent_kr
```

```
[Sentence(text='한 가지 생각을 선택하라.', start=0, end=14, tokens=None, subs=[]),
Sentence(text='그 생각을 당신의 삶으로 만들어라.', start=15, end=34, tokens=None, subs=[]),
Sentence(text='그걸 생각하고, 꿈꾸고, 그에 기반해서 살아가라.', start=35, end=62, tokens=None, subs=[]),
Sentence(text='당신의 몸의 모든 부분, 뇌, 근육, 신경을 그 생각으로 가득 채우고 다른 생각은 다 내버려둬라.',
start=63, end=117,tokens=None, subs=[]),
Sentence(text='이것이 성공하는 방법이다.', start=118, end=132, tokens=None, subs=[])]
```

토큰화(Tokenization)

- Kiwipiepy 이용(한국어)

```
sentence = []

sent_kr = kiwi.split_into_sents(text)

for s in sent_kr:
    sentence.append(s.text)

print(sentence)
```

```
['한 가지 생각을 선택하라.', '그 생각을 당신의 삶으로 만들어라.', '그걸 생각하고, 꿈꾸고, 그에 기반해서 살아가라.', '당신의 몸의 모든 부분, 뇌, 근육, 신경을 그 생각으로 가득 채우고 다른 생각은 다 내버려둬라.', '이것이 성공하는 방법이다.']
```

정제(Cleaning) 및 정규화(Normalizaion)

● 정제 및 정규화 이해

- 정제 : 데이터에서 손상, 부정확한 부분, 관련이 없는 부분을 식별한 뒤에 수정, 삭제 및 대체 등을 하는 과정
- 정규화 : 데이터에서 표현방법이 다른 단어들을 통합시켜 같은 단어로 통합하는 과정
- 토큰화 작업 전·후 실시하나 한글에서는 명사 만 추출하여 사용할 경우 어느 정도 정제화가 됨
- 정제 및 정규화 방법

01

표기가 다른 단어들의 통합

USA, UA → USA

02

대, 소문자 통합

korea, Korea → KOREA

03

불필요한 단어의 제거

길이가 짧은 단어
빈도수가 적은 단어

04

정규 표현식 활용

규칙에 기반 한 번에 제거

정제(Cleaning) 및 정규화(Normalizaion)

● 불용어(Stopword) 제거

- 불용어(stop words): 문장 형성에 중요한 구문록적 가치를 지니고 있지만 무시할 수 있거나 최소한의 의미적 가치를 지닌 단어
- 주로 접속사, 관사, 부사, 대명사, 일반동사 등 자연어에 포함되어 있는 정보를 담지 않는 단어들이 해당됨
- 영어의 전치사(on, in), 한국어의 조사(을, 를) 등은 분석에 필요하지 않은 경우가 많음
- 길이가 짧은 단어, 등장 빈도 수가 적은 단어들도 분석에 큰 영향을 주지 않음
- 일반적으로 사용되는 도구들은 해당 단어들을 제거해주지만 완벽하게 제거되지는 않음
- 사용자가 불용어 사전을 만들어 해당 단어들을 제거하는 것이 좋음
- 도구들이 걸러주지 않는 전치사, 조사 등을 불용어 사전을 만들어 불필요한 단어들을 제거

정제(Cleaning) 및 정규화(Normalizaion)

- **단어를 분리하여 불용어로 설정**

```
stop_words = ‘on in the’

stop_words = stop_words.split(‘ ’)
print(stop_words)
```

```
[‘on’, ‘in’, ‘the’]
```

정제(Cleaning) 및 정규화(Normalizaion)

- **설정된 불용어를 제거**

```
sentence = ‘singer on the stage’

sentence = sentence.split(‘ ’)
print(sentence)
```

```
[‘singer’, ‘on’, ‘the’, ‘stage’]
```

```
nouns = []

for noun in sentence:
    if noun not in stop_words:
        nouns.append(noun)
print(nouns)
```

```
[‘singer’, ‘stage’]
```

정제(Cleaning) 및 정규화(Normalizaion)

- **nltk 패키지의 불용어 리스트 사용**

```
from nltk.corpus import stopwords
nltk.download('stopwords')
```

- 불용어 리스트 확인

```
stop_words = stopwords.words('english')
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',
'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up',
'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where',
'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only',
'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'ouldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't",
'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't",
'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

정제(Cleaning) 및 정규화(Normalizaion)

- nltk 패키지의 불용어 리스트 사용

- 토큰화

```
s = ‘If you do not walk today. you will have to run tomorrow’

words = word_tokenize(s)
print(words)
```

```
[‘If’, ‘you’, ‘do’, ‘not’, ‘walk’, ‘today’, ‘.’, ‘you’, ‘will’, ‘have’, ‘to’, ‘run’, ‘tomorrow’]
```

- 불용어 제거

```
no_stopwords = []
for w in words:
    if w not in stop_words:
        no_stopwords.append(w)
print(no_stopwords)
```

```
[‘If’, ‘walk’, ‘today’, ‘run’, ‘tomorrow’]
```

정제(Cleaning) 및 정규화(Normalizaion)

- **Kiwipiepy 패키지의 불용어 리스트 사용(한국어)**

```
from kiwipiepy.utils import Stopwords

stopwords = Stopwords()
token_kr = []

token_text = kiwi.tokenize(text, stopwords=stopwords)

for token in token_text:
    token_kr.append(token.form)

print(token_kr)
```

```
['가지', '생각', '선택', '라', '생각', '당신', '삶', '만들', '어라', '그거', 'ㄹ', '생각', '꿈꾸', '기반', '살아가', '어라', '당신', '몸', '모든', '부분', '뇌', '근육', '신경', '생각', '가득', '채우', '다른', '생각', '다', '내버리', '두', '어라', '이것', '성공', '방법']
```

3. 키워드 분석

- 텍스트 전처리
- 형태소 분석
- 빈도 분석

키워드 분석(Keyword Analysis)이란?

● 키워드 분석 이해

- 키워드 : 텍스트 자료의 중요한 내용을 압축적으로 제시하는 단어 또는 문구
- 키워드 분석 : 불용어 제거와 어간추출 및 형태소 분석 등을 시행한 후 텍스트에서 많이 등장하는 형태소의 등장 빈도를 분석함으로써 핵심어를 추출하는 것
- 특정 텍스트 자료에 많이 나타나는 형태소가 그 텍스트 주제를 표출할 가능성이 높다는 가정에 기초
- 빈도 분석에서 영어의 전치사나 한국어의 조사와 같이 의미를 별로 담고 있지 않은 불용어는 제외
- 키워드 분석은 텍스트의 주제 추정, 텍스트 유사도, 검색 엔진의 검색 결과 우선 순위 측정 등 다양하게 사용

데이터 전처리

● 데이터셋 파일 읽기

▪ 네이버 영화 리뷰 데이터

[데이터 출처] https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt

```
f = open('data_set/ratings.txt', 'r', encoding='utf-8')
raw = f.readlines() #개행문자를 기준으로 라인 단위 리스트로 반환

print(raw[:3])
```

```
['id\tdocument\tlabel\n', '8112052\t어릴때보고 지금다시봐도 재밌어요ㅋㅋ\t1\n', '8132799\t디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업이 부러웠는데. 사실 우리나라에서도 그 어려운시절에 끝까지 열정을 지킨 노라노 같은 전통이있어 저와 같은 사람들이 꿈을 꾸고 이뤄나갈 수 있다는 것에 감사합니다.\t1\n']
```

데이터 전처리

- **리뷰 데이터만 추출**

```
reviews = []
for i in raw:
    reviews.append(i.split('\t')[1]) #\t(탭)을 기준을 1번째만 추출하여 append
print(reviews[:5])
```

```
['document', '어릴때보고 지금다시봐도 재밌어요ㅋㅋ', '디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업이 부러웠는데. 사실 우리나라에서도 그 어려운시절에 끝까지 열정을 지킨 노라노 같은 전통이있어 저와 같은 사람들이 꿈을 꾸고 이뤄나갈 수 있다는 것에 감사합니다.', '폴리스스토리 시리즈는 1부터 뉴까지 버릴께 하나도 없음.. 최고.', '와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이런게 진짜 영화지']
```

- **'document' 제외**

```
reviews = reviews[1:]
print(reviews[:5])
```

```
['어릴때보고 지금다시봐도 재밌어요ㅋㅋ', '디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업이 부러웠는데. 사실 우리나라에서도 그 어려운시절에 끝까지 열정을 지킨 노라노 같은 전통이있어 저와 같은 사람들이 꿈을 꾸고 이뤄나갈 수 있다는 것에 감사합니다.', '폴리스스토리 시리즈는 1부터 뉴까지 버릴께 하나도 없음.. 최고.', '와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이런게 진짜 영화지', '안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화.']
```

형태소 분석

● 불용어(Stopwords) 제거를 위한 불용어 사전 만들기

- 형태소 분석을 통해 조사, 접속사 등의 제거 가능
- 하지만 한국어는 명사에서도 상당히 많은 불필요한 단어들이 포함
- 사용자가 직접 불용어 사전을 유지하면서 불필요한 단어 제거 필요
- 불용어 예: '영화 전 난 일 걸 뭐 줄 만 건 분 개 끝 잼 이거 번 중 듯 때 게 내 말 나 수 거 점 것'
- 빈도가 너무 커서 분석에 방해되는 단어도 제거 필요하다.(예: '영화')

```
stop_words = '영화 전 난 일 걸 뭐 줄 만 건 분 개 끝 잼 이거 번 중 듯 때 게 내 말 나 수 거 점 것'
stop_words = stop_words.split() #공백을 기준으로 나누고 나면 리스트로 반환

print(stop_words)
```

```
['영화', '전', '난', '일', '걸', '뭐', '줄', '만', '건', '분', '개', '끝', '잼', '이거', '번', '중', '듯',
'때', '게', '내', '말', '나', '수', '거', '점', '것']
```

형태소 분석

● 불용어를 제외하여 형태소 분석

- 한글 텍스트에 대해서 형태소 분석 수행
- 분석으로 추출하는 명사 중에서 불용어에 포함되지 않은 텍스트만 추출하여 저장

```
from kiwipiepy import Kiwi
from tqdm import tqdm
kiwi = Kiwi()

review_token = []
for review in tqdm(reviews):
    for token in kiwi.tokenize(review):
        if (token.tag[0] == 'N') & (token.form not in stop_words): #명사이고 형태소가 불용어가 아니라면
            review_token.append(token.form) #형태소만 append
print(review_token[:100])
```

```
['디자인', '학생', '외국', '디자이너', '그', '전통', '발전', '문화', '산업', '우리', '나라', '시절', '열정', '노라노', '전통', '저', '사람', '꿈', '감사', '폴리스', '스토리', '시리즈', '뉴', '하나', '최고', '연기', '개쩔', '생각', '몰입', '진짜', '안개', '밤하늘', '초승달', '사랑', '사람', '처음', '감동', '감동', '전쟁', '빠로', '굿', '바보', '병', '쉰', '나이', '감동', '훗날', '대사', '감정', '완벽', '이해', '고질라', '능', '오페라', '작품', '극단', '평', '도', '반전', '평점', '긴장감', '스릴', '최고', '전장', '공포', '네고시에이터', '소재', '뿐', '관련', '최고', '밀회', '생각', '상당', '수작', '일본', '년', '최고', '마음', '임팩트', '일품', '오랜만', '범죄', '스릴러', '사랑', '마디', '밤', '잠', '커징텅의', '교복', '션자이', '볼펜', '자국', '마음', '형태', '마지막', '씬', '강압', '용서', '세뇌', '적용']
```

빈도 분석

● 단어 빈도수 측정

- 단어 빈도수 측정에는 'collections' 패키지의 'Counter' 함수를 이용(내장 패키지로 별도 설치가 필요 없음)
- 'counter'를 이용하면 각 단어와 각 단어의 빈도 수를 딕셔너리로 편리하게 생성 가능

```
from collections import Counter

c = Counter(review_token)
c
```

```
Output exceeds the size limit. Open the full output data in a text editor
Counter({'디자인': 53,
         '학생': 189,
         '외국': 202,
         '디자이너': 19,
...
'이상': 2643,
         ...})
```

빈도 분석

```
top_c = c.most_common(10) #빈도수 상위 10개 추출

print(top_c)
```

```
[('연기', 9820), ('최고', 8851), ('평점', 8502), ('생각', 7414), ('스토리', 7180), ('드라마', 6961), ('사람', 6753), ('감동', 6509), ('배우', 6106), ('내용', 5735)]
```

```
top_c = dict(top_c) #딕셔너리로 변환

print(top_c)
```

```
{'연기': 9820, '최고': 8851, '평점': 8502, '생각': 7414, '스토리': 7180, '드라마': 6961, '사람': 6753, '감동': 6509, '배우': 6106, '내용': 5735}
```

빈도 분석

● 단어 빈도 시각화(그래프)

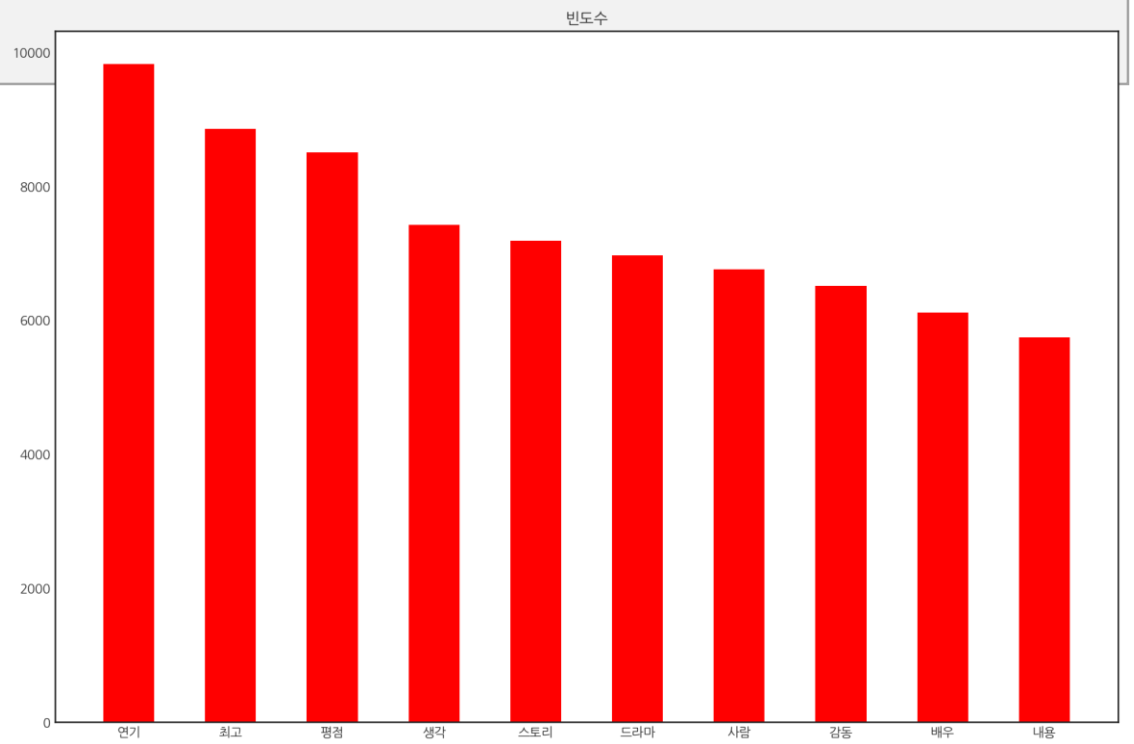
▪ 한글 폰트 설정

```
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import font_manager as fm

plt.style.use('default') #그래프 스타일 설정

%plt.rc('font', family='Gulim', size=7)  #폰트 설정
```

빈도 분석

● 세로 막대 그래프 그리기

```
plt.figure(figsize=(15,10))
plt.bar(top_c.keys(), top_c.values(), color = 'red', width = 0.5) #x축(top_c.keys()), y축(top_c.values())
plt.title('빈도수')

plt.show();
```



빈도 분석

● 단어 빈도 시각화(워드클라우드; WordCloud)

- 텍스트에 담겨있는 여러 형태소들의 등장 빈도를 가장 직관적으로 시각화하는 방법
- 텍스트에 등장하는 단어를 그 등장 빈도에 따라 서로 크기가 다르게 구름 형태로 표현함으로써, 단어의 빈도 수를 한번에 알 수 있음
- 최근에 많은 서비스들이 어떤 핵심어가 많이 등장했는가를 워드클라우드 형식으로 시각화
- 빈도 수만을 시각적으로 표현한 것이기 때문에, 단어들 사이의 연관성이나 의미 구조 등을 분석하는 데는 한계
- 파이썬에서 워드 클라우드를 시각화하기 위해 'matplotlib'와 'WordCloud'를 사용

```
from wordcloud import WordCloud
```

빈도 분석

- ‘WordCloud’로 객체를 생성, ‘generate_from_frequencies()’로 빈도 수에 따라 워드클라우드 생성

```
wc = WordCloud(font_path = ‘gulim’,  width=800, height=400, scale=2.0, max_font_size=250,
max_words=200, background_color=‘white’)

gen = wc.generate_from_frequencies(c) #워드클라우드 생성

plt.figure(figsize = (10, 10))

plt.axis(‘off’)
plt.imshow(gen);
```

4. 텍스트 벡터화

- 원-핫 인코딩
- TF-IDF
- 단어 임베딩

텍스트 벡터화(Text_Vectorization)

● 텍스트 벡터화 이해

- **자연어 처리에서는 기계가 문자를 이해 할 수 있도록 수치화해주는 과정이 반드시 필요**
- **텍스트 벡터화의 대표적인 방법**
  - 원-핫 인코딩(One-hot encoding)
  - TF-IDF(빈도수 기반 텍스트/문서 벡터화)
  - 단어 임베딩(Word Embedding)

원-핫 인코딩(One-hot encoding)

● 원-핫 인코딩(One-hot encoding)

- **인코딩 : 컴퓨터는 텍스트를 직접 처리하는게 아니라 숫자로 변환하여 처리**
  - 인코딩에는 텍스트를 정수로 변환하는 '정수 인코딩'과 원핫 벡터로 표현하는 '원핫 인코딩'이 있음
- **정수 인코딩(Integer Encoding) : 자연어를 컴퓨터가 이해할 수 있는 숫자(정수) 형태로 인코딩하는 과정**
- **원-핫 인코딩 : 정수로 표현되었지만 실제로는 문자인 데이터를 기계가 인식할 수 있도록 버꿔주는 방법**
  - N개의 단어를 각각 N차원의 벡터로 표현하는 방식
  - 단어에 해당되는 차원(인덱스)에 1을 넣고 나머지에는 0을 입력
  - 원-핫 인코딩은 단어 또는 문자를 기준으로 벡터화가 가능

원-핫 인코딩(One-hot encoding)

- **정수 인코딩과 원-핫 인코딩(하나의 문장)**

"나는 자연어 처리를 배운다"

정수 인코딩

✓ 텍스트에 단어가 100개가 있다면
1부터 100까지 차례로 단어와 정수를 맵핑
✓ 일반적으로 빈도가 높은 단어부터 번호 부여

{'나':0, '는':1, '자연어':2, '처리':3, '를':4, '배운다':5}

원핫 인코딩

✓ 단어 집합의 크기를 벡터의 차원으로 하고,
표현하고 싶은 단어의 인덱스에 1의 값을 부여
✓ 다른 인덱스에는 0을 부여함

나	1	0	0	0	0	0
는	0	1	0	0	0	0
자연어	0	0	1	0	0	0
처리	0	0	0	1	0	0
를	0	0	0	0	1	0
배운다	0	0	0	0	0	1

원-핫 인코딩(One-hot encoding)

● 정수 인코딩

```
tokens = ['나', '는', '자연어', '처리', '를', '배운다']
```

▪ 형태소별 인덱스로 번호 부여

```
word_to_index = {word : index for index, word in enumerate(tokens)}
print('정수 인코딩 :', word_to_index)
```

```
정수 인코딩 : {'나': 0, '는': 1, '자연어': 2, '처리': 3, '를': 4, '배운다': 5}
```

원-핫 인코딩(One-hot encoding)

● 원-핫 인코딩

▪ 원-핫 벡터 생성 함수 정의

```
def one_hot_encoding(word, word_to_index):
    one_hot_vector = [0]*(len(word_to_index))
    index = word_to_index[word]
    one_hot_vector[index] = 1
    return one_hot_vector
```

▪ 원-핫 인코딩 : 형태소별 원-핫 벡터 출력

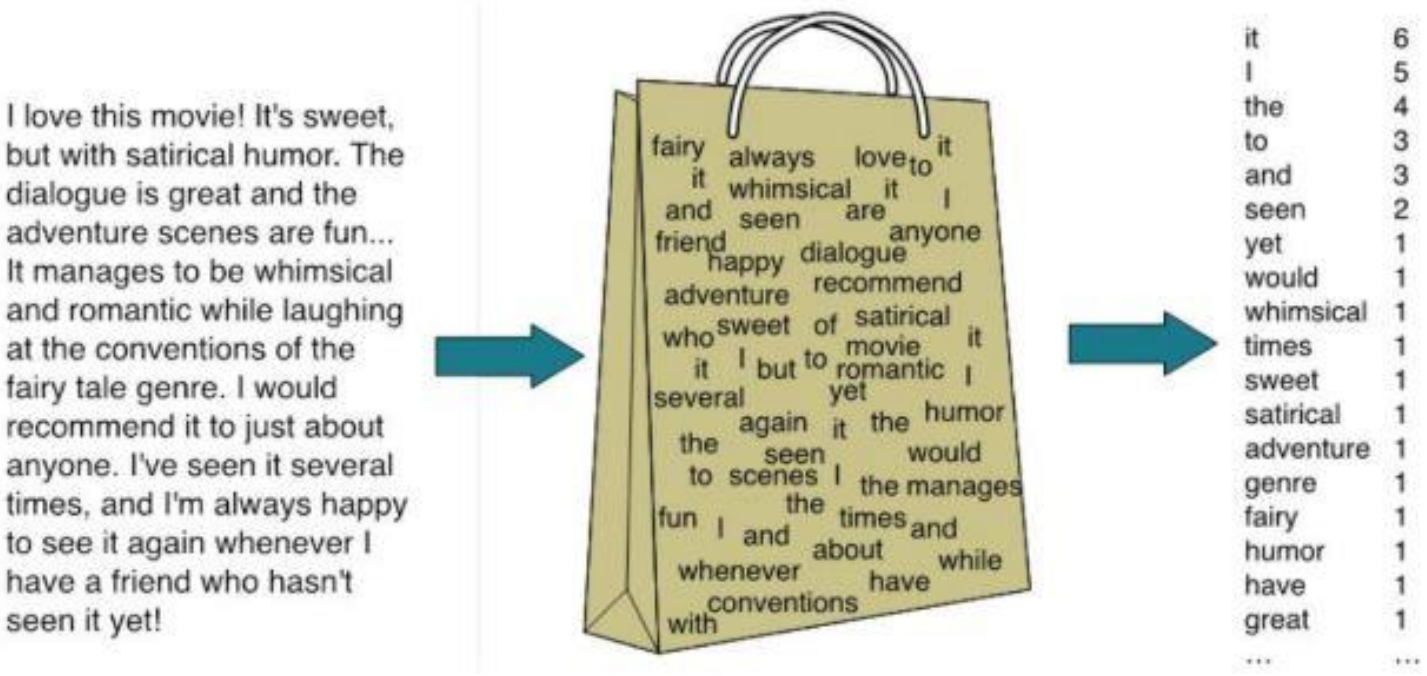
```
for word in tokens:
    print(word, '\t', one_hot_encoding(word, word_to_index))
```

```
나          [1, 0, 0, 0, 0, 0]
는          [0, 1, 0, 0, 0, 0]
자연어      [0, 0, 1, 0, 0, 0]
처리        [0, 0, 0, 1, 0, 0]
를          [0, 0, 0, 0, 1, 0]
배운다      [0, 0, 0, 0, 0, 1]
```

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

● BoW(Bag of Words)

- **BoW : 등장하는 단어들의 숫자를 세서 단어 주머니에 넣어 두는 것**



[출처] https://web.stanford.edu/~jurafsky/slp3/4.pdf

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- **BoW를 만드는 과정**
  - 각 단어에 고유한 인덱스를 부여
  - 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터를 생성
- **데이터 입력**

```
corpus = ['If you do not walk today. you will have to run tomorrow']
```

- **빈도 측정**

```
from sklearn.feature_extraction.text import CountVectorizer
vector = CountVectorizer()
bow = vector.fit_transform(corpus)          #각 단어의 빈도수 측정 -> bow 생성
print(vector.get_feature_names_out()) #토큰화된 단어 목록 확인(알파벳순)
print(bow.toarray())      #배열 형태로 빈도수 출력(알파벳순)
print(vector.vocabulary_)          #벡터에 포함되어 있는 vocabulary 인덱스 값
```

```
['do' 'have' 'if' 'not' 'run' 'to' 'today' 'tomorrow' 'walk' 'will' 'you']
[[1 1 1 1 1 1 1 1 1 1 2]]
{'if': 2, 'you': 10, 'do': 0, 'not': 3, 'walk': 8, 'today': 6, 'will': 9, 'have': 1, 'to': 5, 'run': 4, 'tomorrow': 7}
```

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- **불용어 제거**

```
vector = CountVectorizer(stop_words='english')     #영어 불용어 객체 생성
```

- **BoW**

```
bow = vector.fit_transform(corpus)          #각 단어의 빈도수 측정 -> bow 생성

print(vector.get_feature_names_out()) # 토큰화된 단어 목록 확인(알파벳순)
print(bow.toarray())      #배열 형태로 빈도수 출력 (알파벳순)
print(vector.vocabulary_)          #벡터에 포함되어 있는 vocabulary 인덱스 값
```

```
['run' 'today' 'tomorrow' 'walk']
[[1 1 1 1]]
{'walk': 3, 'today': 1, 'run': 0, 'tomorrow': 2}
```

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

● 문서 단어 행렬(Document-Term Matrix, DTM)

- **다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것**
- **각 문서에 대한 BoW를 하나의 행렬로 표현한 것**

- 아래 4개의 문서를 띄어쓰기 단위 토큰화를 수행한다고 가정하고, 문서 단어 행렬로 표현

문서1 : 사고 싶은 스마트폰
문서2 : 사고 싶은 스마트워치
문서3 : 성능 좋은 스마트폰 스마트폰
문서4 : 나는 스마트폰이 좋아요

- 문장을 공백 단위로 토큰화 후, 단어 사전 구성
- 각 문서에 포함된 토큰의 빈도수 기록

※ 문서 단어 행렬의 한계
- 원-핫 벡터 표현으로 공간적 낭비와 연산 리소스 증가
- 단순 빈도수 기반 가중치를 부여하는 것은 중요하지 않은 단어에 높은 가중치를 부여할 수도 있음
  ex) “the” 는 모든 문서에 자주 등장하는데 DTM에서는 높은 가중치 부여

	나는	사고	성능	스마트워치	스마트폰	스마트폰이	싶은	좋아요	좋은
문서1	0	1	0	0	1	0	1	0	0
문서2	0	1	0	1	0	0	1	0	0
문서3	0	0	1	0	2	0	0	0	1
문서4	1	0	0	0	0	1	0	1	0

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- **데이터 입력**

```
docs = ['사고 싶은 스마트폰', '사고 싶은 스마트워치', '성능 좋은 스마트폰 스마트폰', '나는 스마트폰이 좋아요']
```

- **한국어 BoW**

```
from sklearn.feature_extraction.text import CountVectorizer

vector = CountVectorizer()
bow = vector.fit_transform(docs)

print(bow.toarray())
print(vector.vocabulary_)
```

```
[[0 1 0 0 1 0 1 0 0]
 [0 1 0 1 0 0 1 0 0]
 [0 0 1 0 2 0 0 0 1]
 [1 0 0 0 0 1 0 1 0]]
{'사고': 1, '싶은': 6, '스마트폰': 4, '스마트워치': 3, '성능': 2, '좋은': 8, '나는': 0, '스마트폰이': 5, '좋아요': 7}
```

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- **DTM**

```
import pandas as pd

columns = []
for k, v in sorted(vector.vocabulary_.items(), key=lambda item:item[1]):
      columns.append(k)

df = pd.DataFrame(bow.toarray(), columns=columns)
df
```

	나는	사고	성능	스마트워치	스마트폰	스마트폰이	싶은	좋아요	좋은
0	0	1	0	0	1	0	1	0	0
1	0	1	0	1	0	0	1	0	0
2	0	0	1	0	2	0	0	0	1
3	1	0	0	0	0	1	0	1	0

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

● 어휘 빈도-문서 역빈도(Term Frequency-Inverse Docunment Frequency, TF-IDF) 분석

- **단어의 빈도와 역 문서 빈도를 사용하여 DTM 내의 각 단어들 마다 중요한 정도를 가중치로 부여하는 것**
  - 단순히 빈도수가 높은 단어가 핵심어가 아닌, 특정 문서에서만 집중적으로 등장할 때 해당 단어가 문서의 주제를 잘 담고 있는 핵심어라고 가정
  - 특정 문서에서 특정단어가 많이 등장하고 그 단어가 다른 문서에서 적게 등장할 때, 그 단어를 특정 문서의 핵심어로 간주
    → 특정 문서에서 특정 단어가 많이 등장하는 것을 의미
  - 문서의 유사도 측정, 검색 시스템에서 검색 결과의 중요도 계산, 문서 내 특정 단어의 중요도 계산에 활용
  - 어휘 빈도-문서 역빈도(tf-idf) : 어휘 빈도(tf)와 역문서 빈도(idf)를 곱해서 계산 $tf_{x,y}$
  - 역문서 빈도(idf) : 다른 문서에서 등장하지 않는 단어 빈도를 의미 $\log(N/df_x)$
  - 어휘 빈도-문서 역빈도(tf-idf) : 다음과 같이 표현 $W_{x,y} = tf_{x,y} * \log(N/df_x)$

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- **tf-idf 계산 방법**

TF – IDF = tf × idf

$$\text{- idf} = \log\left(\frac{n}{1+df(t)}\right)$$

- tf : 각 문서에 포함된 각 단어의 등장 빈도 (dtm 값)
- idf: 특정 단어 t가 등장한 문서의 수의 역수, 한 문서내 동일 단어가 많을 수록 값은 작아짐
- n : 문서의 총 개수 , df(t) : 특정 단어 t가 등장한 문서의 수

① tf 계산

- 문장을 공백 단위로 토큰화 후, 단어 사전을 구성하고, 각 문서에 포함된 토큰의 빈도수 기록 (dtm과 동일)

문서1 : 사고 싶은 스마트폰
문서2 : 사고 싶은 스마트워치
문서3 : 성능 좋은 스마트폰 스마트폰
문서4 : 나는 스마트폰이 좋아요

	나는	사고	성능	스마트워치	스마트폰	스마트폰이	싶은	좋아요	좋은
문서1	0	1	0	0	1	0	1	0	0
문서2	0	1	0	1	0	0	1	0	0
문서3	0	0	1	0	2	0	0	0	1
문서4	1	0	0	0	0	1	0	1	0

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- tf-idf 계산 방법

② idf 계산

- 각 단어마다 log(n/(1+df(t))) 계산 (문서 수 t : 4, df(t) : 특정단어 t가 등장한 문서의 수)

나는(1회)	사고(2회)	성능(1회)	스마트워치(1회)	스마트폰(2회)	스마트폰이(1회)	싶은(2회)	좋아요(1회)	좋은(1회)
log(4/(1+1)) = 0.693147	log(4/(1+2)) = 0.287682	log(4/(1+1)) = 0.693147	log(4/(1+1)) = 0.693147	log(4/(1+2)) = 0.287682	log(4/(1+1)) = 0.693147	log(4/(1+2)) = 0.287682	log(4/(1+1)) = 0.693147	log(4/(1+1)) = 0.693147

② idf 계산

- ①에서 구한 tf에 ②에서 구한 idf를 곱하여 tf-idf 계산
- 단어의 중요도에 따라 tf-idf 값이 커짐

	나는	사고	성능	스마트워치	스마트폰	스마트폰이	싶은	좋아요	좋은
문서1	0	0.287682	0	0	0.287682	0	0.287682	0	0
문서2	0	0.287682	0	0.693147	0	0	0.287682	0	0
문서3	0	0	0.693147	0	0.575364	0	0	0	0.693147
문서4	0.693147	0	0	0	0	0.693147	0	0.693147	0

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- tf-idf를 편리하게 계산하기 위해 'scikit-learn'의 'tfidfvectorizer'를 이용
- 앞서 계산한 단어 빈도 수를 입력하여 tf-idf로 변환
- tfidf 객체 생성

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer().fit(docs)

print(tfidf.transform(docs).toarray())
print(tfidf.vocabulary_)
```

```
[[0.         0.57735027 0.         0.         0.57735027 0.  0.57735027 0.         0.        ]
 [0.         0.52640543 0.         0.66767854 0.         0.  0.52640543 0.         0.        ]
 [0.         0.         0.47212003 0.         0.7444497  0.  0.         0.         0.47212003]
 [0.57735027 0.         0.         0.         0.         0.57735027     0.         0.57735027
  0.         ]]
{'사고': 1, '싶은': 6, '스마트폰': 4, '스마트워치': 3, '성능': 2, '좋은': 8, '나는': 0, '스마트폰이': 5, '좋아요': 7}
```

TF-IDF(빈도수 기반 텍스트/문서 벡터화)

- **시인성이 좋게 데이터프레임으로 변환**

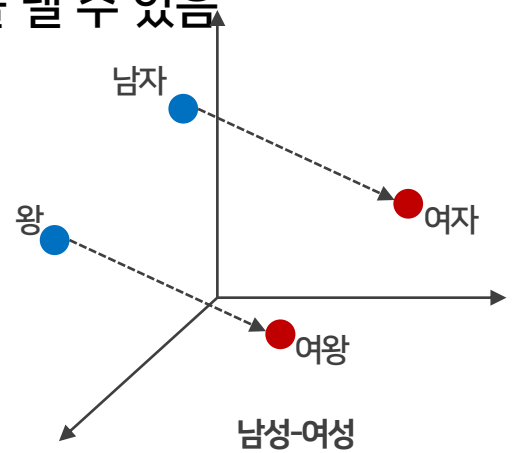
```
import pandas as pd

columns = []
for k, v in sorted(tfidf.vocabulary_.items(), key=lambda item:item[1]):
      columns.append(k)

df = pd.DataFrame(tfidf.transform(docs).toarray(), columns=columns)
df
```

	나는	사고	성능	스마트워치	스마트폰	스마트폰이	싶은	좋아요	좋은
0	0.00000	0.577350	0.00000	0.000000	0.57735	0.00000	0.577350	0.00000	0.00000
1	0.00000	0.526405	0.00000	0.667679	0.00000	0.00000	0.526405	0.00000	0.00000
2	0.00000	0.000000	0.47212	0.000000	0.74445	0.00000	0.000000	0.00000	0.47212
3	0.57735	0.000000	0.00000	0.000000	0.00000	0.57735	0.000000	0.57735	0.00000

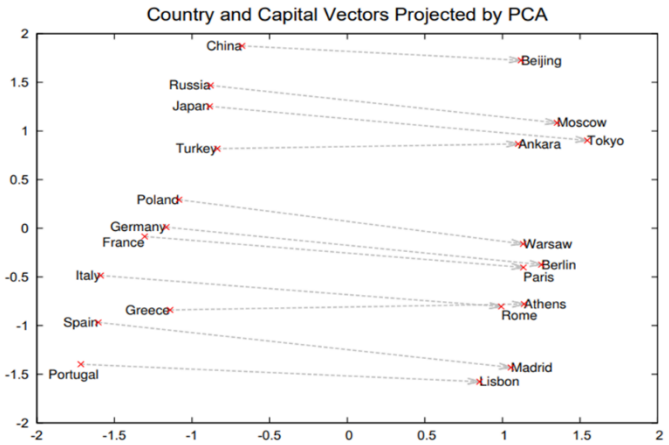
단어 임베딩(Word Embedding)

● 단어 임베딩(Word Embedding)

- **의미를 포함하는 단어를 벡터로 바꾸는 기법, 비슷한 분포를 가진 단어의 주변 단어들도 비슷한 의미를 가진다는 것을 가정**
  - 왼쪽 그림을 보면 왕과 여왕, 여왕과 여자가 같은 방향에 있음
  - 의미가 비슷한 단어는 비슷한 방향에 위치
  - 단어 임베딩은 단어의 의미를 효과적으로 표현하기 때문에 one-hot encoding보다 학습 성능을 높일 수 있음
  - 대량 데이터로 단어 임베딩을 미리 학습시키면, 문서분류 같은 과제에서 더 적은 데이터로 학습된 임베딩을 사용하여 높은 성능을 낼 수 있음



남성-여성



[출처] http://doc.mindscale.kr/km/unstructured/11.html

단어 임베딩(Word Embedding)

- **Word Embedding**
  - 희소표현 : one-hot encoding은 단어의 의미를 전혀 고려하지 않으며 벡터의 길이가 총 단어 수가 되므로 매우 희박(sparse)한 형태가 됨
  - 밀집표현 : 이를 해결하기 위해 단어의 의미를 고려하여 좀 더 조밀한 차원에 단어를 벡터로 표현하는 것을 단어 임베딩(word embedding)이라 함
  - 원핫 벡터로 표현된 단어를 밀집 벡터(dense vector)로 변환하는것을 워드 임베딩(word embedding)이라고 함
    - 이렇게 만들어진 단어 벡터는 단어의 의미 담고있으며, 단어 벡터 간의 연산도 가능
    - 많은 양의 문서를 학습하여 얻어진 단어 벡터는 단어 간의 관계를 보다 정확하게 나타냄

단어 임베딩(Word Embedding)

▪ Word Embedding

- 10,000개의 단어로 이루어진 단어사전에서 희소표현과 밀집표현 비교

원핫벡터(희소표현) : 영화 = [0 0 0 0 0 0 1 0 0 0 0 ----] 단어수가 10,000개인 경우, 하나의 단어 표현에 10,000개의 데이터 필요
밀집표현 : 영화 = [0.1 21.3 0.32 46.7 12.1, --] 64차원의 임베딩 벡터인 경우, 하나의 단어 표현에 64개의 데이터 필요

구분	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

- 워드 임베딩 방법으로는 LSA, **Word2Vec, FastText**, Glove 등이 있음

단어 임베딩(Word Embedding)

- **Word2Vec**
  - 단어를 벡터로 임베딩하는 방식은 머신러닝을 통해 학습되는데, 신경망을 기반으로 한 단어 벡터화의 대표적인 방법
  - 단어 벡터 간 유의미한 유사도를 반영할 수 있도록 단어의 의미를 수치화 하는 방법(문맥기반 학습)
- **Word2Vec 학습 방법**
  - CBOW(continuous bag of words)와 Skip-gram(SG)의 두 가지 알고리즘이 있고 일반적임

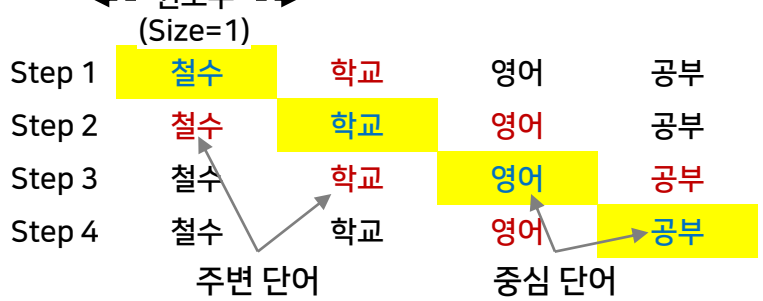
"철수는 학교에서 (영어)를 공부한다."

주변 단어(context word) 중심 단어 (center word)

CBOW

✓ **주변에 있는 단어들로 중심에 있는 단어들을 예측**

- 윈도우 단위로 중심 단어와 주변 단어를 구분해서 반복 학습



Skip-gram

✓ **중심에 있는 단어들로 주변 단어들을 예측**

- 윈도우 단위로 중심 단어와 주변 단어를 구분해서 반복 학습

※ 윈도우 사이즈 1 적용

단어	중심단어	주변단어
철수, 학교, 영어, 공부	학교	철수
	학교	영어
철수, 학교, 영어, 공부	영어	학교
	영어	공부

단어 임베딩(Word Embedding)

- **Word2Vec로 임베딩**

  - 형태소로 분석된 명사를 파일로 읽기

```
f = open('data_set/risk_noun.txt', 'r', encoding='utf-8')
raw = f.read()
print(raw[:1000])
```

```
[['전략', '리스크', '관리', '방법론'], ['전략', '리스크', '관리', '이해', '개요', '리스크', '수익', '관리', '전략', '개요',
'전략', '리스크', '관리', '방법론', '개요', '도입', '방안'], ['전략', '리스크', '관리', '이해', '비즈니스', '리스크', '소유',
'범주', '모형', '비즈니스', '리스크', '소유', '이사회', '궁극', '리스크', '감독', '책임', '최상', '의사', '결정', '책임', '영
진', '이사회', '감사', '위원회', '리스크', '관리', '내부감사', '리스크', '범주', '법무', '담당', '주주', '소비자', '공급', '파
트너', '경쟁자', '시장', '신용', '재무', '구조', '보고', '프로세스', '인적', '자원', '설비', '지배', '구조', '경영', '계획',
'대외', '협력', '재무', '재무', '보고', '대한', '내부통제', '노동', '외국', '노동자', '환경', '배기', '가스', '오염', '물질',
'정책', '입법', '로비', '지적', '소유권', '의사결정', '지원', '정보', '기술'], ['전략', '리스크', '관리', '모델', '외부', '환
경', '이해', '관계자', '프로세스', '리스크', '모니터', '리스크', '구체화', '리스크', '최적화', '전략', '리스크', '관리', '이
해'], ['리스크', '관리', '전략', '리스크', '관리', 'frozenset', '리스크', '최적화', '리스크', '구체화', '리스크', '판단', '리스
크', '평가', '우선', '순위', '결정', '유형', '분류', '확률', '추정', '영향', '모형', '계량', '유형', '대응', '방안', '수립',
'회피', '수용', '정비', '완화', '분석', '상관', '실행', '모니터링', '세련', '리드', '리스크', '평가', '리스크', '대응', ...]]
```

단어 임베딩(Word Embedding)

- Word2Vec로 임베딩

  - 문자열을 리스트로 변환

```
risk_noun = eval(raw)    #스트링을 리스트로 변환
print(risk_noun[:5])
```

```
[['전략', '리스크', '관리', '방법론'], ['전략', '리스크', '관리', '이해', '개요', '리스크', '수익', '관리', '전략', '개요',
'전략', '리스크', '관리', '방법론', '개요', '도입', '방안'], ['전략', '리스크', '관리', '이해', '비즈니스', '리스크', '소유',
'범주', '모형', '비즈니스', '리스크', '소유', '이사회', '궁극', '리스크', '감독', '책임', '최상', '의사', '결정', '책임', '영
진', '이사회', '감사', '위원회', '리스크', '관리', '내부감사', '리스크', '범주', '법무', '담당', '주주', '소비자', '공급', '파
트너', '경쟁자', '시장', '신용', '재무', '구조', '보고', '프로세스', '인적', '자원', '설비', '지배', '구조', '경영', '계획',
'대외', '협력', '재무', '재무', '보고', '대한', '내부통제', '노동', '외국', '노동자', '환경', '배기', '가스', '오염', '물질',
'정책', '입법', '로비', '지적', '소유권', '의사결정', '지원', '정보', '기술'], ['전략', '리스크', '관리', '모델', '외부', '환
경', '이해', '관계자', '프로세스', '리스크', '모니터', '리스크', '구체화', '리스크', '최적화', '전략', '리스크', '관리', '이
해'], ['리스크', '관리', '전략', '리스크', '관리', 'frozenset', '리스크', '최적화', '리스크', '구체화', '리스크', '판단', '리스
크', '평가', '우선', '순위', '결정', '유형', '분류', '확률', '추정', '영향', '모형', '계량', '유형', '대응', '방안', '수립',
'회피', '수용', '정비', '완화', '분석', '상관', '실행', '모니터링', '세련', '리드', '리스크', '평가', '리스크', '대응', '전략',
'리스크', '관리', '이해']]
```

단어 임베딩(Word Embedding)

- **Word2Vec 훈련시키기 : Word2Vec 모델 활용**

```
from gensim.models import Word2Vec

model = Word2Vec(risk_noun, window=2, min_count=1, workers=4, sg=0)
```

- **유사도가 높은 단어 추출**

```
value = model.wv.most_similar('리스크', topn=5)
print(value)
```

```
[('위험', 0.9860552549362183), ('관리', 0.9852123260498047), ('기업', 0.9813206791877747), ('위험관리',
0.9809973239898682), ('평가', 0.976618230342865)]
```

단어 임베딩(Word Embedding)

- **저장된 학습모델로 유사도 높은 단어 추출**

```
model.wv.save_word2vec_format('word2')
from gensim.models import Word2Vec
from gensim.models import KeyedVectors #학습된 모델을 불러오기 위한 패키지 로드

load_model = KeyedVectors.load_word2vec_format('word2') #학습된 모델 불러오기
load_model.most_similar('마케팅', topn=5) #마케팅과 유사도 높은 단어 추출
```

```
[('이해', 0.5496184825897217), ('의미', 0.5442448854446411), ('활동', 0.5360015034675598), ('공급',
0.5355146527290344), ('개별', 0.5308395624160767)]
```

5. 군집 분석

- 텍스트 유사도
- 군집화를 위한 데이터 전처리
- Word2Vec 생성
- Scikit-learn을 이용한 군집화

군집 분석(Cluster Analysis)

● 군집 분석 이해

- 군집 분석은 데이터의 특성에 따라 유사한 것끼리 묶은 것
- 유사성을 기반으로 군집을 분류하고, 군집에 따라 유형별 특징을 분석하는 기법
- 텍스트에 대한 군집 분석에서는 군집으로 묶여진 텍스트들끼리는 최대한 유사하고, 다른 군집으로 묶여진 텍스트들과는 최대한 유사하지 않도록 분류하는 것

텍스트 유사도(Text Similarity)

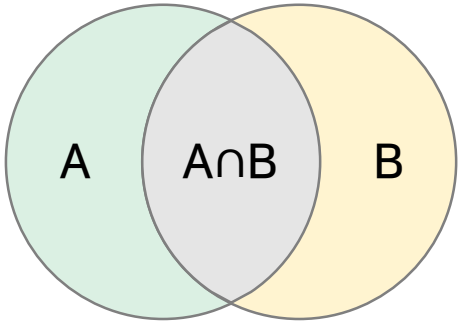
● 텍스트 유사도 이해

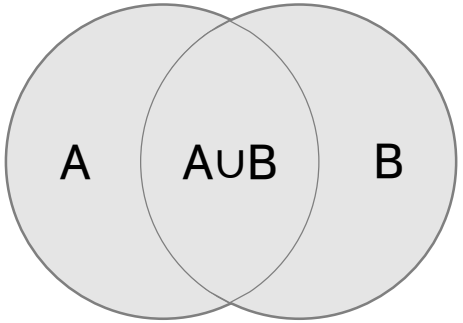
- 텍스트 유사도는 대표적으로 텍스트 쌍에 대한 자카드 유사도와 코사인 유사도로 계산
- 자카드 유사도(Jaccard Similarity): 두 텍스트 문서 사이에 공통된 용어의 수와 해당 텍스트에 존재하는 총 고유 용어 수의 비율을 사용
- 코사인 유사도(Cosine Similarity): 두 벡터 간의 코사인 각도를 이용하여 구할 수 있는 유사도 계산 방식

텍스트 유사도(Text Similarity)

● 자카드 유사도

- **A, B 두개의 집합이 있다고 할 때, 합집합에서 교집합의 비율을 구함으로써 유사도를 계산 방식**





$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

[출처] https://blog.naver.com/sw4r/222223674842

텍스트 유사도(Text Similarity)

- **데이터 입력**

```
d1 = “The sky is blue”
d2 = “The sun is bright”
d3 = “The sun in the sky is bright”
```

- **자카드 유사도 함수 생성**

```
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')

def jaccard_similarity(d1, d2):
    lemmatizer = WordNetLemmatizer()
    words1 = [lemmatizer.lemmatize(word.lower()) for word in word_tokenize(d1)]
    words2 = [lemmatizer.lemmatize(word.lower()) for word in word_tokenize(d2)]
    inter = len(set(words1).intersection(set(words2)))
    union = len(set(words1).union(set(words2)))

    return inter/union
```

텍스트 유사도(Text Similarity)

- **자카드 유사도 계산**

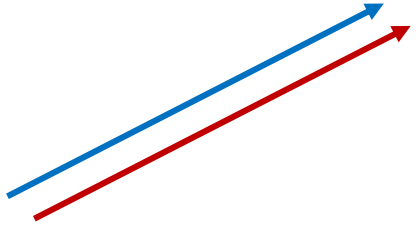
```
print(jaccard_similarity(d1, d2))
print(jaccard_similarity(d1, d3))
print(jaccard_similarity(d2, d3))
```

```
0.3333333333333333
0.42857142857142855
0.6666666666666666
```

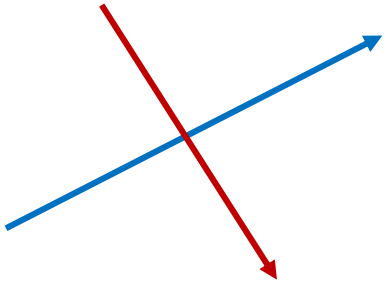
텍스트 유사도(Text Similarity)

● 코사인 유사도

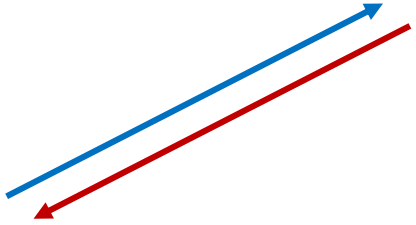
- 두 벡터의 방향이 완전히 동일한 경우에는 1의 값을 가지며, 90°의 각일때 0, 반대방향이면 -1의 값을 가짐
- 코사인 유사도는 -1 이상 1 이하의 값을 가지며 값이 1에 가까울수록 유사도가 높다고 판단



코사인 유사도 1



코사인 유사도 0



코사인 유사도 -1

- 두 벡터 A, B에 대한 코사인 유사도 식, similarity = $\cos\theta = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i \ X \ B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \ X \sqrt{\sum_{i=1}^{n}(B_i)^2}}$

- 유사도 측정 방법으로 유클리드, 자카드 방법등이 있으나 코사인 유사도 측정이 가장 많이 사용됨

텍스트 유사도(Text Similarity)

- **tf-idf vectorizer 객체 생성**

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

import numpy as np

tfidf_vectorizer = TfidfVectorizer()
```

- **tf-idf 계산**

```
docs = np.array([d1,d2,d3])

tfidf_matrix = tfidf_vectorizer.fit_transform(docs)
print(tfidf_matrix.toarray())
```

```
[[0.66283998 0.         0.         0.39148397 0.50410689 0.         0.39148397]
 [0.         0.55847784 0.         0.43370786 0.         0.55847784 0.43370786]
 [0.         0.35934131 0.47249064 0.27906059 0.35934131 0.35934131 0.55812117]]
```

텍스트 유사도(Text Similarity)

- **코사인 유사도 계산**

  - d1과 d1,d2,d3간 유사도

```
similarity = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix)
print(similarity)
```

```
[[1.         0.33957935 0.50888967]]
```

  - 각 문서간 유사도

```
similarity = cosine_similarity(tfidf_matrix, tfidf_matrix)
print(similarity)
```

```
[[1.         0.33957935 0.50888967]
 [0.33957935 1.         0.76446063]
 [0.50888967 0.76446063 1.        ]]
```

텍스트 유사도(Text Similarity)

- 코사인 유사도 계산

  - 다른 문서와 d1,d2,d3간 유사도

```
d5 = ‘this is test’
test = tfidf_vectorizer.transform([d5])
similarity = cosine_similarity(test, tfidf_matrix)
print(similarity)
```

```
[[0.39148397 0.43370786 0.27906059]]
```

텍스트 유사도(Text Similarity)

- 코사인 유사도 계산

  - 시각화

```
import matplotlib.pyplot as plt
import seaborn as sns

similarity = cosine_similarity(tfidf_matrix, tfidf_matrix)
print(similarity)

map = sns.heatmap(similarity, xticklabels=docs, yticklabels=docs, cmap='viridis')
map.figure.set_size_inches(10, 7)
plt.show();
```

```
[[1.         0.33957935 0.50888967]
 [0.33957935 1.         0.76446063]
 [0.50888967 0.76446063 1.        ]]
```

군집화를 위한 데이터 전처리

● 데이터셋 파일 읽기

- **네이버 영화 리뷰 데이터**
  - [데이터 출처] https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt
- **리뷰 데이터 읽기**

```
f = open('data_set/ratings.txt', 'r', encoding='utf-8')
raw = f.readlines()
print(raw[:3])
```

```
['id\tdocument\tlabel\n', '8112052\t어릴때보고 지금다시봐도 재밌어요ㅋㅋ\t1\n', '8132799\t디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업이 부러웠는데. 사실 우리나라에서도 그 어려운시절에 끝까지 열정을 지킨 노라노 같은 전통이있어 저와 같은 사람들이 꿈을 꾸고 이뤄나갈 수 있다는 것에 감사합니다.\t1\n']
```

군집화를 위한 데이터 전처리

● 데이터 전처리

▪ 리뷰 데이터만 추출

```
reviews = []
for i in raw:
    reviews.append(i.split('\t')[1])

print(reviews[:5])
```

```
['document', '어릴때보고 지금다시봐도 재밌어요ㅋㅋ', '디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업이 부러웠는데. 사실 우리나라에서도 그 어려운시절에 끝까지 열정을 지킨 노라노 같은 전통이있어 저와 같은 사람들이 꿈을 꾸고 이뤄나갈 수 있다는 것에 감사합니다.', '폴리스스토리 시리즈는 1부터 뉴까지 버릴께 하나도 없음.. 최고.', '와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이런게 진짜 영화지']
```

군집화를 위한 데이터 전처리

- **'document' 제외**

```
reviews = reviews[1:]
print(reviews[:5])
```

```
['어릴때보고 지금다시봐도 재밌어요ㅋㅋ', '디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업이 부러웠는데. 사실 우리나라에서도 그 어려운시절에 끝까지 열정을 지킨 노라노 같은 전통이있어 저와 같은 사람들이 꿈을 꾸고 이뤄나갈 수 있다는 것에 감사합니다.', '폴리스스토리 시리즈는 1부터 뉴까지 버릴께 하나도 없음.. 최고.', '와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이런게 진짜 영화지', '안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화.']
```

군집화를 위한 데이터 전처리

- **형태소 분석 : 2차원 리스트로 명사만 추출**

```
from kiwipiepy import Kiwi
from kiwipiepy.utils import Stopwords
from tqdm import tqdm

kiwi = Kiwi()

reviews_token = []

stopwords = Stopwords()

for review in tqdm(reviews):
      token = kiwi.tokenize(review, stopwords=stopwords)
      review_token = []
      for t in token:
            if t.tag[0] == 'N': review_token.append(t.form)
      reviews_token.append(review_token)
```

군집화를 위한 데이터 전처리

- **분석 결과 확인**

```
print(reviews_token[:5])
```

```
[[], ['디자인', '학생', '외국', '디자이너', '전통', '발전', '문화', '산업', '나라', '시절', '끝', '열정', '노라노', '전통', '저', '꿈', '감사'], ['폴리스', '스토리', '시리즈', '뉴', '하나', '최고'], ['연기', '개쩔', '거', '생각', '몰입', '진짜', '영화'], ['안개', '밤하늘', '초승달', '영화']]
```

Word2Vec 생성

● 'Word2Vec' 모델 생성

- 'genism'의 'Word2Vec'은 선언과 동시에 학습을 해 단어 벡터들을 생성

```
from gensim.models import Word2Vec
from sklearn.manifold import TSNE
from matplotlib import font_manager as fm
from matplotlib import rc

word2vec = Word2Vec(reviews_token, min_count=5)
word2vec
```

```
<gensim.models.word2vec.Word2Vec at 0x271146ae370>
```

Word2Vec 생성

- 영화와 유사한 단어 추출

```
sim = word2vec.wv.most_similar('영화')
print(sim)
```

```
[('독립', 0.6927603483200073), ('수작', 0.6845133900642395), ('관객', 0.6315966248512268), ('상업',
0.6264661550521851), ('포르노', 0.6221484541893005), ('공포물', 0.6059572100639343), ('개요',
0.60398930311203), ('영상물', 0.6025459170341492), ('스너프', 0.596482515335083), ('왜캐',
0.594750702381134)]
```

- t-sne을 이용한 단어 벡터 시각화

```
tsne = TSNE(n_components=2)
tsne
```

Word2Vec 생성

- **어휘간 유사도 계산**

```
vocab = word2vec.wv.vocab
similarity = word2vec.wv[vocab]
print(similarity)
```

```
[[-0.73233676  0.27043962  0.43560898 ... -0.70385784  0.5132779
   0.12687574]
 [-0.27544227  0.60628736  0.2689779  ... -0.1157082   0.3869125
  -0.21326086]
 [-1.0258583   0.6060101  -0.203811   ... -0.9116861   0.64193004
   0.50361997]
 ...
 [-0.02189205  0.02237049  0.00805911 ... -0.02866768  0.00850523
  -0.00325041]
 [-0.06540474  0.04956115  0.05262875 ... -0.04837998  0.02619991
   0.00979126]
 [-0.02347413  0.02524081  0.01641509 ... -0.03256354  0.0278239
  -0.00713375]]
```

Word2Vec 생성

- **데이터 프레임으로 변환**

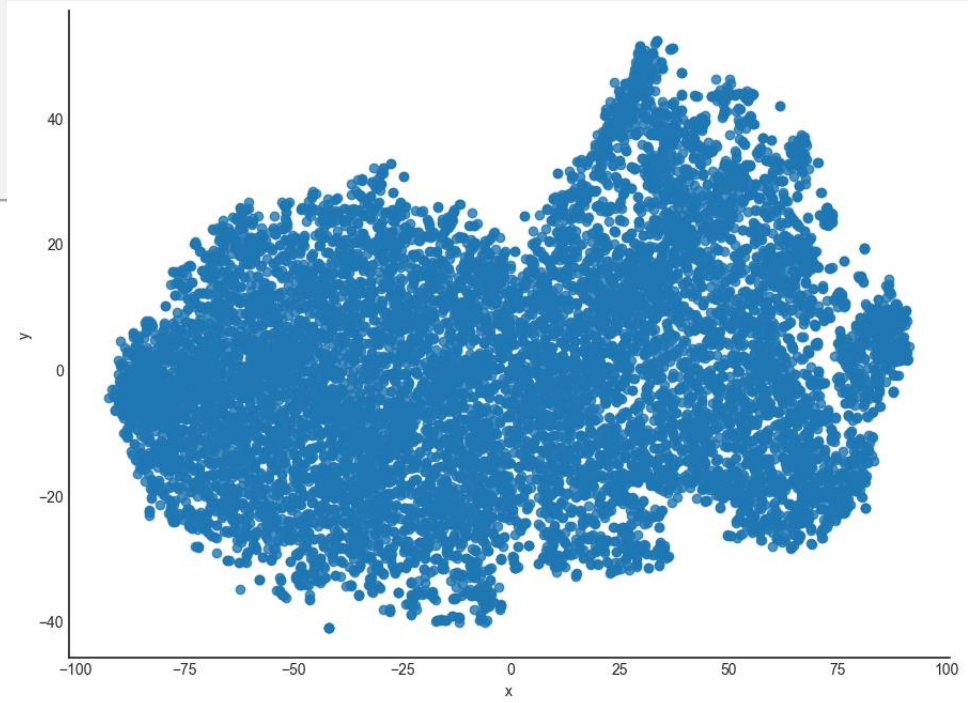
```
import pandas as pd
transform_similarity = tsne.fit_transform(similarity)
df = pd.DataFrame(transform_similarity, index=vocab, columns=['x', 'y'])
df[:10]
```

	x	y
영화	66.173157	29.204741
거	71.904938	-2.111635
점	73.436226	24.978096
연기	29.654057	51.565075
최고	59.791656	36.617825
평점	73.873566	24.674454
이거	68.221382	24.188896
생각	71.849144	-15.455261
스토리	90.166183	5.329839
드라마	53.769367	43.330643

Word2Vec 생성

- **시각화**

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.style.use('default')

map = sns.lmplot(x='x', y='y', data=df, fit_reg=False)
map.figure.set_size_inches(10, 7)
plt.show();
```



Scikit-learn을 이용한 군집화

● 계층적 군집화 이해

- 계층적 군집화란 개별 개체들을 유사한 개체나 그룹과 통합해 군집화를 수행하는 알고리즘
- 모든 개체간 거리나 유사도가 미리 계산되어 있어야만 하며, 계산복잡도도 비계층적 군집화보다 큼
- 비계층적 군집화와는 달리 군집 수를 지정하지 않아도 군집화를 할 수 있는 것이 장점

● 비계층적 군집화 이해

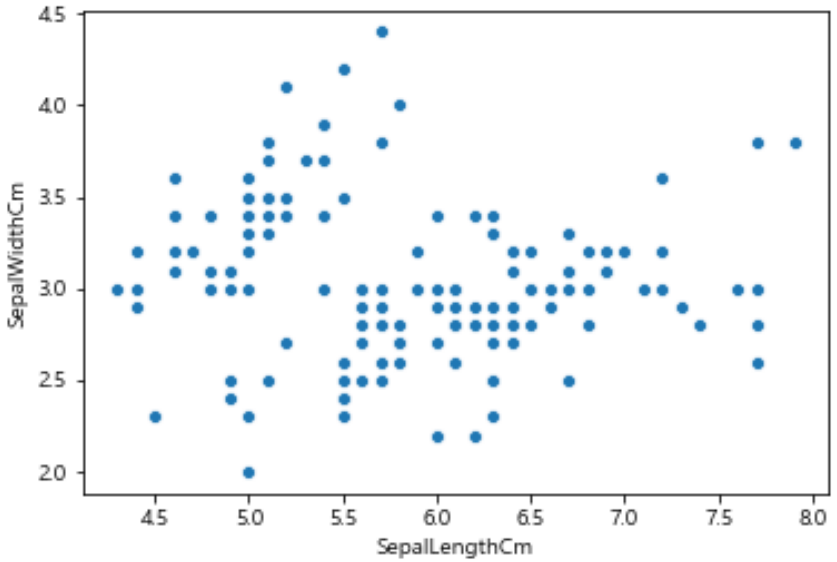
- 비계층적 군집화는 나눌 클러스터 개수를 지정해 각 개체가 어느 클러스터에 속하는 지를 결정
- 계층적 군집화보다 계산 복잡도가 작기 때문에 대량의 데이터에 유리하나, 클러스터 개수에 따라
- 군집화 성능이 크게 좌우되기 때문에 조정이 필요
- 대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

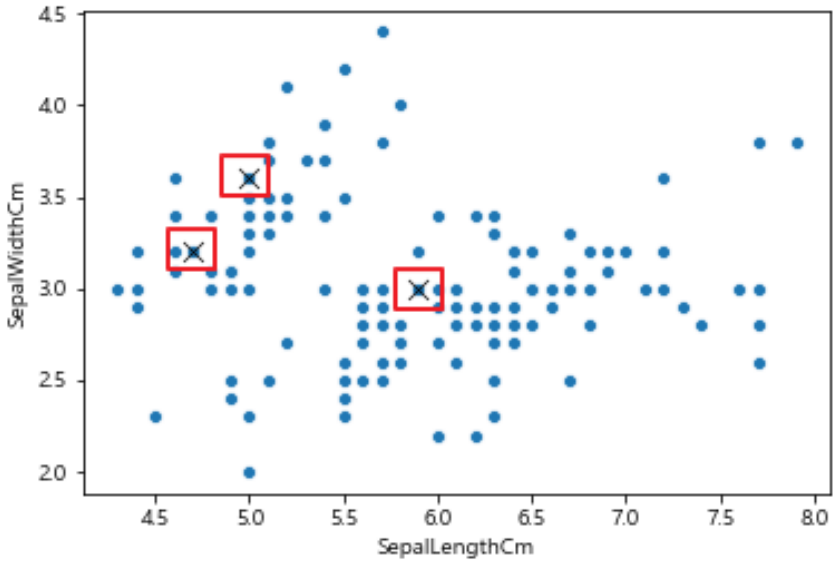
● K-Means 클러스터링 이해

- **비지도 학습 알고리즘으로 사전에 클러스터 개수 k와 초기값을 입력하면 각 데이터의 그룹을 할당해 나가는 알고리즘**

  - a. 일단 K개의 임의의 중심점(centroid)을 배치하고, 각 데이터들을 가장 가까운 중심점으로 할당 (일종의 군집을 형성)



<군집화 전 데이터 포인트들>



<군집수를 3개로 정하고 랜덤하게 군집 중심들 배치>

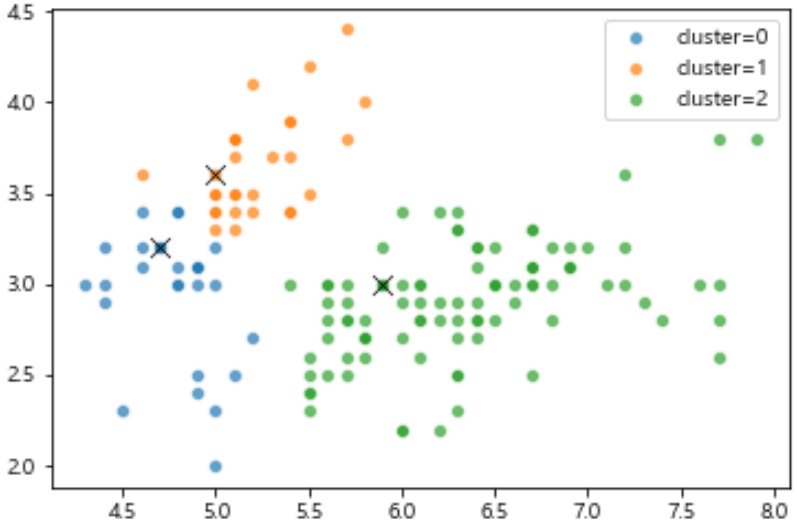
대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

- 비지도 학습 알고리즘으로 사전에 클러스터 개수 k와 초기값을 입력하면 각 데이터의 그룹을 할당해 나가는 알고리즘

- b. 데이터 포인트들과 Centroid들 간 유클리드 거리를 계산하며 Centroid들의 위치를 계속 수정

  · 각각의 데이터 포인트와 Centroid 간 유클리드 거리를 계산

  · 가장 짧은 거리의 Centroid로 데이터 포인트를 할당

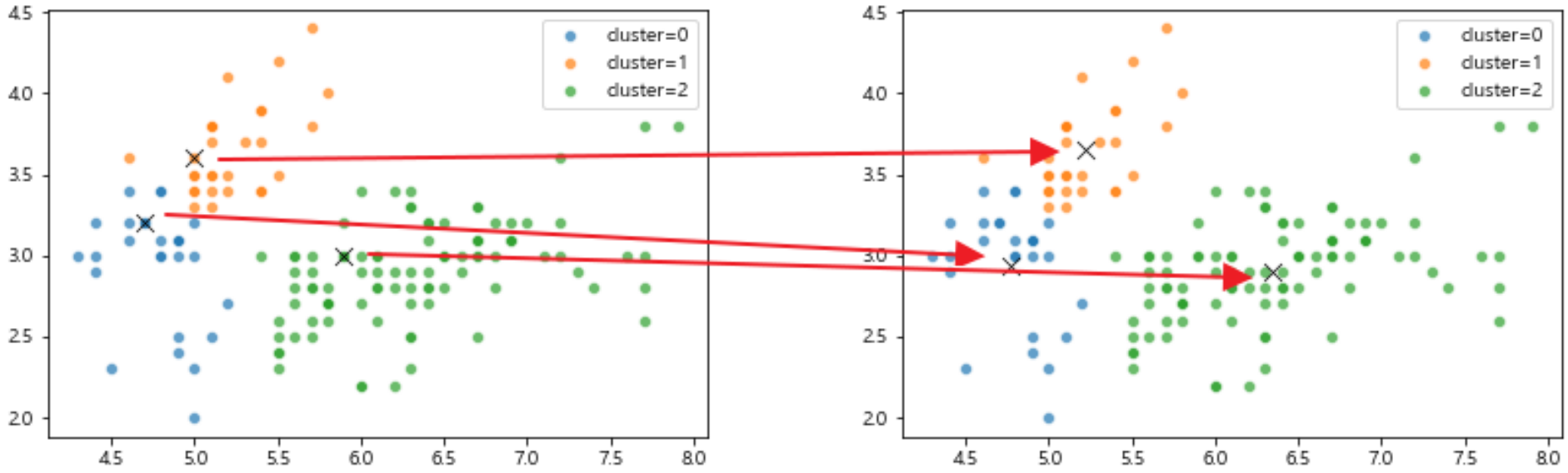


<모든 데이터 포인트들을 가까운 군집(클러스터)에 할당>

대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

- 비지도 학습 알고리즘으로 사전에 클러스터 개수 k와 초기값을 입력하면 각 데이터의 그룹을 할당해 나가는 알고리즘

  - c. 데이터 포인트들의 배정이 끝난 후, Centroid들을 배정된 데이터 포인트의 평균값으로 이동



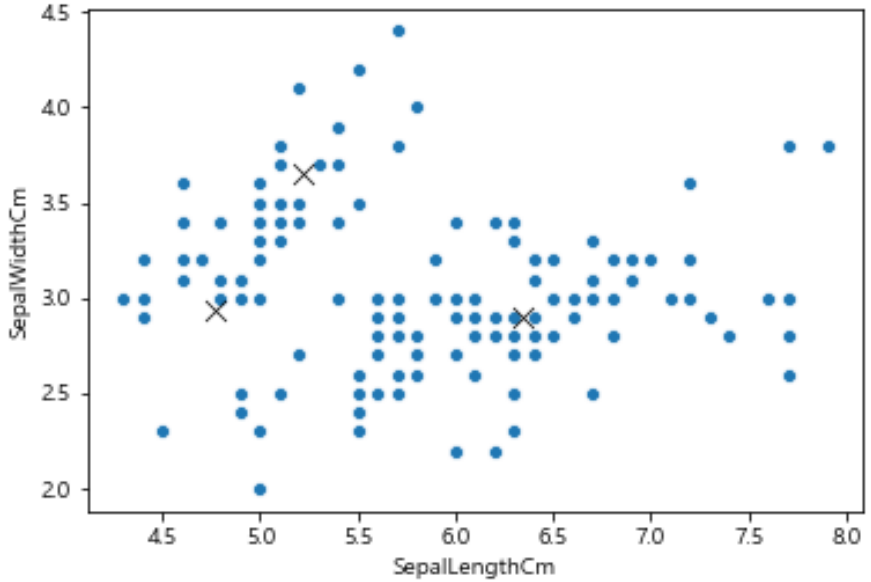
<Centroid 위치 변경 전>

<데이터 포인트들의 평균 값으로 Centroid 위치 변경 후>

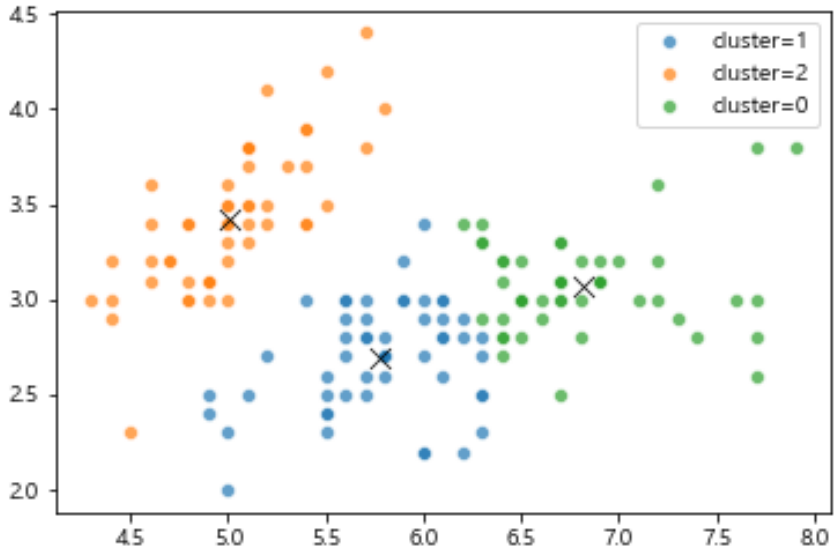
대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

- 비지도 학습 알고리즘으로 사전에 클러스터 개수 k와 초기값을 입력하면 각 데이터의 그룹을 할당해 나가는 알고리즘

  - d. Centroid들이 더 이상 움직이지 않을 때까지 반복



<변경된 Centroid 위치로 다시 거리를 계산하여 클러스터에 할당>



<Centroid 위치가 변경되지 않을 때까지 반복>

[출처] https://studying-haeung.tistory.com/6

대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

● 클러스터 6개로 군집 분석

- **KMeans 객체 생성**

```
from sklearn.cluster import Kmeans

kmeans = KMeans(n_clusters=6, n_init=10)
```

- **어떤 클러스트에 속하는지 클러스터 인덱스 번호 출력**

```
predict = kmeans.fit_predict(df)
print(predict)
```

```
[4 2 2 ... 0 0 0]
```

대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

- **데이터 프레임으로 변환**

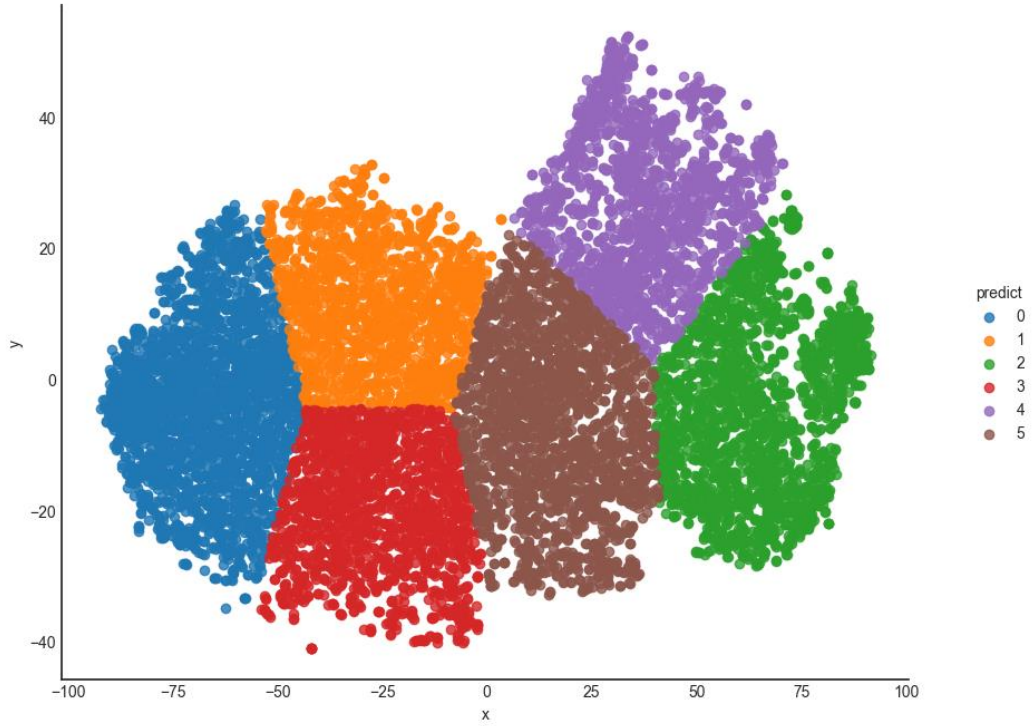
```
results = df
results['predict'] = predict
results[:10]
```

	x	y	predict
영화	66.173157	29.204741	4
거	71.904938	-2.111635	2
점	73.436226	24.978096	2
연기	29.654057	51.565075	4
최고	59.791656	36.617825	4
평점	73.873566	24.674454	2
이거	68.221382	24.188896	2
생각	71.849144	-15.455261	2
스토리	90.166183	5.329839	2
드라마	53.769367	43.330643	4

대표적인 비계층적 군집화 알고리즘인 kMeans를 사용해 실습

- **시각화**

```
map = sns.lmplot(x='x', y='y', data = results, fit_reg = False, hue = 'predict')
map.figure.set_size_inches(10, 7)
plt.show();
```



6. 감정 분석

- 감정을 분석하는 방법
- afinn 감정 어휘 사전을 이용한 감정 분석
- Scikit-learn을 이용한 기계학습 감적분석

감정을 분석하는 방법

● 감정 어휘 사전을 이용한 감정 상태 분류

- 미리 분류해 둔 감정어 사전을 통해 분석하고자 하는 텍스트의 단어들을 사전에 기반해 분류하고, 그 감정 점수를 계산
- 이 때 사용되는 감정어 사전에는 해당 감정에 해당되는 단어를 미리 정의해둬야 함

● 기계학습을 이용한 감정 상태 분류

- 분석 데이터의 일부를 훈련 데이터로 사용해 그로부터 텍스트의 감정 상태를 분류
- 이 때 사용되는 훈련 데이터는 사용자가 분류한 감정 라벨이 포함되어 있어야 하며, 이를 인공 신경망, 의사 결정 트리 등의 기계 학습 알고리즘을 사용하여 분류

afinn 감정 어휘 사전을 이용한 감정 상태 분류

● 감정 사전 준비

- 감정 사전 라이브러리 afinn을 설치
- afinn은 영어에 대한 긍정, 부정에 대한 감정 사전을 제공

● 데이터 준비

- 사용할 데이터를 구성 : 사이킷런에 내장되어 있는 뉴스그룹 데이터를 이용
- 텍스트 파일 불러오기

```
f = open('data_set/newsdata.txt', 'r', encoding='utf-8')
raw = f.read()
print(raw[:1000])
```

```
["From: lerxst@wam.umd.edu (where's my thing)\nSubject: WHAT car is this!?\nNntp-Posting-Host:
rac3.wam.umd.edu\nOrganization: University of Maryland, College Park\nLines: 15\n\n I was wondering if anyone out there
could enlighten me on this car I saw\nthe other day. It was a 2-door sports car, looked to be from the late 60s/\nearly 70s.
It was called a Bricklin. The doors were really small. In addition,\nthe front bumper was separate from the rest of the body.
This is \nall I know. If anyone can tellme a model name, engine specs, years\nof production, where
```

afinn 감정 어휘 사전을 이용한 감정 상태 분류

- **스트링을 리스트로 변환**

```
raw = eval(raw) #스트링을 리스트로 변환
raw[:1]
```

```
[“From: lerxst@wam.umd.edu (where’s my thing)\nSubject: WHAT car is this!?\nNntp-Posting-Host:
rac3.wam.umd.edu\nOrganization: University of Maryland, College Park\nLines: 15\n\n I was wondering if anyone out there
could enlighten me on this car I saw\nthe other day. It was a 2-door sports car, looked to be from the late 60s/\nearly 70s.
It was called a Bricklin. The doors were really small. In addition,\nthe front bumper was separate from the rest of the body.
This is \nall I know. If anyone can tellme a model name, engine specs, years\nof production, where this car is made, history,
or whatever info you\nhave on this funky looking car, please e-mail.\n\nThanks,\n- IL\n   ---- brought to you by your
neighborhood Lerxst ----\n\n\n\n\n”]
```

afinn 감정 어휘 사전을 이용한 감정 상태 분류

● 감정 상태 분류 및 시각화

- 감정 사전을 구성하고 감정 스코어를 측정
- afinn 라이브러리는 감정 사전과 더불어 편리하게 감정가를 계산할 수 있는 함수를 제공
- afinn 객체 생성

```
from afinn import Afinn
afinn = Afinn()
```

- 뉴스별 감정 점수 측정

```
for i in range(10):
    print(afinn.score(raw[i]))
```

```
7.0
11.0
16.0
5.0
-23.0
-25.0
7.0
3.0
16.0
-20.0
```

afinn 감정 어휘 사전을 이용한 감정 상태 분류

- **plt 스타일 설정**

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')
plt.rc('font', family='Gulim', size=13)
```

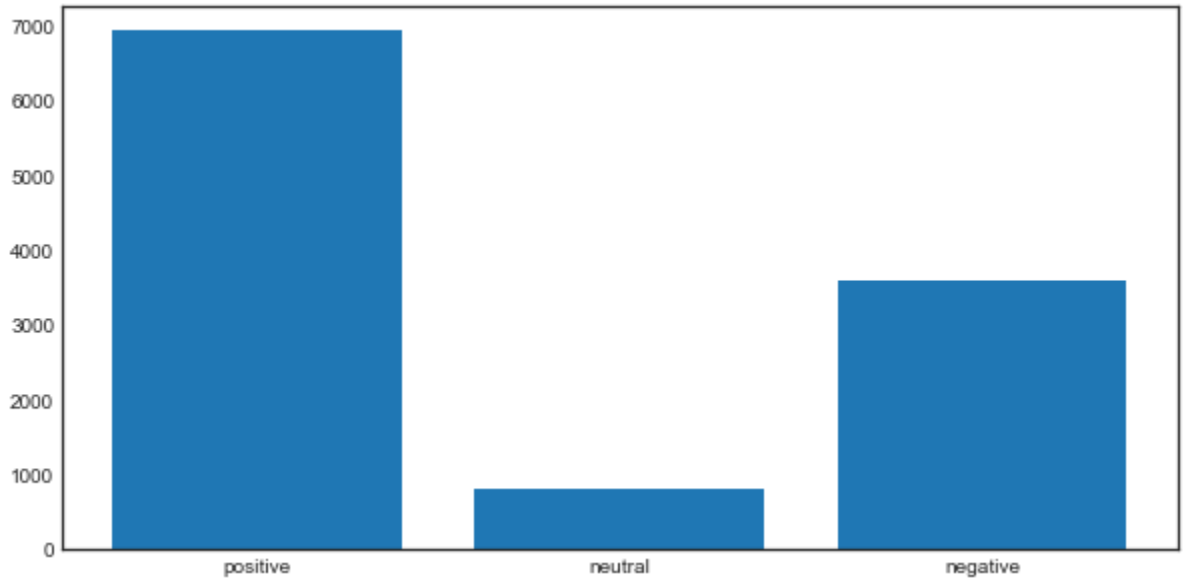
- **시각화**

```
from tqdm import tqdm
positive = 0
neutral = 0
negative = 0
for i in tqdm(raw):
    score = afinn.score(i)
    if score > 0:
        positive += 1
    elif score == 0:
        neutral += 1
    else :
        negative += 1
```

afinn 감정 어휘 사전을 이용한 감정 상태 분류

- 시각화

```
plt.figure(figsize=(10,5))
x_pos = np.array(['positive', 'neutral', 'negative'])
y_pos = [positive, neutral, negative]
plt.bar(x_pos, y_pos)
plt.show();
```



Scikit-learn을 이용한 기계학습 감정분석

● 데이터셋 파일 열기

- 네이버 영화 리뷰 데이터 : https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt

```
import pandas as pd

train_df = pd.read_csv('data_set/ratings.txt', sep = '\t', encoding = 'utf-8')
train_df
```

- 셈플 데이터 선정 : 5만개

```
train_df = train_df.sample(n=50000, random_state=0)
```

	id	document	label
0	8112052	어릴때보고 지금다시봐도 재밌어요ㅋㅋ	1
1	8132799	디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산...	1
2	4655635	폴리스스토리 시리즈는 1부터 뉴까지 버릴께 하나도 없음.. 최고.	1
3	9251303	와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이런...	1
4	10067386	안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화.	1
...	...	...	...
199995	8963373	포켓 몬스터 짜가 ㅡㅡ;;	0
199996	3302770	쓰.레.기	0
199997	5458175	완전 사이코영화. 마지막은 더욱더 이 영화의질을 떨어트린다.	0
199998	6908648	왜난 재미없었지 ㅠㅠ 라따뚜이 보고나서 스머프 봐서 그런가 ㅋㅋ	0
199999	8548411	포풍저그가나가신다영차영차영차	0

200000 rows × 3 columns

Scikit-learn을 이용한 기계학습 감정분석

● 결측치 처리

```
train_df = train_df[train_df['document'].notnull()]

print(train_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 49996 entries, 66441 to 104884
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        49996 non-null  int64
 1   document  49996 non-null  object
 2   label     49996 non-null  int64
dtypes: int64(2), object(1)
memory usage: 1.5+ MB
None
```

Scikit-learn을 이용한 기계학습 감정분석

● 내용과 평가를 Series 객체로 저장

```
text = train_df['document']
score = train_df['label']
type(text),type(score)
```

```
(pandas.core.series.Series, pandas.core.series.Series)
```

● 데이터 셋 분리(Train용 80%, Test용 20%)

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(text, score , test_size=0.2, random_state=0)
print(len(train_x), len(train_y), len(test_x), len(test_y))
```

```
39999 39999 10000 10000
```

Scikit-learn을 이용한 기계학습 감정분석

● TF-IDF 벡터화

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(ngram_range=(1,2), min_df=3, max_df=0.9)
tfidf.fit(train_x)
print(tfidf.get_feature_names_out()[:100])
```

```
['007' '007 시리즈' '0개는' '0점' '0점도' '0점도 아깝다' '0점은' '0점은 없나' '0점은 없는거야'
 '0점을' '0점을 주고' '0점이' '0점이 없네' '10' '100' '100배' '100점' '100퍼' '100퍼센트'
 '109분' '10개' '10글자' '10년' '10년도' '10년만에' '10년은' '10년이' '10년이 지난' '10년전'
 '10년전에' '10대' '10번' '10번이상' '10분' '10분만에' '10분보고' '10분이' '10자' '10자 제한'
 '10자이상' '10점' '10점 ㅋㅋ' '10점 드립니다' '10점 만점' '10점 만점에' '10점 분들은' '10점 영화는'
 '10점 주고' '10점 주는' '10점 준다' '10점도' '10점만점' '10점만점에' '10점만점에 10점' '10점밖에'
 '10점에' '10점으로' '10점은' '10점을' '10점이' '10점이 아깝지' '10점이다' '10점주는' '10점주는 매미'
 '10점주는 매미oo있네' '10점준' '10점준 m창있네' '10점준 매미oo있네' '10점준다' '10점줌' '10점짜리'
 '10점짜리 영화는' '11' '11점을' '12년이' '12세' '13구역' '13세' '13일의' '14' '14년전'
 '15세' '16' '17년이' '18' '18세' '19금' '19세' '19세기' '1人' '1개' '1개도' '1과'
 '1년에' '1도' '1등' '1등은' '1번' '1보다' '1빠']
```

Scikit-learn을 이용한 기계학습 감정분석

```
tfidf_train_x = tfidf.transform(train_x)
print(tfidf_train_x[:4])
```

```
 (0, 15743)        0.48661847110280054
  (0, 14915)       0.4012243293078031
  (0, 7733)        0.44294862477191194
  (0, 1672)        0.42362440992647304
  (0, 582)         0.47598358824669124
  (1, 12140)       0.359957943816268
  (1, 12112)       0.217330835971694
  (1, 10882)       0.23168251747239282
  (1, 6478)        0.4364661109409043
  (1, 2805)        0.4251001010439379
  (1, 2189)        0.4434377685752297
  (1, 2182)        0.30795202307789027
  (1, 1015)        0.32685858179474964
  (2, 14737)       0.5605465632162288
  (2, 12821)       0.5114095530281735
  (2, 11105)       0.4918603117422375
  (2, 3571)        0.426990928792202
  (3, 12408)       1.0
```

Scikit-learn을 이용한 기계학습 감정분석

● 감성 분석 모델 구축 :로지스틱 회귀(Logistic Regression)

▪ 로지스틱 회귀(Logistic Regression)분석

- 일상 속 풀고자 하는 많은 문제 중에서는 두 개의 선택지 중에서 정답을 고르는 문제가 많다.
- 감정분류에서는 긍정과 부정을 결정하는 문제에 활용한다.
- 둘 중 하나를 결정하는 문제를 이진 분류(Binary Classification)라고 하며, 이런 문제를 풀기 위한 대표적인 알고리즘으로 로지스틱 회귀를 활용한다.

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0)
clf.fit(tfidf_train_x, train_y)
```

Scikit-learn을 이용한 기계학습 감정분석

● 분석 모델 평가

```
tfidf_test_x = tfidf.transform(test_x)

test_predict = clf.predict(tfidf_test_x)
from sklearn.metrics import accuracy_score
print('감정 분류 모델의 정확도 : ',round(accuracy_score(test_y, test_predict), 3))
```

```
감정 분류 모델의 정확도 :  0.775
```

▪ 평가 데이터 시각화

```
test_df = pd.DataFrame({'text':test_x, 'label':test_y, 'predict':test_predict})
test_df
```

	text	label	predict
29560	단순한 추억의 영화 그 이상.,	1	1
198035	숨겨진 이영화의 X맨을 찾아라.	0	1
18143	우리주변에서일어나는모든순간을섬세한감성으로끌어낸영화,감동에젖어울고갑니다.	1	0
148881	잘만 킹 만으로는 부족하다. 나인 하프 위크 기대하고 봤다가 낭패;;;	0	0
62409	감동 그 자체 입니다..	1	1
...	...	...	...
135581	조디포스터 때문에 봤다	0	0
105023	제목이 원초적이나 내용은 아님....	0	0
79195	번개치고 귀신튀어나오는 성궤나 불사의 영약 성배나 고대 중남미문명 외계인 개입설이나...	1	1
172524	B급냄새너무많이남. 같잖은 철학과 존내. 지루함	0	0
78226	아카데미 상 은 개나소나 받는게 아니다.	1	0

Scikit-learn을 이용한 기계학습 감정분석

● 감정 분류

```
input_text = ['딱히 대단한 재미도 감동도 없는데 ~! 너무 과대 평과된 영화 중 하나']

#입력 텍스트의 피처 벡터화
st_tfidf = tfidf.transform(input_text)

#최적 감성 분석 모델에 적용하여 감성 분석 평가
st_predict = clf.predict(st_tfidf)

#예측 결과 출력
if(st_predict == 0):
    print('예측 결과: ->> 부정 감성')
else :
    print('예측 결과: ->> 긍정 감성')
```

```
예측 결과: ->> 부정 감성
```

With POSCO

감사합니다

posco
포스코인재창조원