

텍스트 데이터 마이닝

2021년 5월
대진대학교 컴퓨터공학과 서혜선교수
jako403@daejin.ac.kr

텍스트 데이터 마이닝

- 워드 클라우드
- 토픽 모델링
- 감성분석

워드클라우드

- 문서의 키워드, 개념 등을 직관적으로 파악할 수 있도록 핵심단어를 시각적으로 돋보이게 하는 방법
- 단어의 빈도에 따라 글자의 크기가 다르게 표현되기 때문에 어떤 단어가 얼마나 많이 사용되는지를 한눈에 파악 가능

워드클라우드

■ 전처리가 완료된 데이터파일 불러오기

```
import pickle
import pandas as pd
```

```
file_path = 'F:\보험연수원\파이썬과크롤링\data\data.pkl' # 파일 경로 바꿀것
```

```
import pickle
f = open(file_path + 'total_doc.pkl', "rb")
#pickle 모듈을 사용할 때는 파일 형식을 바이트(b) 형식으로 읽고 써야 함
data = pickle.load(f)
f.close()
data
```

	title	doc	url	ch	ch2	token_list_pos	token_noun
0	연금계좌관련 문의드립니다.	안녕하세요 주린이 개월차입니다국민연금에 퇴직연금이 있어서 그동안 개인연금은 관심을 ...	https://cafe.naver.com/likeusstock/158991?art=...	naver	cafe	[(안녕하세요, NNP), (주리, VV), (L, ETM), (이, NNP), (...	[개월, 차입, 국민연금, 퇴직, 개인연금, 관심, 두지, 개인연금, 나스닥, 구매...
1	국민연금 추납과 소득공제	와이프가 결혼 전 일을 하다가 결혼 후 육아로 일을 쉬고 있습니다 한의사라 연제가 ...	https://cafe.naver.com/hotellife/1604742?art=Z...	naver	cafe	[(와이프, NNG), (가, JKS), (결혼, NNG), (전, MM), (일,...	[와이프, 결혼, 결혼, 육아, 한의사, 국민연금, 제한, 소식, 임의, 가입, 시...

워드클라우드

1. 문서 전체의 명사 확보

■ 각 문서별 명사 확인

개별 문장에 포함되는 명사들 추출

```
data['token_noun'] #각 문서별 명사 확인
```

```
0      [개월, 차입, 국민연금, 퇴직, 개인연금, 관심, 두지, 개인연금, 나스닥, 구매...
1      [와이프, 결혼, 결혼, 육아, 한의사, 국민연금, 제한, 소식, 임의, 가입, 시...
2      [퇴직금, 계산, 방법, 지급, 기준, 퇴직금, 계산, 방법, 퇴직금, 계산, 방법...
3      [개인, 주식, 가게, 자금, 펀드, 투자, 운용, 퇴직, 펀드, 근속, 기간, 중...
4      [준비, 공부, 국세청, 미리, 계산, 만원, 환수, 방법, 저축, 대면, 계좌, ...

...

212     [환급, 제도, 환급, 제도, 탈퇴, 일시, 탈퇴, 일시, 환급, 신청, 이란, 일...
213     [주택, 상속자, 아파트, 정액, 후후, 박형, 종류, 정액, 지급, 평생, 고전,...
214     [개인연금, 퇴직, 퇴직, 불입, 저축, 개인연금, 퇴직, 분할, 필요, 하시나, ...
215     [회원, 추천, 해주시, 마법, 공무원, 쥐꼬리, 공무원, 개인연금, 저축, 종합,...
216     [과목, 레포트, 주제, 복지, 법제, 국민연금, 국민연금, 법의, 필요, 국민연금...
Name: token_noun, Length: 217, dtype: object
```

워드클라우드

1. 문서 전체의 명사 확보

■ 문서 전체의 모든 명사에 대한 리스트 구성 [import itertools]

```
import itertools #문서 전체의 명사리스트 확보
noun = list(itertools.chain(*data['token_noun'])) #리스트 언패킹
noun
```

```
['개월',
 '차입',
 '국민연금',
 '퇴직',
 '개인연금',
 '관심',
 '두지',
 '개인연금',
 '나스닥',
 '구매',
 '가입',
 '계좌',
 '개설',
 '연간',
 '한도',
 ...]
```

▪ itertools

- iterable한 (반복가능한) 객체를 이용할 때 사용하는 라이브러리

워드클라우드

2. 상위 빈도 단어 확인

■ 상위 빈도 단어 확인 [import Counter]

단어별 상위빈도를 구하고, 빈도가 많은 상위 50개 단어 출력

```
from collections import Counter #단어들을 쉽게 집계하기 위해서 사용
count = Counter(noun) #리스트 원소의 개수가 계산됨
top = dict(count.most_common(50)) # 상위 50개 출력하기
top
```

```
{'저축': 547,
'보험': 547,
'상품': 475,
'계좌': 413,
'가입': 406,
'펀드': 393,
'금액': 370,
'만원': 362,
'퇴직': 356,
'투자': 351,
'납입': 325,
'소득': 307,
'공제': 299,
'준비': 282,
'수익률': 270,
'수령': 262,
'노후': 238,
'수익': 227,
'기간': 213,
```

- collections
 - 데이터의 개수를 셀 때 유용한 라이브러리
- count.most_common(k)
 - 데이터의 개수가 많은 순으로 정렬해 k개를 반환

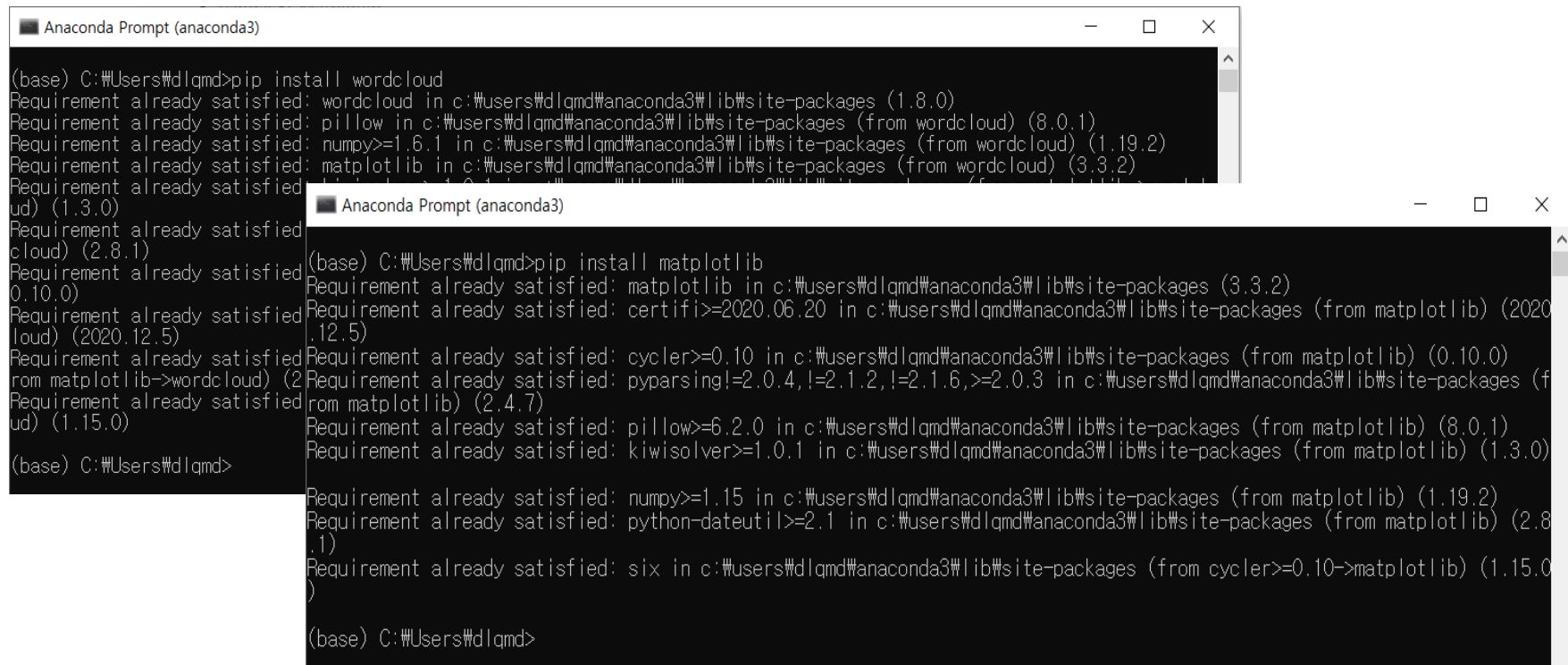
워드클라우드

3. 워드클라우드

- 라이브러리 설치 : 주피터에서 설치 또는 프롬프트에서 설치

`pip install wordcloud / pip install matplotlib`

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```



The image shows two screenshots of the Anaconda Prompt (anaconda3) terminal window. The first screenshot shows the command `pip install wordcloud` being executed, with the output indicating that the package is already installed (version 1.8.0) and its dependencies (pillow, numpy, matplotlib) are also satisfied. The second screenshot shows the command `pip install matplotlib` being executed, with the output indicating that the package is already installed (version 3.3.2) and its dependencies (certifi, cycler, pyparsing, pillow, kiwisolver, numpy, python-dateutil, six) are also satisfied.

```
Anaconda Prompt (anaconda3)
(base) C:\Users\dlqmd>pip install wordcloud
Requirement already satisfied: wordcloud in c:\users\dlqmd\anaconda3\lib\site-packages (1.8.0)
Requirement already satisfied: pillow in c:\users\dlqmd\anaconda3\lib\site-packages (from wordcloud) (8.0.1)
Requirement already satisfied: numpy>=1.6.1 in c:\users\dlqmd\anaconda3\lib\site-packages (from wordcloud) (1.19.2)
Requirement already satisfied: matplotlib in c:\users\dlqmd\anaconda3\lib\site-packages (from wordcloud) (3.3.2)
Requirement already satisfied: ...
(base) C:\Users\dlqmd>

Anaconda Prompt (anaconda3)
(base) C:\Users\dlqmd>pip install matplotlib
Requirement already satisfied: matplotlib in c:\users\dlqmd\anaconda3\lib\site-packages (3.3.2)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\dlqmd\anaconda3\lib\site-packages (from matplotlib) (2020.12.5)
Requirement already satisfied: cycler>=0.10 in c:\users\dlqmd\anaconda3\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\dlqmd\anaconda3\lib\site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\dlqmd\anaconda3\lib\site-packages (from matplotlib) (8.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\dlqmd\anaconda3\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: numpy>=1.15 in c:\users\dlqmd\anaconda3\lib\site-packages (from matplotlib) (1.19.2)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\dlqmd\anaconda3\lib\site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: six in c:\users\dlqmd\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib) (1.15.0)
(base) C:\Users\dlqmd>
```


워드클라우드

3. 워드클라우드

■ 워드클라우드 그리기 (기본)

TF-IDF

TF-IDF(Term Frequency-Inverse Document Frequency)

TF-IDF : 단어의 빈도와 역 문서 빈도를 사용하여 DTM(Document-Term Matrix, 문서 단어 행렬) 내의 각 단어마다 중요한 정도를 가중치로 주는 방법

- TF: 한 문서 내에서 특정 단어가 출현한 빈도수
(문서내 출현 빈도가 높을수록 상대적으로 더 중요)
- IDF: 문서 집합의 전체 문서 수를 특정 단어가 나타나는 문서의 수로 나눈 값
(높은 IDF 값을 가지는 단어는 문서 내에서 중요한 의미를 가지는 단어)
- TF-IDF: 두 값을 곱해서 사용하므로 어떤 단어가 해당 문서에 자주 등장하지만, 다른 문서에는 출현 빈도가 낮을수록 높은 값을 가지게 됨

TF-IDF(Term Frequency-Inverse Document Frequency)

- 단어의 중요도를 표현하는데 있어 단어의 빈도수를 기반으로 측정시 문서의 길이가 길수록 단어의 빈도가 증가하는 경향이 있어 이런 문제점을 해결하기 위한 방안이 TF-IDF 방법임

TF

Term Frequency

특정 단어의 등장 빈도

DF

Document Frequency

특정 단어가 나타나는 문서 수

IDF

Inverse Document Frequency의 약자

특정 단어가 나타나는 문서 수의 역수

TF-IDF

단어의 빈도와 역 문서 빈도를 사용함으로써,

DTM 내의 각 단어들마다 중요한 정도를 가중치로 주는 방법

TF-IDF(Term Frequency-Inverse Document Frequency)

DTM (Document – Term Matrix)

문서단어행렬(DTM)은 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것
즉 각 문서에 대한 BoW(Bag-of_Words)를 하나의 행렬로 만든 것

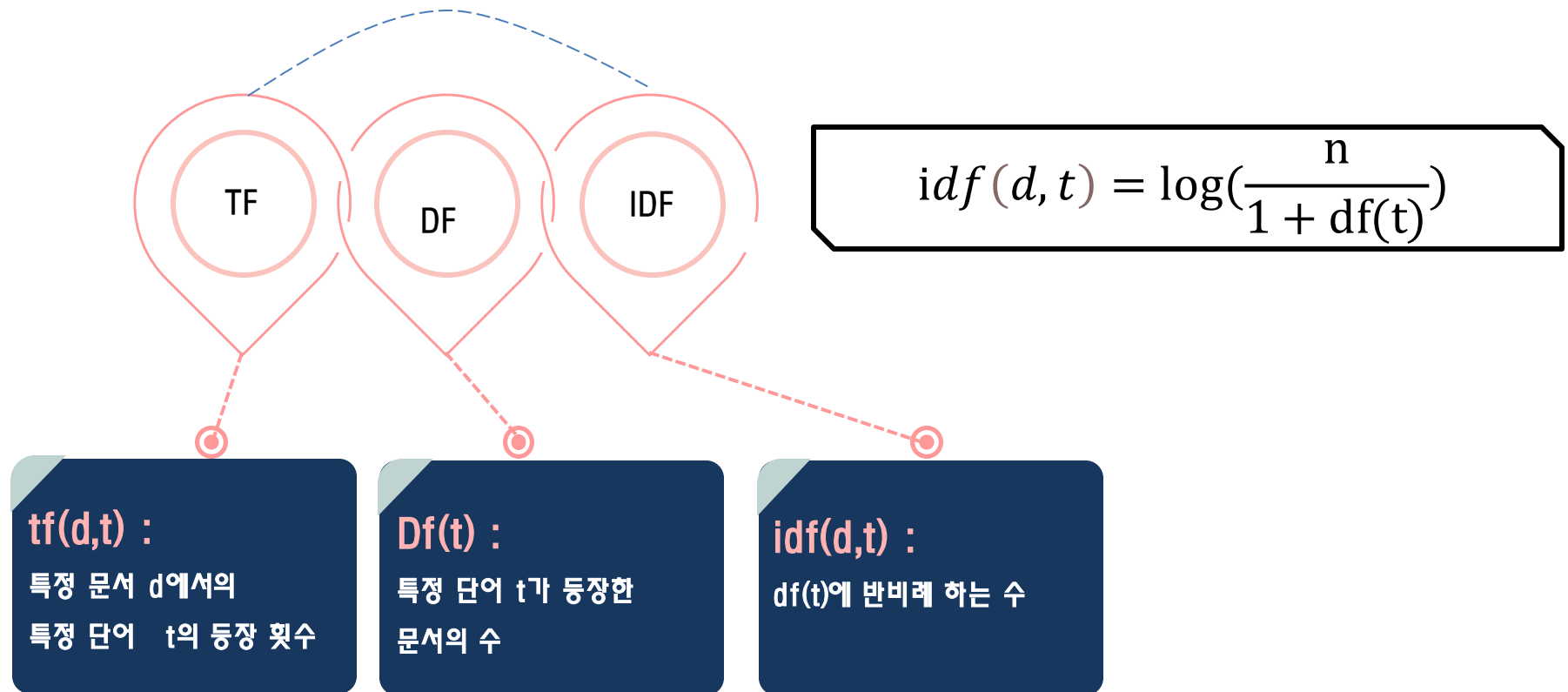
문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

Bag-of-Words(BoW)

단어를 벡터의 열로 할당하고 해당 단어의 출현빈도를 요소로 만든 벡터

TF-IDF(Term Frequency-Inverse Document Frequency)



TF-IDF(Term Frequency-Inverse Document Frequency)

Mathematical TF-IDF()

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

자연로그 \log_e
($e = 2.718281...$)

단어	IDF(역 문서 빈도)
과일이	$\ln(4/(1+1)) = 0.693147$
길고	$\ln(4/(1+1)) = 0.693147$
노란	$\ln(4/(1+1)) = 0.693147$
먹고	$\ln(4/(2+1)) = 0.287682$
바나나	$\ln(4/(2+1)) = 0.287682$
사과	$\ln(4/(1+1)) = 0.693147$
싶은	$\ln(4/(2+1)) = 0.287682$
저는	$\ln(4/(1+1)) = 0.693147$
좋아요	$\ln(4/(1+1)) = 0.693147$

TF-IDF 값

로그 0

		과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0	0.693147	0.693147

TF-IDF(Term Frequency-Inverse Document Frequency)

Mathematical TF-IDF()

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

자연로그 \log_e
($e = 2.718281\dots$)

로그(No)

$idf(d, t) = n/df(t)$
 $n = 1,000,000$

단어 t	$df(t)$	$idf(d, t)$
word1	1	1,000,000
word2	100	10,000
word3	1,000	1,000
word4	10,000	100
word5	100,000	10
word6	1,000,000	1



로그(Yes)

$idf(d, t) = \log(n/df(t))$
 $n = 1,000,000$

단어 t	$df(t)$	$idf(d, t)$
word1	1	6
word2	100	4
word3	1,000	3
word4	10,000	2
word5	100,000	1
word6	1,000,000	0

TF-IDF(Term Frequency-Inverse Document Frequency)

단어	IDF(역 문서 빈도)
과일이	$\ln(4/(1+1)) = 0.693147$
길고	$\ln(4/(1+1)) = 0.693147$
노란	$\ln(4/(1+1)) = 0.693147$
먹고	$\ln(4/(2+1)) = 0.287682$
바나나	$\ln(4/(2+1)) = 0.287682$
사과	$\ln(4/(1+1)) = 0.693147$
싶은	$\ln(4/(2+1)) = 0.287682$
저는	$\ln(4/(1+1)) = 0.693147$
좋아요	$\ln(4/(1+1)) = 0.693147$

TF-IDF 값

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

→ 이러한 TF-IDF값을 기반으로 SNS분석을 실시함

TF-IDF(Term Frequency-Inverse Document Frequency)

- 1) 각 문서의 명사들에 대해 문자열(str)을 구성
- 2) TF-IDF matrix를 계산

```
: #TF-IDF 계산을 위해 명사들의 문자열 구성
doc_noun = []
for i in range(0, len(data['token_noun'])):
    doc_noun.append(' '.join(data['token_noun'][i])) #각 문서의 명사들을 str으로 연결

: #텍스트 문서 모음을 단어 tf-idf 행렬로 변환
from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer(min_df = 0.01, max_df=0.95)
#문서의 1%~95%로 나타나는 단어들을 고려.
#min_df: 특정 단어가 나타나는 '문서의 수의 최소 빈도값을 설정/소수부분만있는 형태면 %
X = vec.fit_transform(doc_noun)
```

TF-IDF(Term Frequency-Inverse Document Frequency)

1) 문서 인덱스, 등장단어 인덱스, TF-IDF값 출력

```
print(X)      # (문서인덱스, 등장단어인덱스) tf-idf값
(0, 719)      0.13138653859066382
(0, 308)      0.11938900944427278
(0, 938)      0.20219200214826835
(0, 40)       0.19710442691704327
(0, 524)      0.15096535545823173
(0, 1032)     0.18443556396464134
(0, 1001)     0.22177081901583626
(0, 453)      0.1778131947135696
(0, 41)       0.1612782729294825
(0, 611)      0.20219200214826835
(0, 275)      0.09320163163895284
(0, 761)      0.38500431061353374
(0, 1072)     0.14569241606912822
(0, 610)      0.15285921795068239
(0, 26)       0.1612782729294825
(0, 60)       0.2022306685848284
(0, 9)        0.08001724097315857
(0, 107)      0.2305746354234529
(0, 176)      0.4838348187884475
(0, 101)      0.14247206940588938
(0, 31)       0.24272038699321472
(0, 1013)     0.10245673454470965
(0, 118)      0.10822924755550498
(0, 29)       0.14405540154306573
(0, 15)       0.0977507190377569
:
:
(215, 309)    0.1515076869238406
(215, 1017)   0.241435388402589
(215, 882)    0.29245920369491774
(215, 15)     0.26228443610063834
(215, 503)    0.11634251248590191
(215, 871)    0.14429912931759156
```

K-Means

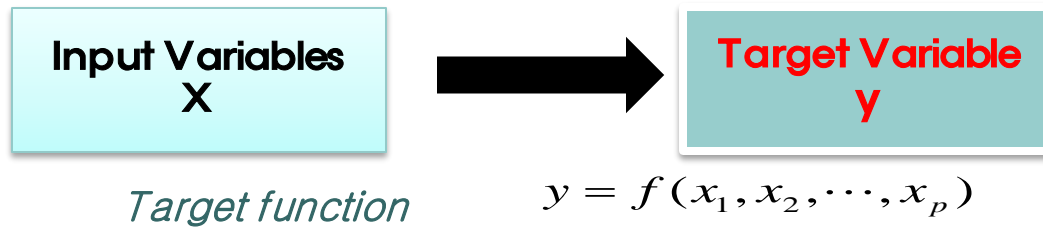
- 대표적인 클러스터링(비슷한 유형의 데이터를 그룹화) 알고리즘
- “K” : 데이터의 세트에서 찾을 것으로 예상되는 클러스터(그룹) 수
“Means” : 각 데이터로부터 그 데이터가 속한 클러스터의 중심까지의 평균 거리

K-Means Clustering(K-평균 군집화)

통계적 학습모형

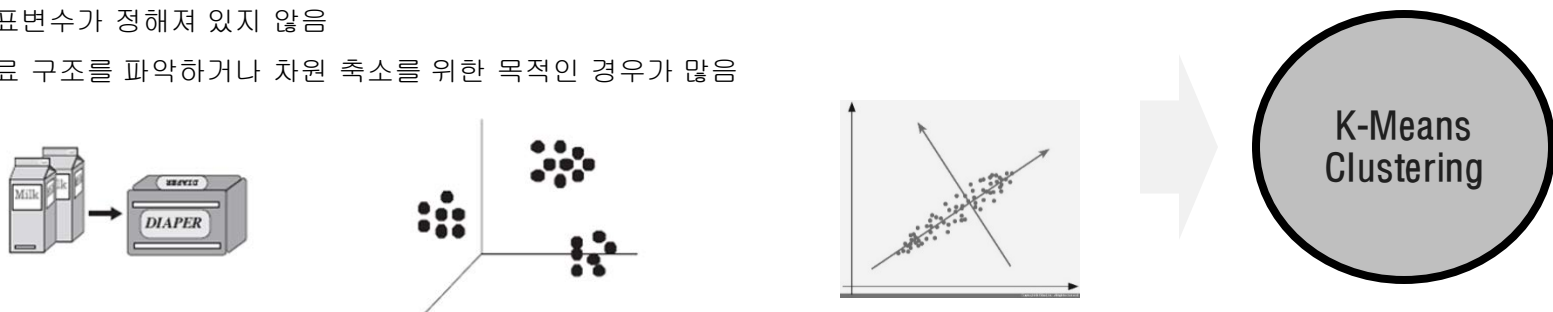
■ 지도 학습모형 (Supervised learning)

- 목표변수(target variable)가 정해져 있으며, 목표변수를 예측하는 것이 목적인 학습모형



■ 비지도 학습모형 (Unsupervised learning)

- 목표변수가 정해져 있지 않음
- 자료 구조를 파악하거나 차원 축소를 위한 목적인 경우가 많음



K-Means Clustering(K-평균 군집화)

- 목표변수를 가지고 있지 않을 때 데이터들에 숨겨진 패턴을 찾아 구조화하고, 유사패턴끼리 군집화 하도록 하는 비지도학습 머신러닝 기법중 대표적인 것이 K-means clustering 임
 - Recommendation Engines(추천 엔진)
 - : 사용자 경험 기반의 상품 그룹핑
 - Search Engines(검색 엔진)
 - : 뉴스주제, 검색결과 등을 군집화
 - Market Segmentation (시장 세분화)
 - : 인구통계, 지역, behavior, life-style 등을 바탕으로 군집화

K-Means Clustering(K-평균 군집화)

▪ K-Means 알고리즘

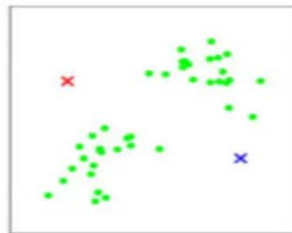
- 1) 먼저 데이터셋에서 K개의 임의의 중심점(centroids)를 지정
- 2) 각 데이터들을 가장 가까운 중심점이 속한 그룹에 할당
- 3) 2)번 결과를 바탕으로 새로운 중심점을 지정
- 4) 2번, 3번 단계를 수렴이 될 때까지, 즉 군집 중심의 변화가 거의 없을 때까지 반복

두 객체 a, b의 가장 가까운 거리(유클리디안 거리 사용)

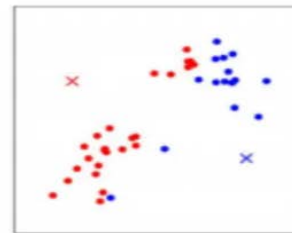
$$distance = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots + (a_n - b_n)^2}$$



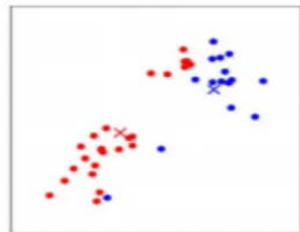
(a)



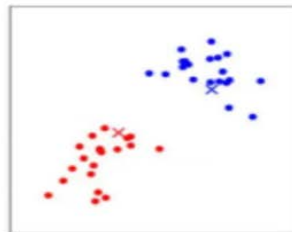
(b)



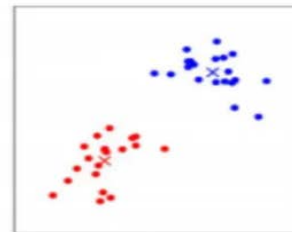
(c)



(d)



(e)



(f)

→ 오차제곱합(SSE)이 최소가 되도록 할당하는 과정

K-Means Clustering(K-평균 군집화)

■ 최적의 클러스터 수(군집 수) 확인

```
from sklearn.cluster import KMeans
def elbow(X):
    sse = []

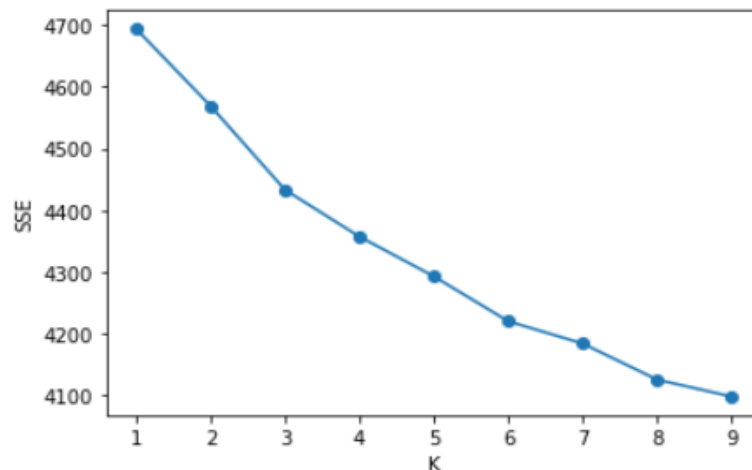
    for i in range(1,10):
        km = KMeans(n_clusters=i, algorithm='auto', random_state=0) #k-means algorithm: auto(기본값))
        km.fit(X)
        sse.append(km.inertia_)
        print(i)

    plt.plot(range(1,10), sse, marker='o')
    plt.xlabel('K')
    plt.ylabel('SSE')
    plt.xticks(range(1,10))
    plt.show()

elbow(X)
```

▪ 엘보우(elbow) 기법

- 오차 제곱합(SSE)이 점점 줄어듦다 어느 순간 줄어드는 비율이 급격하게 작아지면, 그 부분을 최적의 클러스터 개수로 판단



K-Means Clustering(K-평균 군집화)

■ 최적의 클러스터 수로 K-Means clustering 실행

```
#최적의 클러스터 개수(3)로 k-means 실행
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3, algorithm='auto', random_state=0)
model.fit(X)
```

■ K-Means clustering 결과 출력

```
print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, :-1]
terms = vec.get_feature_names()
for i in range(3):
    print("Cluster %d:" % i)
    for ind in order_centroids[i, :30]:    #[행, 열] #[클러스터 인덱스, 클러스터 내 인덱스]
        print(' %s' % terms[ind])    #ind: 가져온 단어 인덱스 #terms[ind]: 해당 인덱스를 가진 단어
    print('\n')
```

K-Means Clustering(K-평균 군집화)

Top terms for each cluster

Cluster 0:

보험
상품
준비
납입
저축
노후
가입
금액
유지
이율
전문가
수익률
비교
연구
보장
수익
추가
원금
기간
재테크
월급
보험료
최저
수령
보증
만원
납부
선택
해지
펀드

Cluster 1:

국민연금
퇴직
가입
만원
신청
금액
제도
주택
질문
소득
노령
지급
퇴직금
수령
회사
납입
재산
납부
부부
기초
개월
답변
직원
급여
확인
복지
노후
기간
수급
공무원

Cluster 2:

계좌
펀드
저축
공제
투자
만원
세액
퇴직
수익률
미국
매수
증권사
개인연금
주식
상품
세금
수익
정도
가입
연말정산
나스닥
자산
소득
수수료
프로
미래
금액
운용
납입
개설

LDA 토픽모델링

- 가장 대표적인 토픽모델링 기법
- 다수의 문서에서 잠재적으로 의미 있는 토픽을 발견하는, 절차적 확률 분포 모델링

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 잠재 디리클레 할당(LDA) 개요

- 토픽모델링이란 문서의 집합으로부터 토픽(주제)를 찾아내는 과정이며, 토픽모델링의 대표적인 방법이 LDA이며 토픽들은 확률분포에 기반하여 단어들을 생성한다고 가정

■ 잠재 디리클레 할당(LDA) 수행 프로세스

- 1) 주어진 단어들의 빈도수를 기반으로 문서단어행렬(DTM)을 생성
- 2) 토픽의 개수(k)를 사전에 설정
- 3) 모든 단어들을 k개중 하나의 토픽에 임의 할당
- 4) 특정 단어를 하나 추출한 후 추출한 해당 단어를 제외하고 각 문서 내 토픽 분포와 토픽 별 단어 분포를 다시 계산(추출된 단어는 새롭게 토픽 할당 분포를 계산)
- 5) 다른 단어를 추출하고 4번 단계를 다시 수행. 그리고 또 다른 단어를 추출하고 계속적으로 모든 단어들이 재계산되도록 반복
- 6) 지정된 반복 횟수만큼 4, 5번 단계를 수행하면서 모든 단어들의 토픽 할당 분포가 변경되지 않고 수렴할 때까지 수행

LDA(Latent Dirichlet Allocation) 토픽모델링

■ LDA 프로세스 예시

doc1

word	apple	tomato	apple	cat	dog
topic	B	B	?	A	A

기준1

Doc1에서 apple의 토픽을 결정
(A, B 각각에 속할 확률 50%)

doc2

word	cute	notebook	woman	apple	apple
topic	B	B	B	B	B

기준2

apple이 전체문서에서 어떤 토픽에 할당되는가?
(B에 할당될 가능성 高)

기준 1 : $\text{prob}(\text{topic } t \mid \text{doc } d)$ – 문서 d의 단어들 중 토픽 t에 해당하는 단어들의 비율

기준 2 : $\text{prob}(\text{word } w \mid \text{topic } t)$ – 각 토픽들 t에서 해당 단어 w의 분포

LDA(Latent Dirichlet Allocation) 토픽모델링

- 주어진 단어들의 빈도수를 기반으로 문서단어행렬(DTM)을 생성
- gensim 라이브러리

```
#명사 리스트 추출  
#pip install gensim  
#설치에러시 관리자권한으로 실행 (--user)
```

#명사만 추출한 리스트를 바탕으로 단어 빈도별 목록을 생성.

```
from gensim import corpora, models  
noun_dic=corpora.Dictionary(data['token_noun']) #딕셔너리 클래스로 사전생성, 각 단어별 id도 함께 생성  
noun_dic.token2id #각 단어 별 생성된 id 확인
```

```
import numpy  
from gensim import corpora, models
```

```
corpus = [noun_dic.doc2bow(text) for text in data['token_noun']]  
#문서의 단어들을 단어의 id와 빈도수로 수치화  
corpus
```

```
{'가입': 0,  
'개설': 1,  
'개월': 2,  
'개인연금': 3,  
'건지': 4,  
'검색': 5,  
'계좌': 6,  
'관심': 7,  
'구매': 8,  
'구지': 9,  
'국민연금': 10,  
'나스닥': 11,  
'두지': 12,  
'만원': 13,
```

```
[(0, 1),  
(1, 1),  
(2, 1),  
(3, 2),  
(4, 1),  
(5, 1),  
(6, 2),  
(7, 1),  
(8, 1),  
(9, 1),  
(10, 1),  
(11, 3),  
(12, 1),  
(13, 1),  
(14, 1),
```

{ LDA(Latent Dirichlet Allocation) 토픽모델링}

■ 일관성(coherence), 혼잡성(perplexity) 그래프 기반의 토픽수 선정

```
#토픽 개수에 따라 분석의 혼잡도와 일관성 분석 후, 최선의 토픽 개수를 정하여 토픽 모델링 실시
import gensim
from gensim.models import CoherenceModel

Lda = gensim.models.ldamodel.LdaModel #LDA기법: 확률을 바탕으로 단어가 특정 주제에 존재할 확률과 문서에
perplexity_score=[]
coherence_score=[]

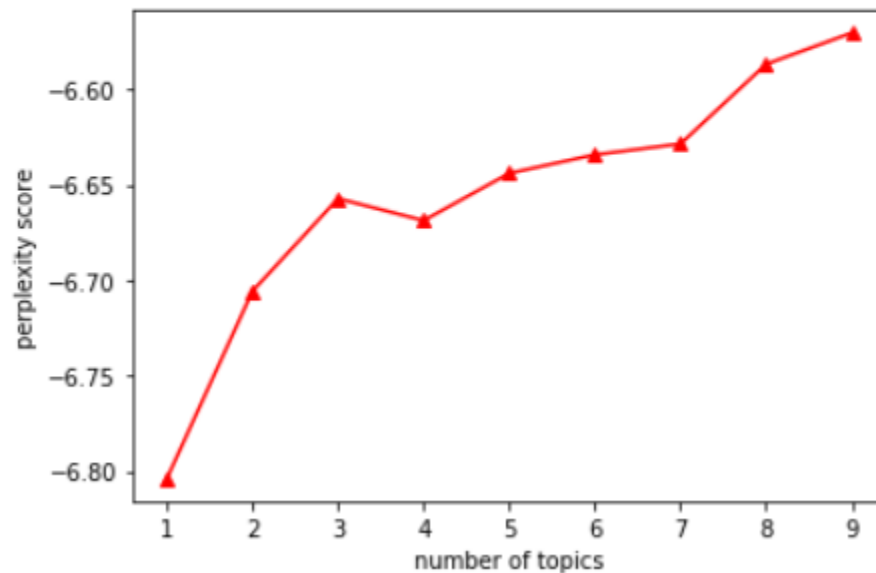
for i in range(1,10): #토픽 개수가 1,2,3,4,5,6,7,8,9인 9가지 경우 각각의 혼잡도와 일관성을 측정
    ldamodel=Lda(corpus, num_topics=i, id2word=noun_dic, passes=15, random_state=0) #passes: 모델 학습시
    perplexity_score.append(ldamodel.log_perplexity(corpus)) #혼잡도
    coherence_score.append(CoherenceModel(model=ldamodel, texts=data['token_noun'],
                                         | dictionary=noun_dic, coherence='c_v').get_coherence()) #일관성

print(i, 'process complete')
```

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 토픽 수에 따른 혼잡성 점수(perplexity score) 그래프 확인

```
plt.plot(range(1,10),perplexity_score,'r',marker='^') #(x,y,color)
plt.xlabel("number of topics")
plt.ylabel("perplexity score") #혼잡성
plt.show()
```

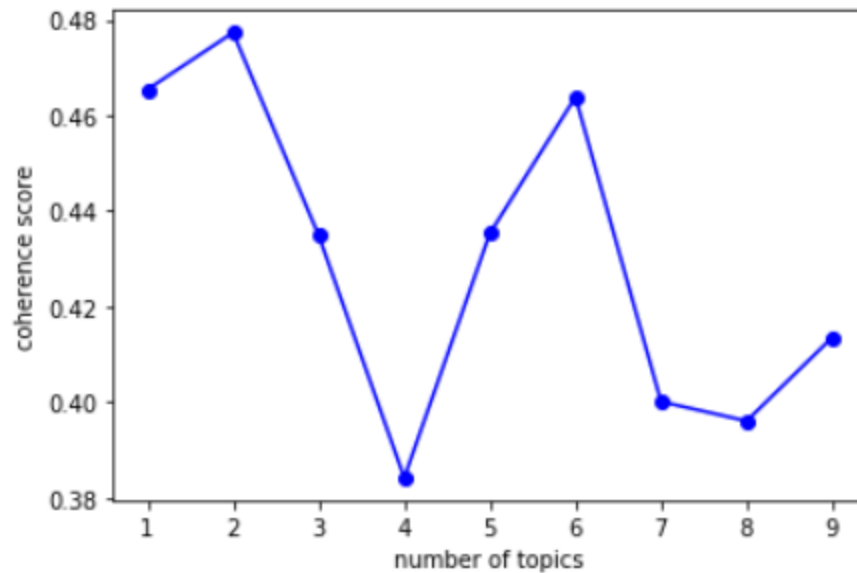


- Perplexity score
 - 확률 모델이 결과를 얼마나 정확하게 예측하는지 판단
 - 낮을수록 정확하게 예측
 - 하지만 낮다고 해서, 결과가 해석에 용이하다는 의미는 아님

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 토픽 수에 따른 일관성 점수(coherence socre) 그래프 확인

```
plt.plot(range(1,10),coherence_score,'b',marker='o')  
plt.xlabel("number of topics")  
plt.ylabel("coherence score") #일관성  
plt.show()
```



- Coherence score
 - 토픽이 얼마나 의미론적으로 일관성 있는지 판단
 - 높을수록 의미론적 일관성 높음

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 토픽 모델링 진행

```
#최적의 토픽수로 토픽 모델링 진행->topic 2개로 확정.  
noun_lda=Lda(corpus, num_topics=2, id2word=noun_dic, passes=15, random_state=0)  
topics=noun_lda.print_topics(num_words=5) #토픽별 5 단어씩 출력  
for topic in topics: #num_topics=20이므로 2개로 압축된 토픽을 각각 출력.  
    print(topic)
```

```
(0, '0.030*"보험" + 0.022*"상품" + 0.016*"가입" + 0.016*"저축" + 0.015*"준비"')  
(1, '0.023*"계좌" + 0.020*"퇴직" + 0.017*"펀드" + 0.016*"저축" + 0.016*"투자"')
```

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 토픽 모델링 결과 확인

```
# import pyLDAvis.gensim , gensim 3.9.0이하버전
# vis=pyLDAvis.gensim.prepare(noun_lda, corpus, noun_dic), gensim 3.9.0이하버전

#gensim version 3.9.0이상 시 적용
import pyLDAvis
import pyLDAvis.gensim_models #prepare함수를 쓰기위한 라이브러리 임포트
pyLDAvis.enable_notebook() # 주피터노트북에서 사용시 필요
lda_viz = pyLDAvis.gensim_models.prepare(noun_lda, corpus, noun_dic)

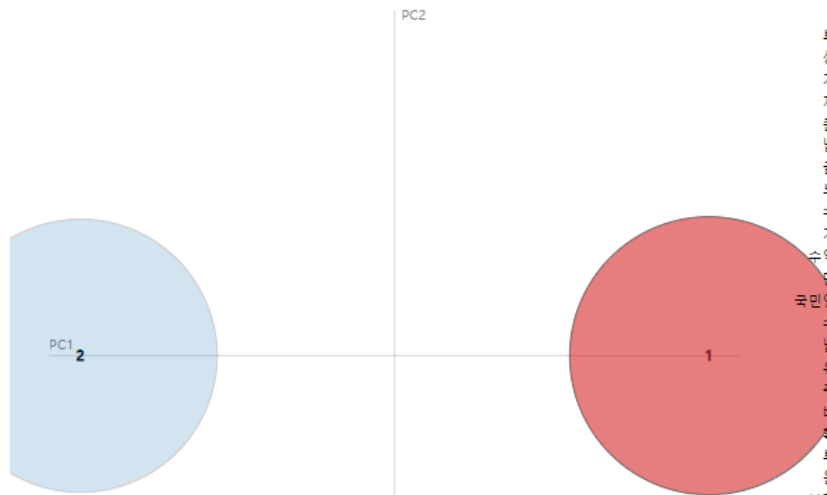
# vis=pyLDAvis.gensim.prepare(noun_lda, corpus, noun_dic)
# vis=gensim_models.prepare(noun_lda, corpus, noun_dic)
pyLDAvis.display(lda_viz)
```

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 토픽 모델링 결과

Selected Topic: Previous Topic Next Topic Clear Topic

Intertopic Distance Map (via multidimensional scaling)



Marginal topic distribution

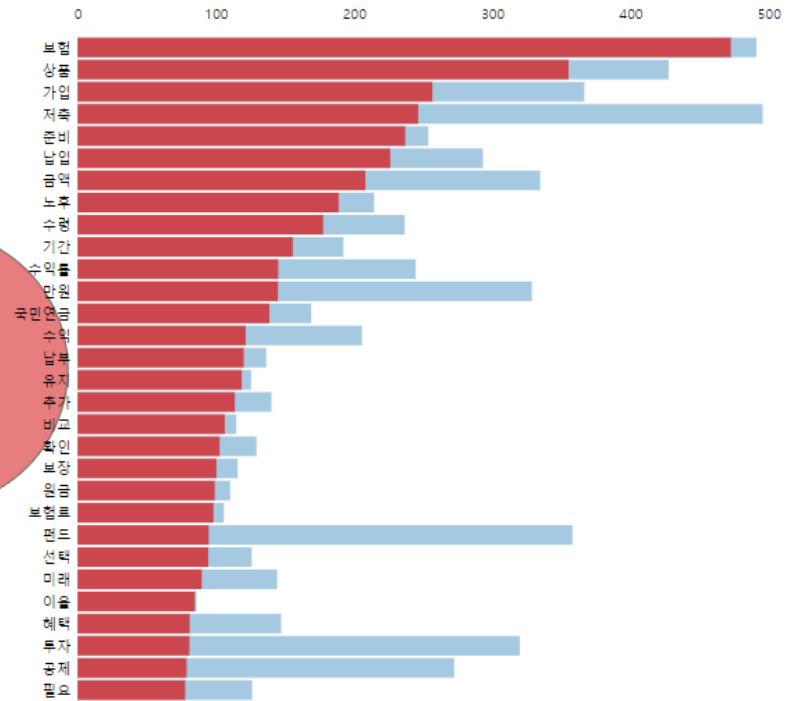


Slide to adjust relevance metric:(2)

$\lambda = 1$

0.0 0.2 0.4 0.6 0.8 1

Top-30 Most Relevant Terms for Topic 1 (50.9% of tokens)



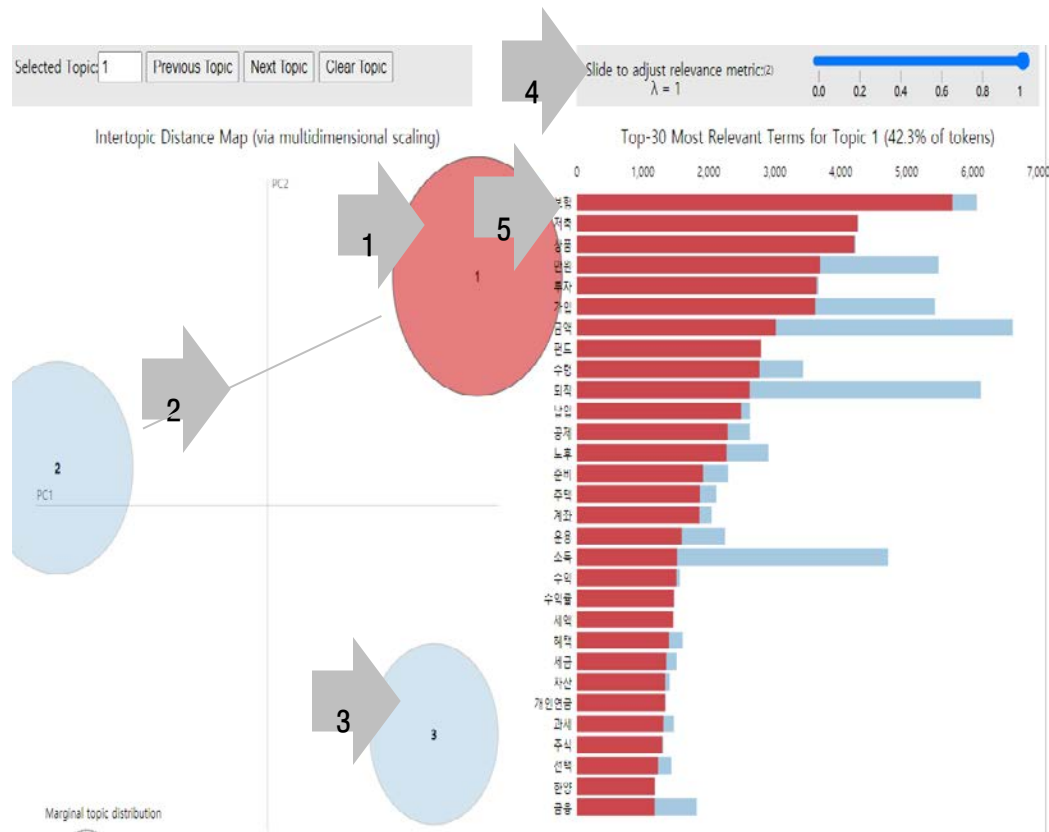
Overall term frequency

Estimated term frequency within the selected topic

1. saliency(term, w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et al. (2012)
2. relevance(term, w | topic t) = $\lambda * p(w | t) + (1 - \lambda) * p(w | t)/p(w)$; see Sievert & Shirley (2014)

LDA(Latent Dirichlet Allocation) 토픽모델링

■ LDA Topic Modeling 해석 방법



1. 토픽 선택

토픽분포도에서 토픽을 클릭하거나 토픽번호를 'Selected Topic'에 직접 입력하여 토픽을 선택하면 각 토픽을 구성하는 30개의 단어 확인 가능

2. 토픽 간의 거리

토픽 간의 거리가 멀 수록 판별 타당도가 높고 주제가 뚜렷하게 구분됨 & 토픽 간의 거리가 가깝거나 겹쳐져 있다면 판별 타당도가 낮음으로 비슷한 주제를 나타냄

3. 토픽의 크기

토픽 원의 크기가 클 수록 높은 빈도수의 단어들로 구성됨
가장 큰 원을 메인 토픽으로 해석 가능

4. λ (람다) 값 설정

λ (람다) 값을 조절하여 토픽을 구성하는 단어의 출현 조건을 설정 가능. λ 값이 낮을수록 각 토픽을 구성하는 단어가 뚜렷해지지만 비교적 빈도가 낮은 단어들로 구성됨

5. 토픽 구성 단어

토픽을 구성하는 단어들을 확인할 수 있으며, 파란막대 그래프는 전체 단어의 빈도를, 빨간막대그래프는 해당 토픽에서의 빈도를 보여줌

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 데이터프레임내 토픽번호 삽입

```
noun_dtm = [noun_dic.doc2bow(text) for text in data['token_noun']]
ldamodel=Lda(noun_dtm, num_topics=3, id2word=noun_dic, passes=10,random_state=0)

topics = []
for i in range(len(noun_dtm)):
    prop_sort=[]
    topic_sort=[]
    for topic , prop in ldamodel.get_document_topics(noun_dtm)[i]: #각 문서마다 토픽 별로 해당할 확률이 부여됨
        prop_sort.append(prop)
        topic_sort.append(topic)
    topics.append(topic_sort[prop_sort.index(max(prop_sort))]) #확률 중 가장 높은 확률의 토픽을 해당 문서의 토픽으로 지정

data['topic'] = topics #모든 문서의 토픽을 데이터프레임에 추가
```

LDA(Latent Dirichlet Allocation) 토픽모델링

■ 파일 저장 및 불러오기

```
f = open(file_path + "topic_doc.pkl", "wb")
pickle.dump(data, f) #저장
f.close()
```

```
f = open(file_path + "topic_doc.pkl", "rb")
aa = pickle.load(f) #불러오기
f.close()
##여기부터 시작.
```

```
#토픽별 데이터프레임 분류
topic0 = aa[aa['topic']==0]
topic1 = aa[aa['topic']==1]
#topic0
```

topic1

	title	doc	url	ch	ch2	token_list_pos	token_noun	topic
0	연금계좌관련 문의드립니다.	안녕하세요 주린이 개월차입니미국민연금에 퇴직연금이 있어서 그동안 개인연금에 관심을...	https://cafe.naver.com/likeusstock/158991?art=...	naver	cafe	[(안녕하세요, NNP), (주리, VV), (L, ETM), (이, NNP), (...	[개월, 자입, 국민연금, 퇴직, 개인연금, 관심, 두지, 개인연금, 나스닥, 구매...	1
2	퇴직금계산방법과 지급기준, 퇴직 연금제도 확인해보세요!	안녕하세요오늘은 퇴직금계산방법과 지급 기준을 알아보려구요외로 많은 분들이 퇴직금계...	https://cafe.naver.com/dokchi/10707550?art=ZXh...	naver	cafe	[(안녕하세요, NNP), (오늘, NNP), (은, JX), (퇴직금, NNP),...	[퇴직금, 계산, 방법, 지급, 기준, 퇴직금, 계산, 방법, 퇴직금, 계산, 방법...	1
3	퇴직연금을 잘 굴러 보세요. 대박수익 가능	전 제 개인돈은 주식을 하지만 가계자금은 주로 펀드 투자로 운용하는데요 그러다보니 ...	https://cafe.naver.com/vilab/176085?art=ZXh0ZX...	naver	cafe	[(전, MM), (제, XPN), (개인, NNG), (돈, NNG), (은, J,...	[개인, 주식, 가계, 자금, 펀드, 투자, 은용, 퇴직, 펀드, 근속, 기간, 중...	1
7	퇴직연금 납입이요....	아래 내용이 궁금합니다 회원님들의 지식을 나누어 주세요 회사에서 형 퇴직연금...	https://cafe.naver.com/serplove/627192?art=ZXh...	naver	cafe	[(아래, NNG), (내용, NNG), (이, JKS), (궁금, XR), (하,...	[회원, 지식, 회사, 퇴직, 가입, 해분, 퇴직, 퇴직, 계좌, 이자]	1
11	연금저축, ISA 자이점에 대해서 궁금한게 있습니다.	형님 삼촌들 모두 즐거운 주말저녁 보내고 계신지요연금저축 의 자이점에 대해 궁금한 ...	https://cafe.naver.com/vilab/176196?art=ZXh0ZX...	naver	cafe	[(형님, NNG), (삼촌, NNG), (를, XSN), (모두, MAG), (즐...	[형님, 삼촌, 주말, 저녁, 신지, 저축, 자이점, 질문, 미국, 미국, 자이나...	1
...
211	2021년 공무원연금 인상을 확정	년 공무원연금 인상은 로 확정 되었습니다와 인상은 퇴직자에게만 적용되는 것이고...	http://cafe.daum.net/teachers119/f1na/2?q=%EC%...	daum	cafe	[(년, NNB), (공무원, NNP), (연금, NNP), (인상을, NNG), (...	[공무원, 인상을, 확정, 인상을, 퇴직자, 적용, 재직자, 봉급, 인상을, 적용...	1
212	Re: 귀국 준비 중 후생연금	연금환급제도 연금환급제도 탈퇴일시금이란 탈퇴일시금 환급신청이란일본에서 연금을...	http://cafe.daum.net/osakalife/2xAE/89957?q=%E...	daum	cafe	[(연금, NNG), (환급, NNG), (제도, NNG), (연금, NNG), (...	[환급, 제도, 환급, 제도, 탈퇴, 일시, 환급, 일시, 환급, 신청, 이란, 일...	1

토픽별 워드클라우드

- 문서의 키워드, 개념 등을 직관적으로 파악할 수 있도록 핵심단어를 시각적으로 돋보이게 하는 기법
- 단어의 빈도에 따라 글자의 크기가 다르게 표현되기 때문에 어떤 단어가 얼마나 많이 사용됐는지 한눈에 파악 가능

토픽별 워드 클라우드

■ 토픽별 명사 확인

```
topic0_tn = topic0['token_noun']
topic1_tn = topic1['token_noun']
#topic0_tn
```

```
topic1_tn
```

```
0   [개월, 차입, 국민연금, 퇴직, 개인연금, 관심, 두지, 개인연금, 나스닥, 구매...
2   [퇴직금, 계산, 방법, 지급, 기준, 퇴직금, 계산, 방법, 퇴직금, 계산, 방법...
3   [개인, 주식, 가계, 자금, 펀드, 투자, 운용, 퇴직, 펀드, 근속, 기간, 중...
7       [회원, 지식, 회사, 퇴직, 가입, 해분, 퇴직, 퇴직, 계좌, 이체]
11  [형님, 삼촌, 주말, 저녁, 신지, 저축, 차이점, 질문, 미국, 미국, 차이나,...
...
211 [공무원, 인상률, 확정, 인상률, 퇴직자, 적용, 재직자, 봉급, 인상률, 적용,...
212 [환급, 제도, 환급, 제도, 탈퇴, 일시, 탈퇴, 일시, 환급, 신청, 이란, 일...
213 [주택, 상속자, 아파트, 정액, 후후, 박형, 종류, 정액, 지급, 평생, 고전,...
214 [개인연금, 퇴직, 퇴직, 불입, 저축, 개인연금, 퇴직, 분할, 필요, 하시나, ...
215 [회원, 추천, 해주시, 마법, 공무원, 쥐꼬리, 공무원, 개인연금, 저축, 종합,...
Name: token_noun, Length: 110, dtype: object
```

■ 토픽별 명사리스트 확보

```
import itertools
topic0_nlist = list(itertools.chain(*topic0_tn)) #리스트 연패킹
topic1_nlist = list(itertools.chain(*topic1_tn)) #리스트 연패킹
```

```
topic1_nlist
```

```
['개월',
 '차입',
 '국민연금',
 '퇴직',
 '개인연금',
 '관심',
 '두지',
 '개인연금',
 '나스닥',
 '구매',
 '가입',
 '계좌',
 '개설',
 '연간',
 '한도',
```

토픽별 워드 클라우드

■ 상위 빈도 단어 확인 (50개씩)

- collections

- 데이터의 개수를 셀 때 유용한 라이브러리

- count.most_common(k)

- 데이터의 개수가 많은 순으로 정렬해 k개를 반환

```
from collections import Counter #단어들을 쉽게 집계하기 위해서 사용
count = Counter(topic0_nlist)
hund0 = dict(count.most_common(50)) # 상위 50개 출력하기
hund0
```

```
count = Counter(topic1_nlist)
hund1 = dict(count.most_common(50)) # 상위 50개 출력하기
hund1
```

```
{'보험': 517,
'상품': 369,
'가입': 271,
'저축': 268,
'준비': 241,
'납입': 234,
'금액': 223,
'노후': 192,
'수령': 191,
'기간': 164,
'만원': 158,
'국민연금': 149,
'수익률': 148,
'수익': 130,
'납부': 130,
'유지': 128,
'추가': 121,
'비교': 113,
'확인': 111,
'보장': 109,
'펀드': 107,
'보험료': 107,
'계좌': 364,
'퇴직': 316,
'펀드': 286,
'저축': 279,
'투자': 256,
'소득': 230,
'공제': 213,
'만원': 204,
'금액': 147,
'가입': 135,
'세액': 134,
'수익률': 122,
'자산': 114,
'상품': 106,
'주식': 102,
'주택': 100,
'운용': 99,
'신청': 99,
'수익': 97,
```

토픽별 워드 클라우드

■ 토픽별 워드클라우드 그리기_토픽0 & 토픽 1

```
# del(hund0['?'])    #워드클라우드로 그릴 딕셔너리 값에서 삭제할 단어 지정
from wordcloud import WordCloud
import matplotlib.pyplot as plt

#디스플레이 설정
%matplotlib inline

wordcloud = WordCloud(font_path = file_path + 'Nanum_Gothic\NanumGothic-Bold.ttf', background_color='white', colormap = "Accent",
                      width=600, height=400).generate_from_frequencies(hund0)

plt.figure(figsize=(6,4)) #width, height비율은 유지하면서 보여지는 크기 지정
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

```
# del(hund1['?'])    #워드클라우드로 그릴 딕셔너리 값에서 삭제할 단어 지정
from wordcloud import WordCloud
import matplotlib.pyplot as plt

#디스플레이 설정
%matplotlib inline

wordcloud = WordCloud(font_path = file_path + 'Nanum_Gothic\NanumGothic-Bold.ttf', background_color='white', colormap = "Accent",
                      width=600, height=400).generate_from_frequencies(hund1)

plt.figure(figsize=(6,4)) #width, height비율은 유지하면서 보여지는 크기 지정
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

토크별 워드 클라우드

Topic 0



Topic 1

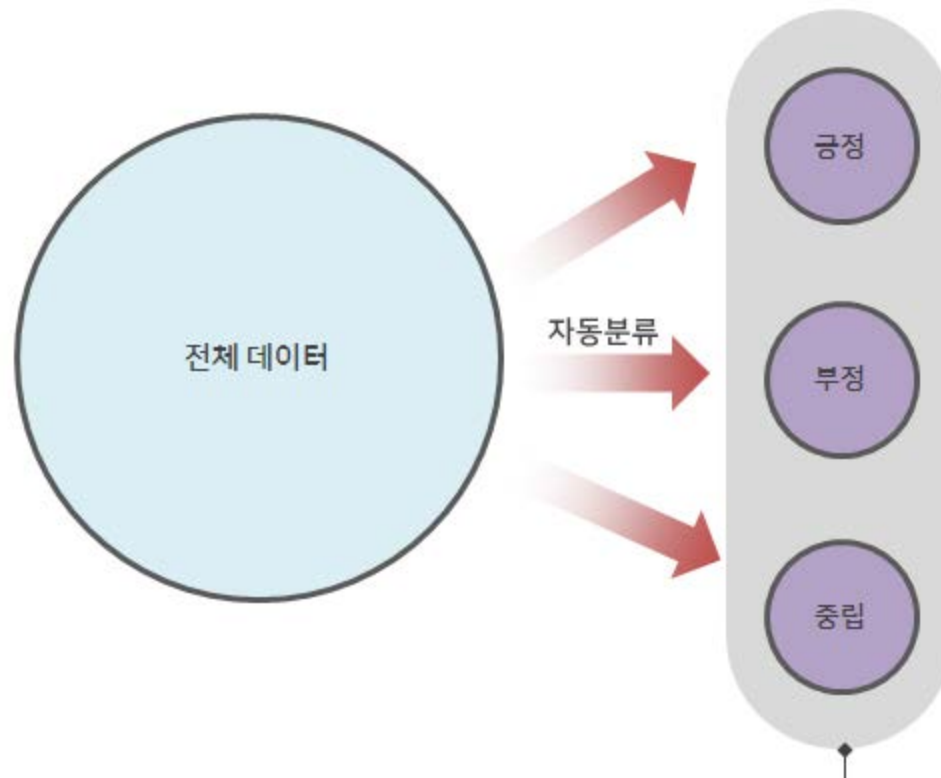


Sentiment Analysis

- 가장 대표적인 토픽모델링 기법
- 다수의 문서에서 잠재적으로 의미 있는 토픽을 발견하는, 절차적 확률 분포 모델링

Sentiment Analysis

- 감성 분석(Sentiment Analysis)은 '오피니언 마이닝(Opinion Mining)'으로도 불리는데, 이는 텍스트에 나타난 사람들의 태도, 의견, 성향과 같은 주관적인 데이터를 분석하는 자연어 처리 기술



Sentiment Analysis

초기 데이터

크롤링을 통해 확보한 데이터를 전처리 과정을 통해
핸들링하기 쉽게 변환해 줌

형태소 분석

자연어 처리에서 어떤 대상 어절을 최소의 의미 단위
인 형태소로 분리한 뒤, 분리된 형태소의 기본형 및
품사 정보를 추출

긍/부정 판별

감성사전을 이용해 긍정, 부정, 중립 판별

■ 감성분석 작업

- 1) 긍/부정의 대상이 되는 단어 또는 개체 추출
- 2) 추출된 단어속에 나타난 의견, 평가, 태도 등의 특징적 양상을 정량화된 데이터로 제시
- 3) 텍스트간의 비교우위를 밝혀 상대적 비교를 함

■ 감성분석 종류

사전기반 감성분석

기계학습기반 감성분석

■ 사전기반 감성분석

감성사전을 활용

감성사전 : 단어와 감성지표(긍정, 부정 정도)를 대응시켜 놓은 자료

자주쓰이는 감성사전 예시

- 1) AFINN : Finn Arup Nielsen이 2009 ~ 2011년에 직접 수집한 감성 어휘들에 대해 -5~+5의 점수를 부여한 사전. 총 2,477개의 감성어가 수록
- 2) EmoLex : 단어들을 긍정, 부정 뿐만 아니라 공포, 기대, 신뢰, 놀람, 슬픔, 기쁨, 역겨움과 같은 8가지 감정으로 분류
인간의 정서 정보를 더욱 풍부하게 반영함. 총 14,182개의 감성어 보유
- 3) Bing Luu lexicon : 감성어들을 긍정, 부정으로만 분류하며 점수지표 없음.
총 6,800여개의 감성어가 있으며 지속적으로 업데이트 되고 있음
- 4) SentiwordNet : 긍정, 부정, 중립으로 분류, 파이썬 NLTK패키지에 있어 사용 편리

한국어 감성분석의 경우 한국어감성사전을 사용

[한국어 감성사전]

KOSAC : <http://word.snu.ac.kr/kosac/index.php>

EmoLex : 14,182개의 단어에 대해 영어 뿐만 아니라 한국어 포함
105개국 언어에 대한 감성 사전을 제공

Sentiment Analysis

■ KOSAC감성사전 불러오기

감성사전을 다루기 쉽게 정제한 파일

```
import pandas as pd
senti2 = pd.read_excel(file_path + 'polarity2.xlsx', engine='openpyxl')
senti2
```

xlsx 파일을 데이터프레임 형식으로 만들기 위한 필수 함수

Unnamed: 0		ngram	freq	COMP	NEG	NEUT	None	POS	max.value	max.prop	Unnamed: 10
0	0	가/JKS	1	0.0	0.0	0.0	0.0	1.0	POS	1.0	NaN
1	1	가/JKS;있/VV	1	0.0	0.0	0.0	0.0	1.0	POS	1.0	NaN
2	2	가/JKS;있/VV;있/EP	1	0.0	0.0	0.0	0.0	1.0	POS	1.0	NaN
3	3	가/VV	3	0.0	0.0	0.0	0.0	1.0	POS	1.0	NaN
4	4	가/VV;ㄴ다/EF	1	0.0	0.0	0.0	0.0	1.0	POS	1.0	NaN
...
16357	16357	힘들/VA;ㄹ/ETM;것/NNB	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0	NaN
16358	16358	힘들/VA;ㄹ/ETM;때/NNG	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0	NaN
16359	16359	힘자/VA	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0	NaN
16360	16360	힘자/VA;ㄴ/ETM	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0	NaN
16361	16361	힘자/VA;ㄴ/ETM;붓/NNG	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0	NaN

16362 rows × 11 columns

Sentiment Analysis

■ KOSAC감성사전에서 필요한 부분만 불러오기_ngram, max_value

#감성사전에서 이용할 부분만 가져오기

```
senti_data = pd.DataFrame()  
senti_data = senti_data.assign(ngram = senti2['ngram'])  
senti_data = senti_data.assign(value = senti2['max.value'])  
senti_data
```

```
for i in range(len(senti2)):  
    senti_data['ngram'][i] = senti_data['ngram'][i].replace(':', ',') ngram의 ':'을 ','으로 변경
```

```
for i in range(len(senti2)):  
    senti_data['ngram'][i] = senti_data['ngram'][i].replace('/', ',') ngram의 '/'를 ','으로 변경
```

```
for i in range(len(senti2)):  
    senti_data['ngram'][i] = senti_data['ngram'][i].split(sep=',') # 샘플단위로 단어 나누기
```

```
senti_data
```

Sentiment Analysis

■ Ngram과 value의 결과

	ngram	value
0	[가, JKS]	POS
1	[가, JKS, 있, VV]	POS
2	[가, JKS, 있, VV, 었, EP]	POS
3	[가, VV]	POS
4	[가, VV, ㄴ 다, EF]	POS
...
16357	[힘들, VA, ㄹ, ETM, 것, NNB]	NEG
16358	[힘들, VA, ㄹ, ETM, 때, NNG]	NEG
16359	[힘차, VA]	NEG
16360	[힘차, VA, ㄴ, ETM]	NEG
16361	[힘차, VA, ㄴ, ETM, 붓, NNG]	NEG

16362 rows × 2 columns

Sentiment Analysis

■ 감성사전을 형태소분석을 위한 excel데이터로 변환

- ngram을 연금데이터내의 단어와 비교하기 용이한 상태로 분리

```
senti_list = []

for i in range(len(senti_data)):
    en_list = []

    ng = senti_data['ngram'][i]
    ng = tuple(ng) #
    if len(ng) == 2:
        a = tuple(ng[0:2]) # list는 대괄호-수정, 삭제 생성 가능, tuple은 소괄호 - 수정
        senti_list.append(a)
    elif len(ng) == 4:
        a = tuple(ng[0:2]) #tuple내에 list가 있는 경우는 리스트요소값의 수정이 가능
        b = tuple(ng[2:4])
        en_list.append(a)
        en_list.append(b)
        senti_list.append(en_list)
    else :
        a = tuple(ng[0:2])
        b = tuple(ng[2:4])
        c = tuple(ng[4:6])
        en_list.append(a)
        en_list.append(b)
        en_list.append(c)
        senti_list.append(en_list)
```

[(단어1, 형태소1), (단어2, 형태소2), (단어3, 형태소3)]의 형태로 분리

Sentiment Analysis

■ 분리된 결과

| senti_list

```
[('가', 'JKS'),  
 [('가', 'JKS'), ('있', 'VV')],  
 [('가', 'JKS'), ('있', 'VV'), ('엇', 'EP')],  
 ('가', 'VV'),  
 [('가', 'VV'), ('나다', 'EF')],  
 ('가', 'JKC'),  
 [('가', 'JKC'), ('되', 'VV')],  
 [('가', 'JKC'), ('되', 'VV'), ('ㄴ', 'ETM')],  
 [('가', 'JKC'), ('되', 'VV'), ('ㄹ', 'ETM')],  
 [('가', 'JKC'), ('되', 'VV'), ('어', 'EC')],  
 [('가', 'JKC'), ('되', 'VV'), ('어야지요', 'EF')],  
 [('가', 'JKC'), ('아니', 'VCN')],  
 [('가', 'JKC'), ('아니', 'VCN'), ('ㄴ가', 'EC')],  
 [('가', 'JKC'), ('아니', 'VCN'), ('면', 'EC')],  
 [('가', 'JKC'), ('아니', 'VCN'), ('ㄴ니다', 'EF')],  
 ('가', 'JKS'),  
 [('가', 'JKS'), ('가능', 'NNG')],  
 [('가', 'JKS'), ('가능', 'NNG'), ('하', 'XSA')],  
 [('가', 'JKS'), ('가장', 'MAG')],  
 [('가', 'JKS'), ('가장', 'MAG'), ('만일', 'MAG')]
```

Sentiment Analysis

■ 긍정/중립/부정(POS /NEU/NEG)을 숫자(1, 0, -1)으로 define

```
#senti_list에  
#ngram 긍부정에 따른 score(-1,0,1) 부여  
score = []  
for i in range(len(senti_data)):  
    if senti_data['value'][i] == 'POS':  
        score.append(1)  
    elif senti_data['value'][i] == 'NEG':  
        score.append(-1)  
    else :  
        score.append(0)
```

```
| #데이터프레임에 senti_list와 score 생성  
senti_data = pd.DataFrame()  
senti_data = senti_data.assign(ngram = senti2['ngram'])  
senti_data = senti_data.assign(value = senti2['max.value'])  
senti_data = senti_data.assign(gramgram = senti_list)  
senti_data = senti_data.assign(score = score)  
senti_data
```


Sentiment Analysis

	ngram	value	gramgram	score
0	가/JKS	POS	(가, JKS)	1
1	가/JKS;있/VV	POS	[(가, JKS), (있, VV)]	1
2	가/JKS;있/VV;엮/EP	POS	[(가, JKS), (있, VV), (엮, EP)]	1
3	가/VV	POS	(가, VV)	1
4	가/VV;ㄴ 다/EF	POS	[(가, VV), (ㄴ 다, EF)]	1
...
16357	힘들/VA;ㄹ/ETM;것/NNB	NEG	[(힘들, VA), (ㄹ, ETM), (것, NNB)]	-1
16358	힘들/VA;ㄹ/ETM;때/NNG	NEG	[(힘들, VA), (ㄹ, ETM), (때, NNG)]	-1
16359	힘차/VA	NEG	(힘차, VA)	-1
16360	힘차/VA;ㄴ/ETM	NEG	[(힘차, VA), (ㄴ, ETM)]	-1
16361	힘차/VA;ㄴ/ETM;붓/NNG	NEG	[(힘차, VA), (ㄴ, ETM), (붓, NNG)]	-1

16362 rows × 4 columns

Sentiment Analysis

■ 형태소가 1개인 것 중에 글자가 한글자인 것(예: 조사 등) 을 제거

```
| #senti_list(gramgram)에서 형태소가 하나이면서 해당 문자의 길이가 한글자(예: 조사 등) 때, 인덱스를 가져옴
count = []
for i in range(len(senti_list)):
    if len(senti_list[i]) == 2:
        if len(senti_list[i][0]) == 1:
            count.append(i)
```

```
| senti_data2 = senti_data.drop(count) #데이터프레임에서 가져온 인덱스에 해당하는 행 제거 #두번 실행시 오류!
senti_data = senti_data2.reset_index(drop=True) #인덱스 초기화
senti_data
```

Sentiment Analysis

■ 제외된 결과 (16,363 -> 15,835개로 감소)

	ngram	value	gramgram	score
0	가/JKS;있/VV	POS	[(가, JKS), (있, VV)]	1
1	가/JKS;있/VV;있/EP	POS	[(가, JKS), (있, VV), (있, EP)]	1
2	가/VV;ㄴ 다/EF	POS	[(가, VV), (ㄴ 다, EF)]	1
3	가/JKC;되/VV	NEG	[(가, JKC), (되, VV)]	-1
4	가/JKC;되/VV;ㄴ /ETM	NEUT	[(가, JKC), (되, VV), (ㄴ, ETM)]	0
...
15830	힘들/VA;ㄹ/ETM;것/NNB	NEG	[(힘들, VA), (ㄹ, ETM), (것, NNB)]	-1
15831	힘들/VA;ㄹ/ETM;때/NNG	NEG	[(힘들, VA), (ㄹ, ETM), (때, NNG)]	-1
15832	힘차/VA	NEG	(힘차, VA)	-1
15833	힘차/VA;ㄴ/ETM	NEG	[(힘차, VA), (ㄴ, ETM)]	-1
15834	힘차/VA;ㄴ/ETM;붓/NNG	NEG	[(힘차, VA), (ㄴ, ETM), (붓, NNG)]	-1

15835 rows × 4 columns

제외됨

	ngram	value	gramgram	score
0	가/JKS	POS	(가, JKS)	1
1	가/JKS;있/VV	POS	[(가, JKS), (있, VV)]	1
2	가/JKS;있/VV;있/EP	POS	[(가, JKS), (있, VV), (있, EP)]	1
3	가/VV	POS	(가, VV)	1
4	가/VV;ㄴ 다/EF	POS	[(가, VV), (ㄴ 다, EF)]	1
...
16357	힘들/VA;ㄹ/ETM;것/NNB	NEG	[(힘들, VA), (ㄹ, ETM), (것, NNB)]	-1
16358	힘들/VA;ㄹ/ETM;때/NNG	NEG	[(힘들, VA), (ㄹ, ETM), (때, NNG)]	-1
16359	힘차/VA	NEG	(힘차, VA)	-1
16360	힘차/VA;ㄴ/ETM	NEG	[(힘차, VA), (ㄴ, ETM)]	-1
16361	힘차/VA;ㄴ/ETM;붓/NNG	NEG	[(힘차, VA), (ㄴ, ETM), (붓, NNG)]	-1

16362 rows × 4 columns

Sentiment Analysis

■ 정리된 형태소 데이터를 저장하고 불러오기

```
import pickle
f = open(file_path + "gram_list_pre.pkl", "wb") # 지금까지 정리된 형태소데이터를 저장
pickle.dump(senti_data, f)
f.close()
```

```
f = open(file_path + "topic_doc.pkl", "rb")
docs = pickle.load(f) #불러오기
f.close()
docs
```

	title	doc	url	ch	ch2	token_list_pos	token_noun	topic
0	연금계좌관련 문의 드립니다.	안녕하세요 주린이 개월차입니다국민연금에 퇴직연금이 있어서 그동안 개인연금은 관심을 ...	https://cafe.naver.com/likeusstock/158991?art=...	naver	cafe	[(안녕하세요, NNP), (주리, VV), (ㄴ, ETM), (이, NNP), (...	[개월, 차입, 국민연금, 퇴직, 개인연금, 관심, 두지, 개인연금, 나스닥, 구매...	1
1	국민연금 추납과 소득공제	와이프가 결혼 전 일을 하다가 결혼 후 육아로 일을 쉬고 있습니다 한의사라 언제가 ...	https://cafe.naver.com/hotellife/1604742?art=Z...	naver	cafe	[(와이프, NNG), (가, JKS), (결혼, NNG), (전, MM), (일,...	[와이프, 결혼, 결혼, 육아, 한의사, 국민연금, 제한, 소식, 임의, 가입, 시...	0

Sentiment Analysis

■ 감성분석을 위한 문서별 단어의 긍·부정 평균 산출

```
1 # 각 문서별 단어의 평균을 구함 (감성분석을 위해)
def AVG(a):
    try :
        avg = sum(a) / len(a)
        return avg
    except :
        return 0
```

```
1 from tqdm.notebook import tqdm # 분석중간중간의 진행도를 표시해줄 (%fh)

total_senti = []
for j in tqdm(range(0, len(docs))):
    docu_senti = []
    for i in range(0, len(senti_data)):
        if senti_data['gramgram'][i] in docs['token_list_pos'][j]:
            docu_senti.append(senti_data['score'][i]) #평균감성값을 넣어줄 score 생성(-1, 0, 1)
    total_senti.append(AVG(docu_senti))
```

```
1 len(total_senti) #평균 score 개수 = 문서 개수
```

Sentiment Analysis

```
docs['score'] = total_senti
docs
```

0	연금계좌관련 문의드립니다.	안녕하세요 주린이 개월차입니다국민연금에 퇴직연금이 있어서 그동안 개인연금은 관심을 ...	https://cafe.naver.com/likeusstock/158991?art=...	naver	cafe	[(안녕하세요, NNP), (주리, VV), (ㄴ, ETM), (이, NNP), (...	국민연금, 퇴직, 개인연금, 관심, 두지, 개인연금, 나스닥, 구매...	1	0.148148
1	국민연금 추납과 소득공제	와이프가 결혼 전 일을 하다가 결혼 후 육아로 일을 쉬고 있습니다 한의사라 언제가 ...	https://cafe.naver.com/hotellife/1604742?art=Z...	naver	cafe	[(와이프, NNG), (가, JKS), (결혼, NNG), (전, MM), (일, ...	[와이프, 결혼, 결혼, 육아, 한의사, 국민연금, 제한, 소식, 임의, 가입, 시...	0	-0.037037
2	퇴직금계산방법과 지급기준, 퇴직연금제도 확인해보세요!	안녕하세요오늘은 퇴직금계산방법과 지급 기준을 알아보려구요의외로 많은 분들이 퇴직금계...	https://cafe.naver.com/dokchi/10707550?art=ZXh...	naver	cafe	[(안녕하세요, NNP), (오늘, NNP), (은, JX), (퇴직금, NNP), ...	[퇴직금, 계산, 방법, 지급, 기준, 퇴직금, 계산, 방법, 퇴직금, 계산, 방...	1	-0.140351

Sentiment Analysis

■ 감성스코어를 포함함 데이터 저장 / 긍부정 판별에 필요한 필드만 topic_senti로 만들기

```
| import pickle  
| f = open(file_path + "score_doc.pkl", "wb") # 평균score값 데이터 저장  
| pickle.dump(docs, f)  
| f.close()
```

```
| f = open(file_path + 'score_doc.pkl', "rb")  
| aa = pickle.load(f)  
| f.close()  
| aa
```

```
#긍부정 판별에 필요한 부분만 가져오기  
topic_senti = pd.DataFrame()  
topic_senti['text'] = aa['doc']  
topic_senti['topic_num'] = aa['topic']  
topic_senti['topic_score'] = aa['score']
```

Sentiment Analysis

■ 토픽별 긍·부정 빈도수 산출하기

```
senti_0 = [0,0,0] #토픽0의 긍정/중립/부정 개수
senti_1 = [0,0,0] #토픽1의 긍정/중립/부정 개수

# 평균score > 0.3 :긍정 / -0.3 <= 평균score <= 0.3 :중립 / 나머지 :부정

for i in range(len(topic_senti)):
    # For topic 0
    if topic_senti['topic_num'].iloc[i] == 0: #모든 문서(행)를 비교하여 토픽이 0일때
        if topic_senti['topic_score'].iloc[i] > 0.3:
            senti_0[0] = senti_0[0] + 1 # 카운트하라
        elif topic_senti['topic_score'].iloc[i] <= 0.3 and topic_senti['topic_score'].iloc[i] >= -0.3:
            senti_0[1] = senti_0[1] + 1
        else:
            senti_0[2] = senti_0[2] + 1

    # For topic 1
    else : # topic_senti['topic_num'].iloc[i] == 1:
        if topic_senti['topic_score'].iloc[i] > 0.3:
            senti_1[0] = senti_1[0] + 1
        elif topic_senti['topic_score'].iloc[i] <= 0.3 and topic_senti['topic_score'].iloc[i] >= -0.3:
            senti_1[1] = senti_1[1] + 1
        else:
            senti_1[2] = senti_1[2] + 1
```


Sentiment Analysis

■ 토픽별 긍·부정 빈도수 결과

```
print(senti_0)
print(senti_1)
```

```
[5, 97, 5]
[4, 103, 3]
```

```
senti_0 = [5,97,5]
senti_1 = [4,103,3]
```

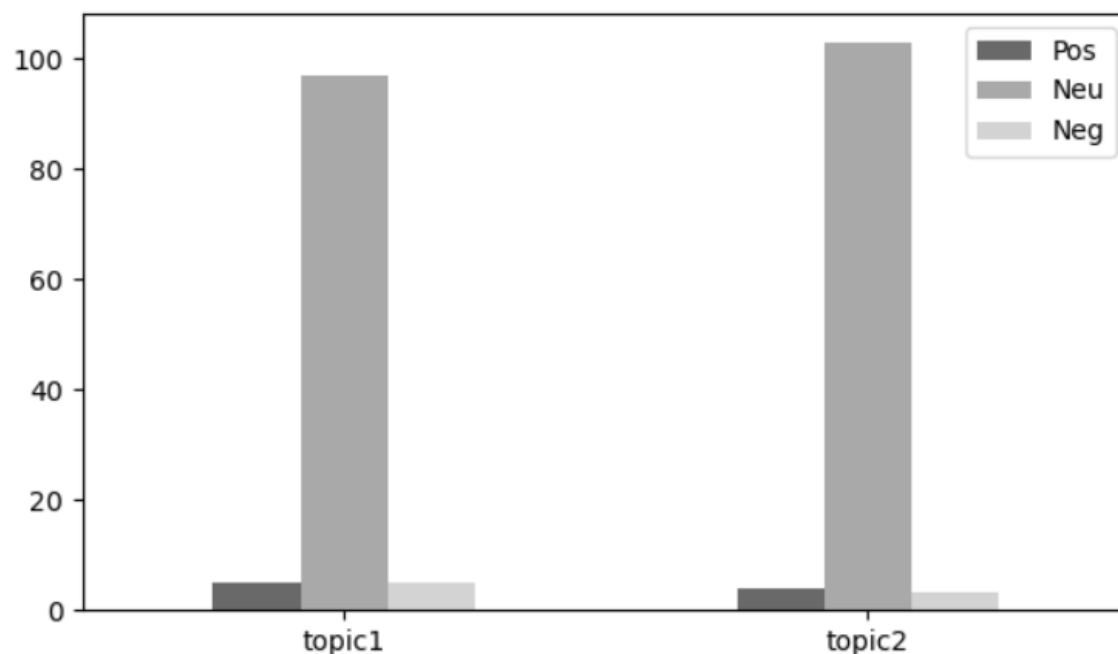
```
import pandas as pd
senti_bar = pd.DataFrame([senti_0,senti_1],
                          index=['topic1','topic2'],
                          columns=['Pos','Neu','Neg'])
senti_bar
```

	Pos	Neu	Neg
topic1	5	97	5
topic2	4	103	3

Sentiment Analysis

■ 토픽별 긍·부정 빈도의 시각화

```
ax = senti_bar.plot(kind = 'bar', color=['dimgray', 'darkgray','lightgray'],figsize = (7,4),  
                    legend = True, fontsize = 10, rot = 0) #figsize : x축, y축 박스 크기
```



Sentiment Analysis

■ 형태소 분석 실행

```
token_list = komoran.nouns(docs['doc'][0])
```

token_noun

'배상미판',
'기준',
'배제',
'확정',
'수익',
'변화',
'수령',
'확정',
'과세',
'연금',
'보험',
'가지',
'한화',
'수익률',
'최저',
'수령',
'보장',
'남성',
'만원',
'납부',

```
token_list = komoran.pos(docs['doc'][0])
```

token_list

('이', 'VCP'),
('ㄴ', 'ETM'),
('건', 'NNB'),
('처음', 'NNP'),
('해지', 'NNP'),
('신청', 'NNP'),
('을', 'JKO'),
('하', 'VV'),
('았', 'EP'),
('던', 'ETM'),
('목요일', 'NNP'),
('밤', 'NNG'),
('에', 'JKB'),
('는', 'JX'),
('만원', 'NNG'),
('정도', 'NNG'),
('로', 'JKB'),
('예상', 'NNG'),
('되', 'XSV'),

Sentiment Analysis

■ 감성사전 비교(한국어)

	ngram	freq	COMP	NEG	NEUT	None	POS	max.value	max.prop
0	가/JKS	1	0.0	0.0	0.0	0.0	1.0	POS	1.0
1	가/JKS;있/VV	1	0.0	0.0	0.0	0.0	1.0	POS	1.0
2	가/JKS;있/VV;있/EP	1	0.0	0.0	0.0	0.0	1.0	POS	1.0
3	가/VV	3	0.0	0.0	0.0	0.0	1.0	POS	1.0
4	가/VV;ㄴ다/EF	1	0.0	0.0	0.0	0.0	1.0	POS	1.0
...
16357	힘들/VA;ㄹ/ETM;것/NNB	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0
16358	힘들/VA;ㄹ/ETM;때/NNG	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0
16359	힘차/VA	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0
16360	힘차/VA;ㄴ/ETM	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0
16361	힘차/VA;ㄴ/ETM;붓/NNG	1	0.0	1.0	0.0	0.0	0.0	NEG	1.0

[Kosac 감성사전]

```
{'word': '고상한 남자', 'word_root': '고상 남자', 'polarity': '1'},
{'word': '고상한 이야기', 'word_root': '고상 이야기', 'polarity': '1'},
{'word': '고상한 인품', 'word_root': '고상 인품', 'polarity': '2'},
{'word': '고상한 품채', 'word_root': '고상 품채', 'polarity': '2'},
{'word': '고생스러운', 'word_root': '고생', 'polarity': '-2'},
{'word': '고생을', 'word_root': '고생', 'polarity': '-1'},
{'word': '고생을 격다', 'word_root': '고생 격', 'polarity': '-1'},
{'word': '고생을 하다', 'word_root': '고생 하다', 'polarity': '-1'},
{'word': '고생이나', 'word_root': '고생', 'polarity': '-1'},
{'word': '고생하거나', 'word_root': '고생', 'polarity': '-1'},
{'word': '고생하거나 두려워하여', 'word_root': '고생 두려워하', 'polarity': '-2'},
{'word': '고생하다', 'word_root': '고생', 'polarity': '-1'},
{'word': '고성능', 'word_root': '고성능', 'polarity': '2'},
{'word': '고소 공포증', 'word_root': '고소 공포증', 'polarity': '-1'},
{'word': '고약하께', 'word_root': '고약', 'polarity': '-2'},
{'word': '고약하다', 'word_root': '고약', 'polarity': '-2'},
{'word': '고약함', 'word_root': '고약', 'polarity': '-2'},
```

[KNU 한국어 감성사전]

<영화 리뷰 데이터 감성분석 사례 >

<https://romanticq.github.io/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D/text-mining-tech2/>

Kaggle data (팀별 스터디 필수)

<https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews>

KNUSAC용 권장

- KOSAC은 다루기가 어렵고 중립 감성어 비율이 과다