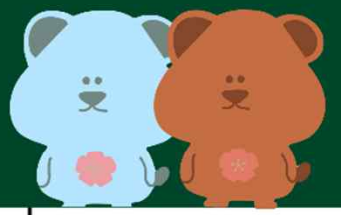
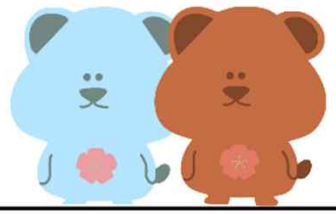


13주차 발표

소예림, 차수빈, 조혜빈

EWHA EURON

목차



EWHA EURON

01. 텍스트 분석의 이해

EWHA EURON

#1.1 텍스트 분석

#1 NLP (National Language Processing) VS. 텍스트 분석

- NLP: 머신이 인간의 언어를 이해하고 해석하는 데 더 중점을 두고 기술이 발전
- 텍스트 분석: 비정형 텍스트에서 의미 있는 정보를 추출하는 것에 좀 더 중점을 두고 기술이 발전

#2 피처 벡터화 (피처 추출)

- 텍스트를 word 기반의 다수의 피처로 추출하고 이 피처에 단어 빈도수와 같은 숫자 값을 부여하여 텍스트를 단어의 조합인 벡터 값으로 표현
- BoW (Bag of Words), Word2Vec, GloVe, FastText

#3 텍스트 분석 프로세스

- 텍스트 전처리 → 피처 벡터화 → ML 모델 수립 및 학습/예측/평가

#1.2 피처 벡터화

#1 Word2Vec

- 신경망 모델을 사용하여 텍스트로부터 단어의 연관성을 학습
- 단어의 의미에 따라 벡터 공간에 projecting
- 비슷한 의미를 가진 단어들끼리 모이고, 다른 의미를 가진 단어들끼리 멀어지도록

#2 GloVe

- Word2Vec처럼 단어의 의미에 따라 벡터 공간에 projecting
- Word2Vec과 달리 local statistics (local context information of words)에 의존하는 것이 아니라 word co-occurrence를 바탕으로 global statistics에 의존함

#3 FastText

- Modified version of Word2Vec
- Word2Vec에서는 각 단어가 BoG로 표현되지만 FastText에서는 bag of character n-gram으로 표현됨

#1.3 NLP task

#1 카카오 PORORO 분석 패키지 – 텍스트 분석 패키지

text 예제:

가수 김태연은 걸 그룹 소녀시대, 소녀시대-태티서 및 소녀시대-Oh!GG의 리더이자 메인보컬이다. 2004년 SM에서 주최한 청소년 베스트 선발 대회에서 노래짱 대상을 수상하며 SM 엔터테인먼트에 캐스팅되었다. 이후 3년간의 연습생을 거쳐 2007년 소녀시대의 멤버로 데뷔했다. 태연은 1989년 3월 9일 대한민국 전라북도 전주시 완산구에서 아버지 김종구, 어머니 김희자 사이의 1남 2녀 중 둘째로 태어났다. 가족으로는 오빠 김지웅, 여동생 김하연이 있다. 어릴 적부터 춤을 좋아했고 특히 명절 때는 친척들이 춤을 시키면 곧잘 추었다던 태연은 TV에서 보아를 보고 가수의 꿈을 갖게 되었다고 한다. 전주양지초등학교를 졸업하였고 전주양지중학교 2학년이던 2003년 SM아카데미 스타라이트 메인지방보컬과 4기에 들어가게 되면서 아버지와 함께 주말마다 전주에서 서울로 이동하며 가수의 꿈을 키웠다. 2004년에 당시 보컬 트레이너였던 더 원의 정규 2집 수록곡 〈You Bring Me Joy (Part 2)〉에 피처링으로 참여했다. 당시 만 15세였던 태연은 현재 활동하는 소속사 SM 엔터테인먼트에 들어가기 전이었다. 이후 태연은 2004년 8월에 열린 제8회 SM 청소년 베스트 선발 대회에서 노래짱 부문에 출전해 1위(대상)를 수상하였고 SM 엔터테인먼트에 정식 캐스팅되어 연습생 생활을 시작하게 되었다. 2005년 청담고등학교에 입학하였으나, 학교 측에서 연예계 활동을 용인하지 않아 전주예술고등학교 방송문화예술과로 전학하였고 2008년 졸업하면서 학교를 빛낸 공로로 공로상을 수상했다. 태연은 연습생 생활이 힘들어 숙소에서 몰래 뛰쳐나갔다가 하루 만에 다시 돌아오기도 했다고 이야기하기도 했다. 이후 SM엔터테인먼트에서 3년여의 연습생 기간을 거쳐 걸 그룹 소녀시대의 멤버로 정식 데뷔하게 되었다.

샘플 데이터 생성

```
input_text1 = """가수 김태연은 걸 그룹 소녀시대, 소녀시대-태티서 및 소녀시대-Oh!GG의 리더이자 메인보컬이다
```

EWHA EURON

#1.3 NLP task

#1 카카오 PORORO 분석 패키지 – 텍스트 분석 패키지

문서 요약:

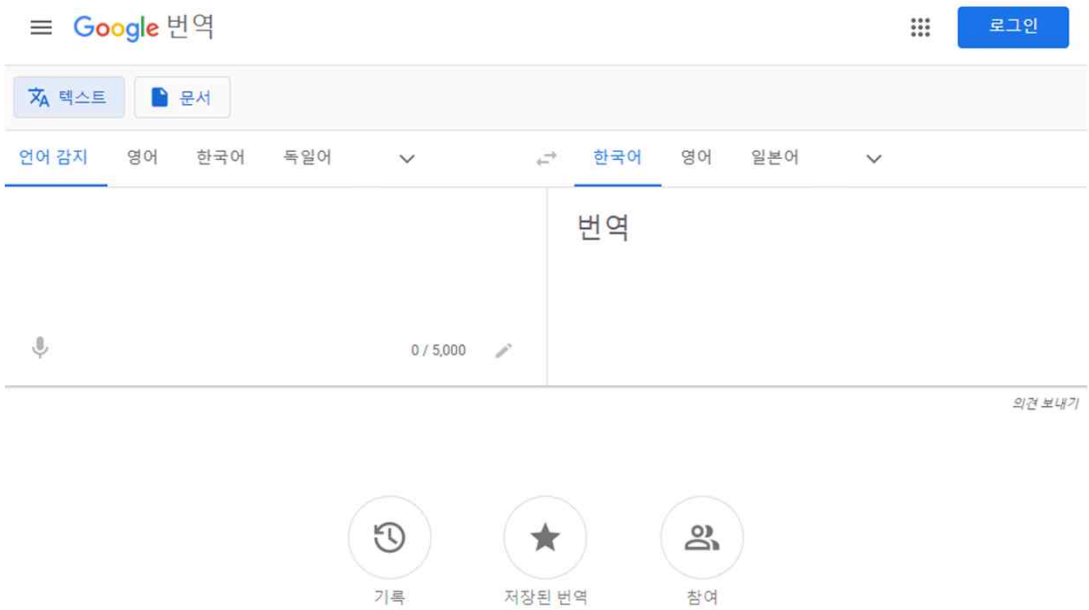
```
abs_summ = Pororo(task="text_summarization", lang="ko", model="abstractive")

abs_summ(
    input_text1,
    beam=5,
    len_penalty=0.6,
    no_repeat_ngram_size=3,
    top_k=50,
    top_p=0.7
)
```

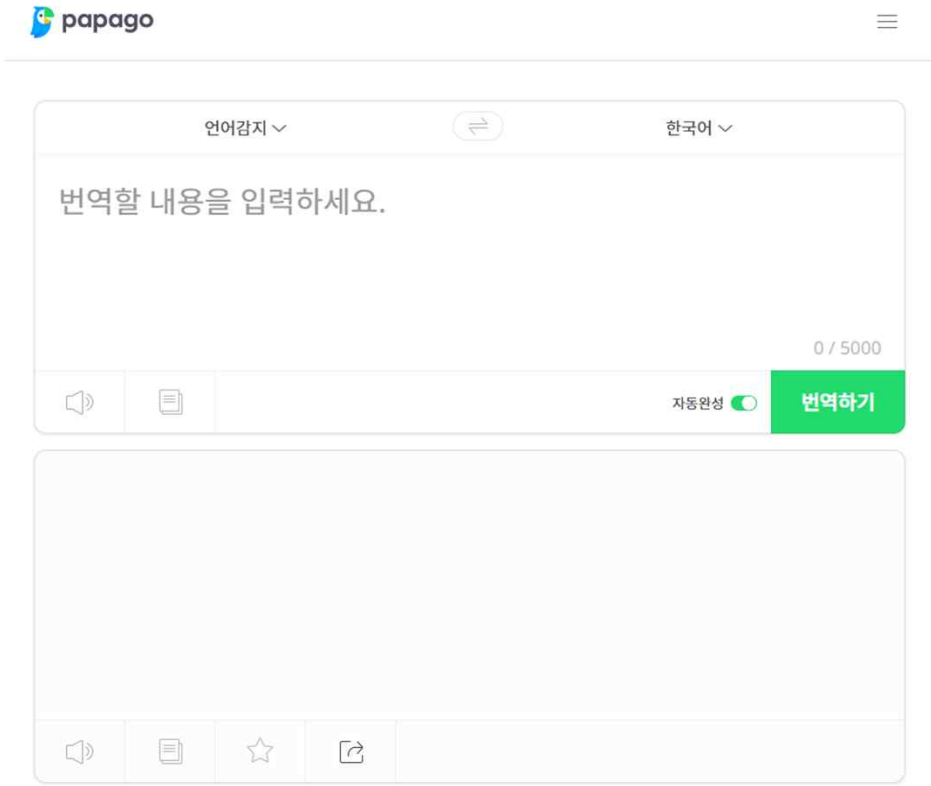
요약 결과

'가수 김태연은 2004년 SM 청소년 베스트 선발 대회에서 노래짱 대상을 수상하여 SM 엔터테인먼트에 캐스팅되어 3년간의 연습생 생활을 거쳐 2007년 소녀시대의 멤버로 데뷔했다.'

#1.3 NLP task

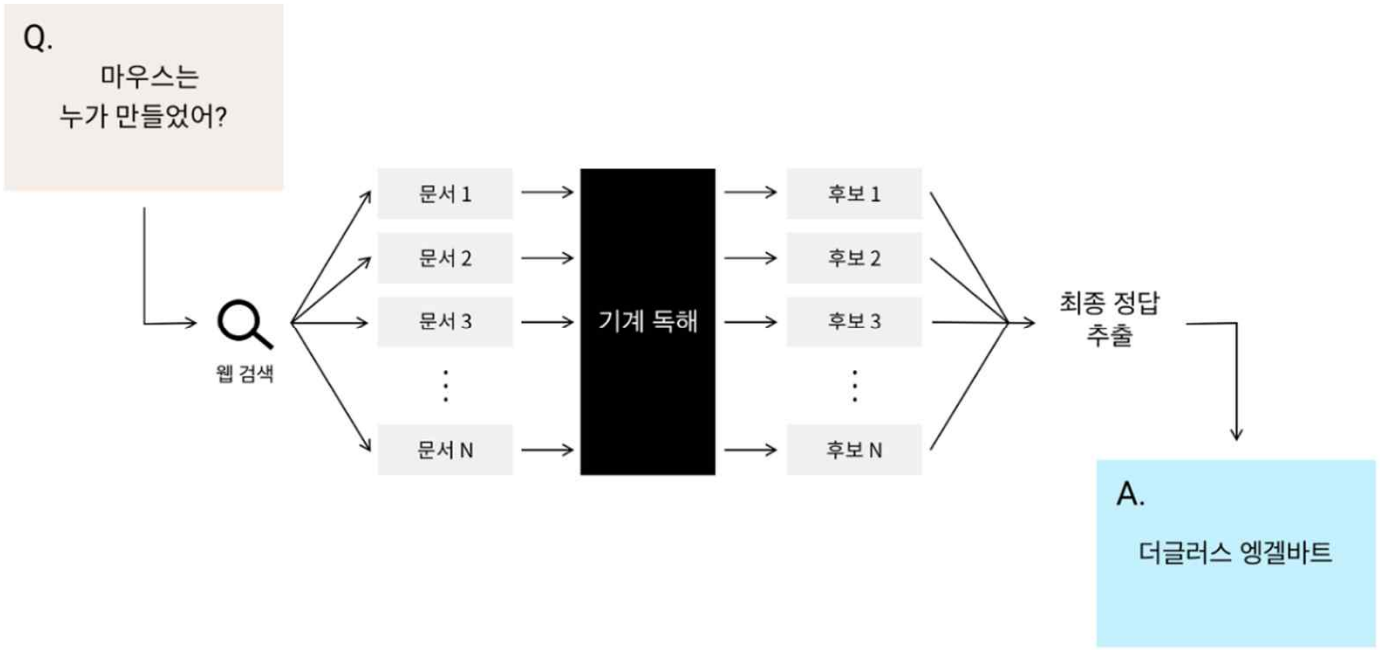
#2 기계 번역: Google 번역, 네이버 Papago 등





#1.3 NLP task

#3 질의응답



02. 텍스트 전처리

EWHA EURON

#2.1 텍스트 사전 준비 작업

#1 텍스트 정규화

: 텍스트를 머신러닝 알고리즘이나 NLP 애플리케이션에 입력 데이터로 사용하기 위해 클렌징, 정제, 토큰화, 어근화 등의 다양한 텍스트 데이터의 사전 작업을 수행하는 것을 의미

- 클렌징
- 토큰화
- 필터링 / 스톱 워드 제거 / 철자 수정
- Stemming
- Lemmatization

#2.2 텍스트 토큰화

#1 문장 토큰화

문장의 마침표, 개행문자 등 문장의 마지막을 뜻하는 기호에 따라 분리

```
from nltk import sent_tokenize
import nltk
nltk.download('punkt')

text_sample = 'The Matrix is everywhere its all around us, here even in this room. ₩
               You can see it out your window or on your television. ₩
               You feel it when you go to work, or go to church or pay your taxes.'
sentences = sent_tokenize(text=text_sample)
print(type(sentences),len(sentences))
print(sentences)
```

NTLK의 sent_tokenize를 이용
3개의 문장으로 된 문자열을 list 객체로 반환

```
 3
['The Matrix is everywhere its all around us, here even in this room.', 'You can see it out your window or on your television.', 'You feel it when you go
to work, or go to church or pay your taxes.']
```

EWHA EURON

#2.2 텍스트 토큰화

#2 단어 토큰화

공백, 콤마, 마침표, 개행문자 등으로 단어를 분리
단어의 순서가 중요하지 않은 경우 문장 토큰화를 사용하지 않고 단어 토큰화만 사용해도 충분

```
from nltk import word_tokenize

sentence = "The Matrix is everywhere its all around us, here even in this room."
words = word_tokenize(sentence)
print(type(words), len(words))
print(words)
```

NTLK의 word_tokenize를 이용

```
<class 'list'> 15
['The', 'Matrix', 'is', 'everywhere', 'its', 'all', 'around', 'us', ',', 'here', 'even', 'in', 'this', 'room', '.']
```

EWHA EURON

#2.2 텍스트 토큰화

sent_tokenize()와 word_tokenize()의 조합으로 문장의 모든 단어 토큰화

```
from nltk import word_tokenize, sent_tokenize

#여러개의 문장으로 된 입력 데이터를 문장별로 단어 토큰화 만드는 함수 생성
def tokenize_text(text):

    # 문장별로 분리 토큰
    sentences = sent_tokenize(text)
    # 분리된 문장별 단어 토큰화
    word_tokens = [word_tokenize(sentence) for sentence in sentences]
    return word_tokens

#여러 문장들에 대해 문장별 단어 토큰화 수행.
word_tokens = tokenize_text(text_sample)
print(type(word_tokens),len(word_tokens))
print(word_tokens)
```

```
 3
[['The', 'Matrix', 'is', 'everywhere', 'its', 'all', 'around', 'us', ',', 'here', 'even', 'in', 'this', 'room', '.'], ['You', 'can', 'see', 'it', 'out',
'your', 'window', 'or', 'on', 'your', 'television', '.'], ['You', 'feel', 'it', 'when', 'you', 'go', 'to', 'work', ',', 'or', 'go', 'to', 'church', 'or',
'pay', 'your', 'taxes', '.']]
```

#2.2 텍스트 토큰화

정규표현식 (Regular Expression)

- 정규표현식 모듈 re
- 특정 규칙이 있는 텍스트 데이터를 빠르게 정제할 수 있음

정규표현식 모듈 함수
- re.compile() : 정규표현식을 컴파일
- re.search() : 문자열 전체에 대해 정규표현식과 매치되는지 검색
- re.match() : 문자열의 처음 시작 부분이 정규표현식과 매치되는지 검색
- re.split() : 정규표현식을 기준으로 문자열 분리하여 리스트로 리턴
- re.findall() : 정규표현식과 매치되는 모든 문자열을 찾아 리스트로 리턴
- re.sub() : 정규표현식과 일치하는 부분을 다른 문자열로 대체

#2.2 텍스트 토큰화

정규표현식을 이용한 토큰화

```
from nltk.tokenize import RegexpTokenizer

text = "Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop"

tokenizer1 = RegexpTokenizer("[\w]+")
tokenizer2 = RegexpTokenizer("\s+", gaps=True)

print(tokenizer1.tokenize(text))
print(tokenizer2.tokenize(text))
```

```
['Don', 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'Mr', 'Jone', 's', 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
["Don't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name,', 'Mr.', "Jone's", 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```

tokenizer1에 사용한 \w+는 문자 또는 숫자가 1개 이상인 경우를 의미
gaps = True는 해당 정규식을 토큰으로 나누기 위한 기준으로 사용한다는 의미

tokenizer2의 결과는 위의 tokenizer1의 결과와는 달리 apostrophe나 온점을 제외하지 않고 토큰화가 수행된 것을 확인할 수 있음

EWHA EURON

#2.3 스톱 워드 제거

Stop word
: 분석에 큰 의미가 없는 단어 (is, the, a, will 등)

- 문법적인 특성으로 인해 빈번하게 텍스트에 나타나므로 사전에 제거하지 않으면 빈번함 때문에 오히려 중요한 단어로 인지될 수 있음
- NLTK에 언어별로 스톱 워드가 목록화 되어 있음

```
import nltk

stopwords = nltk.corpus.stopwords.words('english')
all_tokens = []
# 위 예제의 3개의 문장별로 얻은 word_tokens list 에 대해 stop word 제거 Loop
for sentence in word_tokens:
    filtered_words=[]
    # 개별 문장별로 tokenize된 sentence list에 대해 stop word 제거 Loop
    for word in sentence:
        #소문자로 모두 변환합니다.
        word = word.lower()
        # tokenize 된 개별 word가 stop words 들의 단어에 포함되지 않으면 word_tokens에 추가
        if word not in stopwords:
            filtered_words.append(word)
    all_tokens.append(filtered_words)

print(all_tokens)
```

```
[['matrix', 'everywhere', 'around', 'us', ',', 'even', 'room', '.'], ['see', 'window', 'television', '.'], ['feel', 'go', 'work', ',', 'go', 'church', 'pa
y', 'taxes', '.']]
```

EWHA EURON

#2.4 Stemming & Lemmatization

Stemming & Lemmatization
: 문법적 또는 의미적으로 변화하는 단어의 원형을 찾는 것

Stemming

- 단순화된 방법을 사용하여 원형 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향이 있음
- NTLK의 Porter, Lancaster, Snowball Stemmer

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

print(stemmer.stem('working'),stemmer.stem('works'),stemmer.stem('worked'))
print(stemmer.stem('amusing'),stemmer.stem('amuses'),stemmer.stem('amused'))
print(stemmer.stem('happier'),stemmer.stem('happiest'))
print(stemmer.stem('fancier'),stemmer.stem('fanciest'))
```

```
work work work
amus amus amus
happy happiest
fant fanciest
```

#2.4 Stemming & Lemmatization

Lemmatization

- 문법적 요소와 의미적인 부분을 감안하여 정확한 철자의 어근 단어를 추출
- 단어의 품사를 입력해줘야 함
- NTLK의 WordNetLemmatizer
- Stemming 보다 정교하며 의미론적인 기반에서 단어의 원형을 찾음

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

lemma = WordNetLemmatizer()
print(lemma.lemmatize('amusing','v'),lemma.lemmatize('amuses','v'),lemma.lemmatize('amused','v'))
print(lemma.lemmatize('happier','a'),lemma.lemmatize('happiest','a'))
print(lemma.lemmatize('fancier','a'),lemma.lemmatize('fanciest','a'))
```

```
amuse amuse amuse
happy happy
fancy fancy
```

#2.5 한국어 전처리

PyKoSpacing
: 띄어쓰기가 되어 있지 않은 문장을 띄어쓰기한 문장으로 변환

#Py-Hanspell
: 네이버 한글 맞춤법 검사기를 바탕으로 만든 맞춤법 보정 패키지, 띄어쓰기 또한 보정

#SOYNLP
: 품사 태킹, 단어 토큰화 등을 지원
: 비지도 학습으로 단어 토큰화, 데이터에 자주 등장하는 단어들을 단어로 분석

#Customized KoNLPy
: 형태소 분석기 Twitter 사용
: add_dictionary('단어', '품사')의 형식으로 사용자 사전 추가 가능
ex) twitter.add_dictionary('은경이', 'Noun')

03. BoW

#3.1 BoW

#1 BoW (Bag of Words)

- 문서가 가지는 모든 단어를 문맥이나 순서를 무시하고 일괄적으로 단어에 대해 빈도 값을 부여해 피처 값을 추출

+ 쉽고 빠른 구축
- 문맥 의미 반영 부족
- 희소 행렬 문제

#2 BoW 피처 벡터화

- 모든 문서에서 모든 단어를 칼럼 형태로 나열
- 각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경하는 것

- 카운트 기반의 벡터화
- TF-IDF 기반의 벡터화

#3.2 카운트 벡터화

#1 카운트 벡터화

- 단어 피처에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수, Count를 부여하는 경우
- Count 값이 높을수록 중요한 단어로 인식
- 언어의 특성상 문장에서 자주 사용될 수 밖에 없는 단어까지 높은 값을 부여하게 됨

#2 CountVectorizer 클래스

max_df	전체 문서에서 너무 높은 빈도수를 가지는 단어 피처를 제외
min_df	전체 문서에서 너무 낮은 빈도수를 단어 피처를 제외
max_features	추출하는 피처의 개수를 제한하여 정수로 값을 지정
stop_words	'english' 로 지정하면 영어의 스톱 워드로 지정된 단어는 추출에서 제외
n_gram_range	n_gram 범위를 설정, 튜플로 (범위 최솟값, 범위 최댓값)을 지정
anlayzer	피처 추출을 수행할 단위를 지정, default = 'word'
token_pattern	토큰화를 수행하는 정규 표현식 패턴을 지정
tokenizer	토큰화를 별도의 커스텀 함수로 이용시 적용

#3.3 TF-IDF

#1 TF-IDF 벡터화

- Term Frequency – Inverse Document Frequency
- 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 페널티를 주는 방식으로 값을 부여함
- 문서마다 텍스트가 길고 문서의 개수가 많은 경우 카운트 방식보다는 TF-IDF 방식을 사용하는 것이 더 좋은 예측 성능을 보장할 수 있음

#2 $TFIDF_i$

- $TFIDF_i = TF_i * \log\frac{N}{DF_i}$

- $TF_i =$ 개별 문서에서의 단어 i 빈도
- $DF_i =$ 단어 i를 가지고 있는 문서 개수
- N = 전체 문서 개수

#3 TfidfVectorizer 클래스

EWHA EURON

#3.3 TF-IDF

#4 TF-IDF 원리

- TF: 개별 문서에서 해당 단어의 빈도
- IDF: 해당 단어의 중요도

문서1: 배가 부르다
문서2: 배의 가격이 비싸다
문서3: 진짜 사과가 진짜 좋다
문서4: 아침엔 사과가 좋다

	TF				IDF	TF * IDF			
가격이	0	1/3	0	0	ln(4/1)	0.00000	0.57735	0.00000	0.00000
배가	1/2	0	0	0	ln(4/1)	0.70711	0.00000	0.00000	0.00000
배의	0	1/3	0	0	ln(4/1)	0.00000	0.57735	0.00000	0.00000
부르다	1/2	0	0	0	ln(4/1)	0.70711	0.00000	0.00000	0.00000
비싸다	0	1/3	0	0	ln(4/1)	0.00000	0.57735	0.00000	0.00000
사과가	0	0	1/4	1/3	ln(4/2)	0.00000	0.00000	0.34432	0.52641
아침엔	0	0	0	1/3	ln(4/1)	0.00000	0.00000	0.00000	0.66768
좋다	0	0	1/4	1/3	ln(4/2)	0.00000	0.00000	0.34432	0.52641
진짜	0	0	2/4	0	ln(4/1)	0.00000	0.00000	0.87344	0.00000

#3.4 BOW 벡터화를 위한 희소행렬

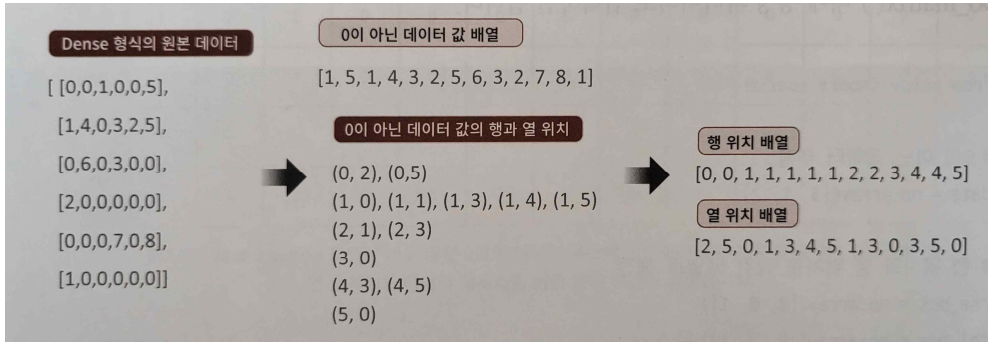
#1 희소 행렬

- 대규모 행렬의 대부분의 값을 차지하는 행렬
- BOW 형태를 가진 언어 모델의 피처 벡터화는 대부분 희소 행렬
- 불필요한 0 값이 메모리에 할당되어 공간이 많이 필요하며, 연산 시에도 시간이 많이 소모
  ➔ 희소행렬을 물리적으로 적은 메모리 공간을 차지할 수 있도록 변환해야 함 : COO 형식, CSR 형식

#3.4 BOW 벡터화를 위한 희소행렬

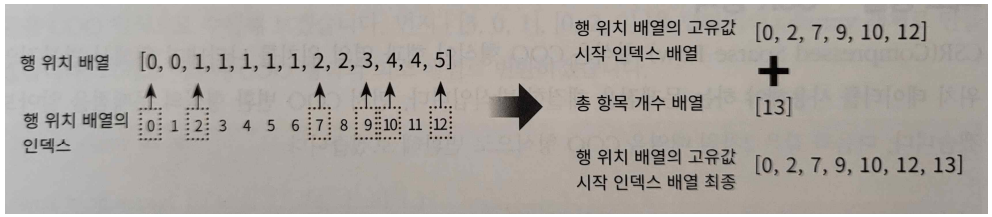
#2 COO 형식

- 0이 아닌 데이터만 별도의 데이터 배열에 저장하고, 그 데이터가 가리키는 행과 열의 위치를 별도의 배열로 저장하는 방식



#3 CSR 방식

- COO 형식이 행과 열의 위치를 나타내기 위해서 반복적인 위치 데이터를 사용해야 하는 문제점을 해결한 방식



04. 텍스트 분류 실습 - 20 뉴스그룹

EWHA EURON

#4-1. 텍스트 분류

- 특정 문서의 분류를 학습 데이터를 통해 학습해 모델을 생성한 뒤, 해당 모델을 이용해 다른 문서의 분류를 예측하는 것

- 텍스트를 피처 벡터화로 변환 시 일반적으로 **희소 행렬** 형태가 됨
  - 희소 행렬 처리 알고리즘: 로지스틱 회귀, 선형 서포트 벡터 머신, 나이브 베이즈 등

- 텍스트 기반 분류 Process
  1. 텍스트 정규화
  2. 피처 벡터화
  3. 적합한 머신러닝 알고리즘을 적용해 분류 학습/예측/평가

#4-2. 텍스트 정규화

```
### 예제 데이터 준비

from sklearn.datasets import fetch_20newsgroups
news_data = fetch_20newsgroups(subset='all', random_state=156)
```

```
### 데이터 key값 확인

print(news_data.keys())
```

인터넷에서 내려받은 파일이 로컬 컴퓨터에 저장되는 디렉토리/파일명

```
dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])
```

```
### Target 클래스 구성 확인

import pandas as pd

print('target 클래스의 값과 분포도 ₩n',pd.Series(news_data.target).value_counts().sort_index())
print('target 클래스의 이름들 ₩n',news_data.target_names)
```

- 20개의 클래스(0 ~ 19)
- Class명
  0. alt.atheism
  1. comp.graphics
  2. comp.os.ms-windows.misc
  3. comp.sys.ibm.pc.hardware
  4. comp.sys.mac.hardware
  5. comp.windows.x
  6. misc.forsale
  7. rec.autos
  8. rec.motorcycles
  9. rec.sport.baseball
  10. rec.sport.hockey
  11. sci.crypt
  12. sci.electronics
  13. sci.med
  14. sci.space
  15. soc.religion.Christian
  16. talk.politics.guns
  17. talk.politics.Mideast
  18. talk.politics.misc
  19. talk.religion.misc

#4-2. 텍스트 정규화

```
### 개별 데이터가 텍스트로 어떻게 구성되는지 확인

print(news_data.data[0])
```

```
From: egreen@east.sun.com (Ed Green - Pixel Cruncher)
Subject: Re: Observation re: helmets
Organization: Sun Microsystems, RTP, NC
Lines: 21
Distribution: world
Reply-To: egreen@east.sun.com
NNTP-Posting-Host: laser.east.sun.com

In article 211353@mavenry.altcit.eskimo.com, maven@mavenry.altcit.eskimo.com (Norman Hamer) writes:
>
> The question for the day is re: passenger helmets, if you don't know for
>certain who's gonna ride with you (like say you meet them at a .... church
>meeting, yeah, that's the ticket)... What are some guidelines? Should I just
>pick up another shoei in my size to have a backup helmet (XL), or should I
>maybe get an inexpensive one of a smaller size to accomodate my likely
>passenger?

If your primary concern is protecting the passenger in the event of a
crash, have him or her fitted for a helmet that is their size.  If your
primary concern is complying with stupid helmet laws, carry a real big
spare (you can put a big or small head in a big helmet, but not in a
small one).

---
Ed Green, former Ninjaite |I was drinking last night with a biker,
  Ed.Green@East.Sun.COM   |and I showed him a picture of you.  I said,
DoD #0111  (919)460-8302  |"Go on, get to know her, you'll like her!"
 (The Grateful Dead) -->  |It seemed like the least I could do...
```

#4-2. 텍스트 정규화

```
### remove 파라미터를 이용해 header와 footer 등을 제거
### subset 파라미터를 이용해 train/test set 분리

from sklearn.datasets import fetch_20newsgroups

# subset='train'으로 학습용(Train) 데이터만 추출
# remove=('headers', 'footers', 'quotes')로 내용만 추출
train_news= fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'), random_state=156)
X_train = train_news.data
y_train = train_news.target
print(type(X_train))

# subset='test'으로 테스트(Test) 데이터만 추출
# remove=('headers', 'footers', 'quotes')로 내용만 추출
test_news= fetch_20newsgroups(subset='test',remove=('headers', 'footers','quotes'),random_state=156)
X_test = test_news.data
y_test = test_news.target
print('학습 데이터 크기 {0} , 테스트 데이터 크기 {1}'.format(len(train_news.data) , len(test_news.data)))
```

```
<class 'list'>
학습 데이터 크기 11314 , 테스트 데이터 크기 7532
```

EWHA EURON

#4-3. 피처 벡터화 변환과 ML 모델 학습/예측/평가

- 피처 벡터화

1. Count 기반 피처 벡터화

📌 주의사항

- 테스트 데이터에서 CountVectorizer 적용 시 반드시 학습 데이터를 이용해 fit( )이 수행된 CountVectorizer 객체를 이용해 데이터를 변환해야 함
- 테스트 데이터의 피처 벡터화 시 fit_transform( )을 사용하면 x
  → 테스트 데이터 기반으로 다시 CountVectorizer가 fit( )을 수행하고 transform( ) 진행하면 학습 시 사용된 피처 개수와 예측 시 사용할 피처 개수가 **달라짐**

```
from sklearn.feature_extraction.text import CountVectorizer

### CountVectorization으로 feature extraction 변환
cnt_vect = CountVectorizer() # 객체 생성
cnt_vect.fit(X_train) # 학습
X_train_cnt_vect = cnt_vect.transform(X_train) # 데이터 변환

### 학습 데이터로 fit( )된 CountVectorizer를 이용하여 테스트 데이터를 feature extraction 변환
X_test_cnt_vect = cnt_vect.transform(X_test)

print('학습 데이터 Text의 CountVectorizer Shape:',X_train_cnt_vect.shape)
```

```
학습 데이터 Text의 CountVectorizer Shape: (11314, 101631)
```

EWHA EURON

#4-3. 피처 벡터화 변환과 ML 모델 학습/예측/평가

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

### LogisticRegression을 이용하여 학습/예측/평가
lr_clf = LogisticRegression() # 객체 생성
lr_clf.fit(X_train_cnt_vect , y_train) # 학습
pred = lr_clf.predict(X_test_cnt_vect) # 예측
print('CountVectorized Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test,pred))) # 평가
```

```
CountVectorized Logistic Regression 의 예측 정확도는 0.608
```

EWHA EURON

#4-3. 피처 벡터화 변환과 ML 모델 학습/예측/평가

- 피처 벡터화

2. TF-IDF 기반 피처 벡터화

```
from sklearn.feature_extraction.text import TfidfVectorizer

### TF-IDF Vectorization 적용하여 학습 데이터셋과 테스트 데이터 셋 변환
tfidf_vect = TfidfVectorizer() # 객체 생성
tfidf_vect.fit(X_train) # 학습
X_train_tfidf_vect = tfidf_vect.transform(X_train) # 변환
X_test_tfidf_vect = tfidf_vect.transform(X_test) # 학습 데이터로 fit()된 객체를 사용해야 한다.

### LogisticRegression을 이용하여 학습/예측/평가
lr_clf = LogisticRegression() # 객체 생성
lr_clf.fit(X_train_tfidf_vect , y_train) # 학습
pred = lr_clf.predict(X_test_tfidf_vect) # 예측
print('TF-IDF Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred))) # 평가
```

```
TF-IDF Logistic Regression 의 예측 정확도는 0.674
```

- Count 기반: 0.608
- TF-IDF 기반: 0.674

⇒ 일반적으로 문서 내에 텍스트가 많고 많은 문서를 가지는 텍스트 분석에서는 카운트 벡터화보다는 TF-IDF 벡터화가 좋은 예측 결과를 도출함

EWHA EURON

#4-3. 피처 벡터화 변환과 ML 모델 학습/예측/평가

- 성능 향상

1. 최적의 ML 알고리즘 선택
2. 최상의 피처 전처리 수행하기
   - 하이퍼 파라미터 튜닝 등

```
### stop words 필터링을 추가하고 ngram을 기본(1,1)에서 (1,2)로 변경하여 Feature Vectorization 적용
tfidf_vect = TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_df=300 )
tfidf_vect.fit(X_train)
X_train_tfidf_vect = tfidf_vect.transform(X_train)
X_test_tfidf_vect = tfidf_vect.transform(X_test)

lr_clf = LogisticRegression()
lr_clf.fit(X_train_tfidf_vect , y_train) # 학습
pred = lr_clf.predict(X_test_tfidf_vect) # 예측
print('TF-IDF Vectorized Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred))) # 평가
```

```
TF-IDF Vectorized Logistic Regression 의 예측 정확도는 0.692
```

```
### GridSearchCV를 이용해 로지스틱 회귀의 하이퍼 파라미터 최적화 수행

from sklearn.model_selection import GridSearchCV

# 최적 C 값 도출 튜닝 수행. CV는 3 Fold셋으로 설정.
params = { 'C':[0.01, 0.1, 1, 5, 10]}
grid_cv_lr = GridSearchCV(lr_clf ,param_grid=params , cv=3 , scoring='accuracy' , verbose=1)
grid_cv_lr.fit(X_train_tfidf_vect , y_train)
print('Logistic Regression best C parameter :',grid_cv_lr.best_params_ )

# 최적 C 값으로 학습된 grid_cv로 예측 수행하고 정확도 평가.
pred = grid_cv_lr.predict(X_test_tfidf_vect)
print('TF-IDF Vectorized Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred)))
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
Logistic Regression best C parameter : {'C': 10}
TF-IDF Vectorized Logistic Regression 의 예측 정확도는 0.701
```

#4-4. 사이킷런 파이프라인(Pipeline) 사용 및 GridSearchCV와의 결합

- 사이킷런의 Pipeline 클래스를 이용해 피처 벡터화와 ML 알고리즘 학습/예측을 위한 코드 작성을 한 번에 진행할 수 있음

- 데이터의 전처리와 머신러닝 학습 과정을 통일된 API 기반에서 처리할 수 있음
  -> 더 직관적인 ML 모델 코드를 생성할 수 있음

- 대용량 데이터의 피쳐 벡터화 결과를 별도 데이터로 저장하지 않고 스트림 기반에서 바로 머신러닝 알고리즘의 데이터로 입력할 수 있음 -> 수행 시간 절약

#4-4. 사이킷런 파이프라인(Pipeline) 사용 및 GridSearchCV와의 결합

```
### pipeline 객체 선언

from sklearn.pipeline import Pipeline

# TfidfVectorizer 객체를 tfidf_vect 객체명으로, LogisticRegression객체를 lr_clf 객체명으로 생성하는 Pipeline
pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_df=300)),
    ('lr_clf', LogisticRegression(C=10))
])

# 별도의 TfidfVectorizer 객체의 fit_transform( )과 LogisticRegression의 fit(), predict( )가 필요 없음.
# pipeline의 fit( ) 과 predict( ) 만으로 한꺼번에 Feature Vectorization과 ML 학습/예측이 가능.
pipeline.fit(X_train, y_train) # 학습
pred = pipeline.predict(X_test) # 예측
print('Pipeline을 통한 Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred))) # 평가
```

```
Pipeline을 통한 Logistic Regression 의 예측 정확도는 0.701
```

EWHA EURON

#4-4. 사이킷런 파이프라인(Pipeline) 사용 및 GridSearchCV와의 결합

- GridSearchCV에 Pipeline 입력 시의 param_grid 설정

- 하이퍼 파라미터명이 객체 변수명과 결합돼 제공
- GridSearchCV가 Pipeline을 구성하는 피처 벡터화 객체의 파라미터와 Estimator 객체의 하이퍼 파라미터를 각각 구별할 수 있도록 개별 객체 명과 파라미터명/하이퍼 파라미터명을 결합해 Key 값으로 할당

```
### GridSearchCV에 Pipeline을 입력하면서 TfidVectorizer의 parameter와 LogisticRegression의 하이퍼 파라미터를 함께 최적화
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words='english')),
    ('lr_clf', LogisticRegression(solver='liblinear'))
])

# Pipeline에 기술된 각각의 객체 변수에 언더바(_) 2개를 연달아 붙여 GridSearchCV에 사용될 파라미터/하이퍼 파라미터 이름과 값을 설정
params = { 'tfidf_vect__ngram_range': [(1,1), (1,2), (1,3)],
           'tfidf_vect__max_df': [100, 300, 700],
           'lr_clf__C': [1, 5, 10]
}

# GridSearchCV의 생성자에 Estimator가 아닌 Pipeline 객체 입력
grid_cv_pipe = GridSearchCV(pipeline, param_grid=params, cv=3 , scoring='accuracy',verbose=1)
grid_cv_pipe.fit(X_train , y_train)
print(grid_cv_pipe.best_params_ , grid_cv_pipe.best_score_)

pred = grid_cv_pipe.predict(X_test)
print('Pipeline을 통한 Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred)))
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
{'lr_clf__C': 10, 'tfidf_vect__max_df': 700, 'tfidf_vect__ngram_range': (1, 2)} 0.7550828826229531
Pipeline을 통한 Logistic Regression 의 예측 정확도는 0.702
```

05. 감성 분석

EWHA EURON

#5-1. 감성 분석

- 문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
- 문서 내 텍스트가 나타내는 여러 가지 주관적인 단어와 문맥을 기반으로 **감성(Sentiment) 수치**를 계산
- 긍정 감성 지수와 부정 감정 지수로 구성
  → 이들을 합산해 긍정/ 부정 감성 결정
- **감정 분석 방식**

  1. 지도학습

  - **학습** 데이터와 **타깃** 레이블 값을 기반으로 감성 분석 학습 수행
  - 이를 기반으로 다른 데이터의 감성 분석 예측
  - 일반적인 텍스트 기반의 **분류**와 거의 동일

  2. 비지도학습

  - Lexicon이라는 일종의 **감성 어휘 사전** 활용
  - 용어/ 문맥 정보 이용 -> 긍정적, 부정적 감성 여부 판단

#5-2. 지도학습 기반 감성 분석 – IMDB 영화평

```
import pandas as pd

### Data Loading

review_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Euron 3기_DS/data/Week13/labeledTrainData.tsv', header = 0, sep = "₩t", quoting = 3)
review_df.head(3)
```

	id	sentiment	review
0	"5814_8"	1	"With all this stuff going down at the moment ...
1	"2381_9"	1	"₩"The Classic War of the Worlds₩" by Timothy ...
2	"7759_3"	0	"The film starts with a manager (Nicholas Bell...

- Data Description
  - id: 각 데이터의 id
  - sentiment: 영화평(review)의 Sentiment 결과 값(Target Label)
    1은 긍정적 평가, 0은 부정적 평가
  - review: 영화평의 텍스트

EWHA EURON

#5-2. 지도학습 기반 감성 분석 – IMDB 영화평

```
### 텍스트 구성 확인하기

print(review_df['review'][0])
```

```
"With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary here and there, watched The Wiz and watched Moonwalker again. Maybe i just want to get a certain
```

1. 데이터 전처리

- HTML 형식의 텍스트 → **
** 태그 삭제(공백 처리)
- 숫자/ 특수문자 삭제(공백 처리) → **정규 표현식** 활용

```
import re

# <br> html 태그는 replace 함수로 공백으로 변환
review_df['review'] = review_df['review'].str.replace('<br />',' ')

# 파이썬의 정규 표현식 모듈인 re를 이용하여 영어 문자열이 아닌 문자는 모두 공백으로 변환
review_df['review'] = review_df['review'].apply( lambda x : re.sub("[^a-zA-Z]", " ", x) )
```

#5-2. 지도학습 기반 감성 분석 – IMDB 영화평

2. 데이터 준비

- 결정 값 클래스인 sentiment 칼럼 추출 → label 데이터 세트
- 원본 데이터 세트에서 id와 sentiment 칼럼 삭제 → 피처 데이터 세트
- train_test_split( ) 이용 → 학습/테스트용 데이터 세트로 분리

```
from sklearn.model_selection import train_test_split

class_df = review_df['sentiment']
feature_df = review_df.drop(['id','sentiment'], axis=1, inplace=False)

X_train, X_test, y_train, y_test = train_test_split(feature_df, class_df, test_size = 0.3, random_state = 156)
X_train.shape, X_test.shape
```

```
((17500, 1), (7500, 1))
```

EWHA EURON

#5-2. 지도학습 기반 감성 분석 – IMDB 영화평

3. 피처 벡터화 & 예측 성능 측정

- Pipeline 객체 이용 → 한 번에 수행
- **Count** 벡터화/ **TF-IDF** 벡터화 적용
- Classifier로 **LogisticRegression** 이용
- 예측 성능 평가: 테스트 데이터 세트의 **정확도** + ROC-AUC 측정

#5-2. 지도학습 기반 감성 분석 – IMDB 영화평

- Count 벡터화 적용

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score

# 스톱 워드는 English, filtering, ngram은 (1,2)로 설정해 CountVectorization 수행
# LogisticRegression의 C는 10으로 설정
pipeline = Pipeline([
    ('cnt_vect', CountVectorizer(stop_words = 'english', ngram_range = (1,2))),
    ('lr_clf', LogisticRegression(C = 10))])

# Pipeline 객체를 이용하여 fit(), predict()로 학습/예측 수행
# predict_proba()는 roc_auc 때문에 수행

pipeline.fit(X_train['review'], y_train) # 학습
pred = pipeline.predict(X_test['review']) # 예측
pred_probs = pipeline.predict_proba(X_test['review'])[:,1] # 평가

print('예측 정확도는 {0:.4f}, ROC-AUC는 {1:.4f}'.format(accuracy_score(y_test ,pred),
                                                    roc_auc_score(y_test, pred_probs)))
```

- 예측 정확도: 0.8860
- ROC-AUC: 0.9503

EWHA EURON

#5-2. 지도학습 기반 감성 분석 – IMDB 영화평

- TF-IDF 벡터화 적용

```
# 스톱 워드는 english, filtering, ngram은 (1,2)로 설정해 TF-IDF 벡터화 수행
# LogisticRegression의 C는 10으로 설정

pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words = 'english', ngram_range = (1,2) )),
    ('lr_clf', LogisticRegression(C = 10))])

pipeline.fit(X_train['review'], y_train) # 학습
pred = pipeline.predict(X_test['review']) # 예측
pred_probs = pipeline.predict_proba(X_test['review'])[:,1] # 평가

print('예측 정확도는 {0:.4f}, ROC-AUC는 {1:.4f}'.format(accuracy_score(y_test ,pred),
                                        roc_auc_score(y_test, pred_probs)))
```

- 예측 정확도: 0.8936
- ROC-AUC: 0.9598

→ TF-IDF 기반 피처 벡터화의 예측 성능이 조금 더 나아짐

EWHA EURON

#5-3. 비지도학습 기반 감성 분석

- 결정된 레이블 값을 가지고 있지 않은 데이터를 분석
- Lexicon 기반
  - 주로 감성만을 분석하기 위해 지원하는 **감성 어휘 사전**
  - **감성 지수** 활용 → 긍정 감성 + 부정 감성
  - 단어의 위치, 주변 단어, 문맥, POS(Part of Speech/ 품사) 등을 참고해 결정
  - **NLTK 패키지** 활용

#5-3. 비지도학습 기반 감성 분석

- WordNet
  - NLP에서 제공하는 방대한 **영어 어휘 사전** 모듈
  - 다양한 상황에서 같은 어휘라도 다르게 사용되는 어휘의 **시맨틱 정보**를 제공
  - 각각의 품사로 구성된 개별 단어를 Synset이라는 개념을 이용해 표현

- Synset
  - 단어가 가지는 **문맥 + 시맨틱 정보** 제공
  - synsets( ) 호출 시 여러 개의 Synset 객체를 가지는 리스트가 반환됨
  - POS(Part Of Speech/ 품사), 정의, 부명제 등으로 시맨틱적인 요소를 표현할 수 있음
  - path_similarity( ) : 어떤 어휘와 다른 어휘 간의 **유사도** 표시

#5-3. 비지도학습 기반 감성 분석

1. SentiWordNet을 이용한 감성 분석

- WordNet Synset과 SentiWordNet SentiSynset 클래스의 이해

```
### NLTK의 모든 데이터 세트/ 패키지 다운로드

import nltk
nltk.download('all')
```

WordNet

```
[ ]  from nltk.corpus import wordnet as wn

     term = 'present'

     # 'present'라는 단어로 wordnet의 synsets 생성

     synsets = wn.synsets(term)
     print('synsets() 반환 type :', type(synsets))
     print('synsets() 반환 값 갯수:', len(synsets))
     print('synsets() 반환 값 :', synsets)
```

```
synsets() 반환 type : <class 'list'>
synsets() 반환 값 갯수: 18
synsets() 반환 값 : [Synset('present.n.01'), Synset('present.n.02'), Synset('present.n.03'), Synset('show.v.01'), Synset('present.v.02'), Synset('stage.v.01'),
```

EWHA EURON

#5-3. 비지도학습 기반 감성 분석

- WordNet을 이용한 감성 분석

```
### synset 객체가 가지는 여러 가지 속성 살펴보기

for synset in synsets:
    print('##### Synset name :  ', synset.name(),'#####')
    print('POS :',synset.lexname()) # 품사
    print('Definition:',synset.definition()) # 정의
    print('Lemmas:',synset.lemma_names()) # 부명제
```

```
##### Synset name :  present.n.01 #####
POS : noun.time
Definition: the period of time that is happening now; any continuous stretch of time including the moment of speech
Lemmas: ['present', 'nowadays']
##### Synset name :  present.n.02 #####
POS : noun.possession
Definition: something presented as a gift
Lemmas: ['present']
##### Synset name :  present.n.03 #####
POS : noun.communication
Definition: a verb tense that expresses actions or states at the time of speaking
Lemmas: ['present', 'present_tense']
##### Synset name :  show.v.01 #####
POS : verb.perception
Definition: give an exhibition of to an interested audience
Lemmas: ['show', 'demo', 'exhibit', 'present', 'demonstrate']
```

- sysnset 객체: 하나의 단어가 가질 수 있는 여러 가지 **시맨틱 정보**를 개별 클래스로 나타낸 것

EWHA EURON

#5-3. 비지도학습 기반 감성 분석

- WordNet을 이용한 감성 분석

```
### 어휘 간의 유사도 파악

# synset 객체를 단어별로 생성
tree = wn.synset('tree.n.01')
lion = wn.synset('lion.n.01')
tiger = wn.synset('tiger.n.02')
cat = wn.synset('cat.n.01')
dog = wn.synset('dog.n.01')

entities = [tree , lion , tiger , cat , dog]
similarities = []
entity_names = [entity.name().split('.')[0] for entity in entities]

# 단어별 synset들을 iteration 하면서 다른 단어들의 synset과 유사도를 측정
for entity in entities:
    similarity = [round(entity.path_similarity(compared_entity), 2)  for compared_entity in entities ]
    similarities.append(similarity)

# 개별 단어별 synset과 다른 단어의 synset과의 유사도를 DataFrame형태로 저장
similarity_df = pd.DataFrame(similarities , columns = entity_names,index = entity_names)
similarity_df
```

	tree	lion	tiger	cat	dog
tree	1.00	0.07	0.07	0.08	0.12
lion	0.07	1.00	0.33	0.25	0.17
tiger	0.07	0.33	1.00	0.25	0.17
cat	0.08	0.25	0.25	1.00	0.20
dog	0.12	0.17	0.17	0.20	1.00

EWHA EURON

#5-3. 비지도학습 기반 감성 분석

- SentiWordNet을 이용한 감성 분석
  - senti_synsets()
    - Senti_Synset 클래스를 리스트 형태로 반환
    - SentiSynset 객체: 감성 지수(단어의 감성) + 객관성 지수(객관성을 나타냄)
      - 감성 지수: 긍정 감정 지수 + 부정 감정 지수
      - 어떤 단어가 전혀 감성적이지 않으면 객관성 지수는 1이 되고, 감성 지수는 모두 0이 됨

#5-3. 비지도학습 기반 감성 분석

- SentiWordNet을 이용한 감성 분석

```
import nltk
from nltk.corpus import sentiwordnet as swn

senti_synsets = list(swn.senti_synsets('slow'))
print('senti_synsets() 반환 type :', type(senti_synsets))
print('senti_synsets() 반환 값 갯수:', len(senti_synsets))
print('senti_synsets() 반환 값 :', senti_synsets)
```

```
senti_synsets() 반환 type : <class 'list'>
senti_synsets() 반환 값 갯수: 11
senti_synsets() 반환 값 : [SentiSynset('decelerate.v.01'), SentiSynset('slow.v.02'), SentiSynset('slow.v.03'), SentiSynset('slow.a.01'),
```

```
import nltk
from nltk.corpus import sentiwordnet as swn

father = swn.senti_synset('father.n.01')
print('father 긍정감성 지수: ', father.pos_score())
print('father 부정감성 지수: ', father.neg_score())
print('father 객관성 지수: ', father.obj_score())
print('₩n')
fabulous = swn.senti_synset('fabulous.a.01')
print('fabulous 긍정감성 지수: ',fabulous .pos_score())
print('fabulous 부정감성 지수: ',fabulous .neg_score())
```

```
father 긍정감성 지수:  0.0
father 부정감성 지수:  0.0
father 객관성 지수:  1.0


fabulous 긍정감성 지수:  0.875
fabulous 부정감성 지수:  0.125
```

EWHA EURON

#5-3. 비지도학습 기반 감성 분석

- SentiWordNet을 이용한 영화 감상평 감성 분석
  - 순서

  1. 문서를 **문장** 단위로 분해
  2. 다시 문장을 **단어** 단위로 토큰화하고 **품사 태깅**
  3. 품사 태깅된 단어 기반으로 **synset 객체**와 **senti_synset 객체** 생성
  4. Senti_synset에서 긍정/부정 감성 지수를 구하고, 이를 모두 합산해 특정 임계치 값 이상일 때 긍정 감성, 아닐 때 부정 감성으로 결정

#5-3. 비지도학습 기반 감성 분석

- SentiWordNet을 이용한 영화 감상평 감성 분석

```
### 품사 태깅을 수행하는 내부 함수 생성

from nltk.corpus import wordnet as wn

# 간단한 NTLK PennTreebank Tag를 기반으로 WordNet 기반의 품사 Tag로 변환
def penn_to_wn(tag):
    if tag.startswith('J'):
        return wn.ADJ
    elif tag.startswith('N'):
        return wn.NOUN
    elif tag.startswith('R'):
        return wn.ADV
    elif tag.startswith('V'):
        return wn.VERB
    return
```

#5-3. 비지도학습 기반 감성 분석

- SentiWordNet을 이용한 영화 감상평 감성 분석

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import sentiwordnet as swn
from nltk import sent_tokenize, word_tokenize, pos_tag
```

```
### 문서를 문장 -> 단어 토큰 -> 품사 태깅 후 SentiSynset 클래스 생성, Polarity Score를 합산하는 함수
def swn_polarity(text):
    # 감성 지수 초기화
    sentiment = 0.0
    tokens_count = 0
    lemmatizer = WordNetLemmatizer()
    raw_sentences = sent_tokenize(text)

    ### 분해된 문장별로 단어 토큰 -> 품사 태깅 후에 SentiSynset 생성 -> 감성 지수 합산
    for raw_sentence in raw_sentences:
        # NTLK 기반의 품사 태깅 문장 추출
        tagged_sentence = pos_tag(word_tokenize(raw_sentence))
        for word , tag in tagged_sentence:
            # WordNet 기반 품사 태깅과 어근 추출
            wn_tag = penn_to_wn(tag)
            if wn_tag not in (wn.NOUN , wn.ADJ, wn.ADV):
                continue
            lemma = lemmatizer.lemmatize(word, pos=wn_tag)
            if not lemma:
                continue
            # 어근을 추출한 단어와 WordNet 기반 품사 태깅을 입력해 Synset 객체를 생성
            synsets = wn.synsets(lemma , pos=wn_tag)
            if not synsets:
                continue
            # sentiwordnet의 감성 단어 분석으로 감성 synset 추출
            # 모든 단어에 대해 긍정 감성 지수는 +로 부정 감성 지수는 -로 합산해 감성 지수 계산
            synset = synsets[0]
            swn_synset = swn.senti_synset(synset.name())
            sentiment += (swn_synset.pos_score() - swn_synset.neg_score())
            tokens_count += 1
    if not tokens_count:
        return 0
    # 총 score가 0 이상일 경우 긍정(Positive) 1, 그렇지 않을 경우 부정(Negative) 0 반환
    if sentiment >= 0 :
        return 1
    return 0
```

EWHA EURON

#5-3. 비지도학습 기반 감성 분석

- SentiWordNet을 이용한 영화 감상평 감성 분석

```
### IMDB 감상평의 개별 문서에 적용해 긍정/부정 감성 예측

review_df['preds'] = review_df['review'].apply( lambda x : swn_polarity(x) )
y_target = review_df['sentiment'].values
preds = review_df['preds'].values
```

```
### 예측 성능

from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
from sklearn.metrics import recall_score, f1_score, roc_auc_score
import numpy as np

print(confusion_matrix( y_target, preds))
print("정확도:", np.round(accuracy_score(y_target , preds), 4))
print("정밀도:", np.round(precision_score(y_target , preds),4))
print("재현율:", np.round(recall_score(y_target, preds), 4))
```

```
[[7668 4832]
 [3636 8864]]
```

- 정확도: 0.6613
- 정밀도: 0.6472
- 재현율: 0.7091

→ 정확도 지표를 포함한 전반적인 성능 평가 지표가 만족스러울 만한 수치는 아니다.

EWHA EURON

#5-3. 비지도학습 기반 감성 분석

2. VADER를 이용한 감성 분석

- VADER는 **소셜 미디어**의 감성 분석 용도로 만들어진 룰 기반의 Lexicon
- **SentimentIntensityAnalyzer** 클래스를 이용해 감성 분석 수행
  - 'neg' : 부정 감성 지수
  - 'neu' : 중립 감성 지수
  - 'pos' : 긍정 감성 지수
- compound: neg, neu, pos score를 적절히 조합해 **-1에서 1 사이**의 감성 지수 표현
  - 0.1 이상: 긍정/ 0.1 이하: 부정 감성

#5-3. 비지도학습 기반 감성 분석

2. VADER를 이용한 감성 분석

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

senti_analyzer = SentimentIntensityAnalyzer() # 객체 생성
senti_scores = senti_analyzer.polarity_scores(review_df['review'][0]) # 감성 점수 구하기
print(senti_scores) # 감성 점수가 딕셔너리 형태로 반환됨
```

```
{'neg': 0.13, 'neu': 0.743, 'pos': 0.127, 'compound': -0.7943}
```

```
def vader_polarity(review,threshold=0.1):
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review) #  감성 점수 구하기

    # compound 값에 기반하여 threshold(임계값)가 입력값보다 크면 1, 그렇지 않으면 0을 반환
    agg_score = scores['compound']
    final_sentiment = 1 if agg_score >= threshold else 0
    return final_sentiment

### apply lambda 식을 이용하여 레코드별로 vader_polarity( )를 수행하고 결과를 'vader_preds'에 저장
review_df['vader_preds'] = review_df['review'].apply(lambda x : vader_polarity(x, 0.1))
y_target = review_df['sentiment'].values
vader_preds = review_df['vader_preds'].values

print(confusion_matrix( y_target, vader_preds))
print("정확도:", np.round(accuracy_score(y_target , vader_preds),4))
print("정밀도:", np.round(precision_score(y_target , vader_preds),4))
print("재현율:", np.round(recall_score(y_target, vader_preds),4))
```

```
[[ 6747  5753]
 [ 1858 10642]]
정확도: 0.6956
정밀도: 0.6491
재현율: 0.8514
```

- 정확도 향상
- 재현율: 약 86.05% → 매우 크게 향상됨

EWHA EURON

#5-4. 한국어 감성 어휘 사전

- KoNLPy 라이브러리
  - 한국어 처리를 위한 파이썬 라이브러리
  - 파이썬에서 사용할 수 있는 오픈 소스 형태소 분석기
    - 기존에 공개된 형태소 분석기인 꼬꼬마, 코모란, 한나눔, 트위터, 메카브 등을 한 번에 설치/ 사용할 수 있도록 함
  - 관련 문서: https://konlpy.org/ko/latest/index.html#

  - Window 환경에서 KoNLPy 설치
    - 형태소 분석기는 JAVA 기반으로 개발됨 -> Oracle JDK(Java Development Kit) 설치 권장
    - 설치 관련: [딥러닝 파이토치 교과서(wikidocs)] https://thebook.io/080289/ch09/01/02-02/

#5-4. 한국어 감성 어휘 사전

- **한국어 처리를 위한 감성 어휘 사전**

1. Hannanum 시스템 사전
   - KAIST 말뭉치를 이용해 생성된 사전
2. Kkma 시스템 사전
   - 세종 말뭉치를 이용해 생성된 사전
3. Mecab 시스템 사전
   - 세종 말뭉치로 만들어진 CSV 형태의 사전
4. KNU한국어감성사전
   - 군산대 소프트웨어융합공학과 연구실에서 개발한 한국어감정사전
   - 표준국어대사전을 구성하는 각 단어의 뜻풀이를 분석하여 긍부정어를 추출
   - 여러 자료들을 통합해 14000여개의 1-gram, 2-gram, 관용구, 문형, 축약어, 이모티콘 등에 대한 긍정, 중립, 부정 판별 및 정도(degree)의 값 계산

- **예제**

1. 한국어 감성 분석기(Github): https://github.com/mrlee23/KoreanSentimentAnalyzer
2. 한국어 토큰화 – 딥러닝 파이토치 교과서: https://thebook.io/080289/ch09/02/02-03/

06. 토픽 모델링

EWHA EURON

#6-1. 토픽 모델링(Topic Modeling)

- 문서 집합에 숨어 있는 주제를 찾아내는 것
- 머신러닝 기반의 토픽 모델: 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출
- 자주 사용되는 기법

- LSA(Latent Semantic Analysis)
  - 기존의 DTM이나 DTM에 단어의 중요도에 따른 가중치를 주었던 TF-IDF 행렬은 단어의 의미를 전혀 고려하지 못한다는 단점이 존재
  - LSA는 기본적으로 DTM이나 TF-IDF 행렬에 절단된 SVD(truncated SVD)를 사용하여 **차원을 축소**시키고, 단어들의 **잠재적인 의미**를 끌어낸다는 아이디어
  - 이미 계산된 LSA에 새로운 데이터를 추가하여 계산하려고 하면 보통 처음부터 다시 계산해야 함
    → 새로운 정보에 대해 업데이트가 어려움

- LDA(Latent Dirichlet Allocation)
  - LSA의 단점 보완
  - 문서들은 토픽들의 혼합으로 구성되어져 있으며, 토픽들은 **확률 분포**에 기반하여 단어들을 생성한다고 가정
    - 데이터가 주어지면, LDA는 문서가 생성되던 과정을 **역추적**
  - Count 기반의 Vectorizer만 적용

#6-1. 토픽 모델링(Topic Modeling)

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

### 모토사이클, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 의학, 우주 주제를 추출
cats = ['rec.motorcycles', 'rec.sport.baseball', 'comp.graphics', 'comp.windows.x',
        'talk.politics.mideast', 'soc.religion.christian', 'sci.electronics', 'sci.med'  ]

### 위에서 cats 변수로 기재된 category만 추출
# featch_20newsgroups( )의 categories에 cats 입력
news_df= fetch_20newsgroups(subset = 'all',remove = ('headers', 'footers', 'quotes'),
                            categories = cats, random_state = 0)

# LDA는 Count기반의 Vectorizer만 적용
count_vect = CountVectorizer(max_df = 0.95, max_features = 1000, min_df = 2, stop_words = 'english', ngram_range = (1,2))
feat_vect = count_vect.fit_transform(news_df.data) # 피처 벡터화
print('CountVectorizer Shape:', feat_vect.shape)
```

```
CountVectorizer Shape: (7862, 1000)
```

#6-1. 토픽 모델링(Topic Modeling)

```
### LDA 적용

lda = LatentDirichletAllocation(n_components = 8, random_state = 0) # 토픽 개수 조정(추출한 주제 개수에 맞도록)
lda.fit(feat_vect)
```

```
LatentDirichletAllocation(n_components=8, random_state=0)
```

```
### 속성값 확인

print(lda.components_.shape)
lda.components_
```

```
(8, 1000)
array([[3.60992018e+01, 1.35626798e+02, 2.15751867e+01, ...,
        3.02911688e+01, 8.66830093e+01, 6.79285199e+01],
       [1.25199920e-01, 1.44401815e+01, 1.25045596e-01, ...,
        1.81506995e+02, 1.25097844e-01, 9.39593286e+01],
       [3.34762663e+02, 1.25176265e-01, 1.46743299e+02, ...,
        1.25105772e-01, 3.63689741e+01, 1.25025218e-01],
       ...,
       [3.60204965e+01, 2.08640688e+01, 4.29606813e+00, ...,
        1.45056650e+01, 8.33854413e+00, 1.55690009e+01],
       [1.25128711e-01, 1.25247756e-01, 1.25005143e-01, ...,
        9.17278769e+01, 1.25177668e-01, 3.74575887e+01],
       [5.49258690e+01, 4.47009532e+00, 9.88524814e+00, ...,
        4.87048440e+01, 1.25034678e-01, 1.25074632e-01]])
```

EWHA EURON

#6-1. 토픽 모델링(Topic Modeling)

```
### 주제 추출 함수
def display_topics(model, feature_names, no_top_words):
    for topic_index, topic in enumerate(model.components_):
        print('Topic #',topic_index)

        # components_ array에서 가장 값이 큰 순으로 정렬했을 때, 그 값의 array index를 반환.
        topic_word_indexes = topic.argsort()[::-1]
        top_indexes = topic_word_indexes[:no_top_words]

        # top_indexes 대상인 index별로 feature_names에 해당하는 word feature 추출 후 join으로 concat
        feature_concat = ' '.join([feature_names[i] for i in top_indexes])
        print(feature_concat)
```

```
# CountVectorizer 객체 내의 전체 word들의 명칭을 get_features_names( )를 통해 추출
feature_names = count_vect.get_feature_names()

# Topic별 가장 연관도가 높은 word를 15개만 추출
display_topics(lda, feature_names, 15)
```

EWHA EURON

#6-1. 토픽 모델링(Topic Modeling)

```
Topic # 0
just time year don said like know didn bike good years got game think going
Topic # 1
image graphics jpeg dos software thanks color data images gif does format computer pc information
Topic # 2
god jesus church christ christian people believe christians bible faith sin paul man life team
Topic # 3
medical research health disease 1993 information patients cancer number hiv study use 10 april treatment
Topic # 4
file program window use server version display available windows ftp output code sun set motif
Topic # 5
don like just know think does people way make good problem want use time ve
Topic # 6
armenian people israel armenians jews turkish jewish israeli government war said turkey arab 000 armenia
Topic # 7
edu com 00 10 dos dos 12 24 pub mail cs 11 16 ac 3d 04
```

- Topic #0 : 일반적인 단어가 많음
- Topic #1 : 명확하게 컴퓨터 그래픽스 영역의 주제어가 추출됨
- Topic #2 : 기독교에 관련된 주제어
- Topic #3 : 의학에 관련된 주제어
- Topic #4 : 윈도우 운영체제와 관련된 주제어
- Topic #5 : 일반적인 단어
- Topic #6 : 중동 분쟁 등에 관련된 주제어
- Topic #7 : 윈도우 운영체제와 관련된 주제어(애매)

=> Topic #0, #5, #7에서 주로 애매한 주제어가 추출됐다.

7. 복습과제 소개: 감성분석 - 여성 의류 온라인 쇼핑몰 리뷰 데이터

EWHA EURON

#7.1 데이터 소개

데이터 다운로드: https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews

```
In [5]: df = pd.read_csv("Womens Clothing E-Commerce Reviews.csv", index_col=0)
        print(df.shape)
        df.head(3)
```

```
(23486, 10)
```

Out[5]:

	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	Class Name
0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmates	Intimate	Intimates
1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	Dresses
2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	Dresses

```
In [6]: df.groupby(['Rating', 'Recommended IND'])['Recommended IND'].count()
```

```
Out[6]: Rating  Recommended IND
        1       0                826
                1                 16
        2       0               1471
                1                 94
        3       0               1682
                1               1189
        4       0                168
                1               4909
        5       0                 25
                1              13106
        Name: Recommended IND, dtype: int64
```

<주요 데이터>

- Review Text: 분석 대상인 review 내용이 들어 있는 문자열 변수
- Rating: 고객이 부여한 제품 점수에 대한 양의 정수 변수
  1~5까지 있고 5에 가까울 수록 좋은 점수
- Recommended IND: 고객이 제품을 권장하는지를 나타내는 이진 변수
  1은 추천함 0은 추천하지 않음을 의미

EWHA EURON

#7.2 텍스트 전처리

- text merging

In [8]:
```
text_df = df[['Title', 'Review Text', 'Recommended IND']]
text_df.head()
```

Out[8]:

	Title	Review Text	Recommended IND
0	NaN	Absolutely wonderful - silky and sexy and comf...	1
1	NaN	Love this dress! it's sooo pretty. i happene...	1
2	Some major design flaws	I had such high hopes for this dress and reall...	0
3	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	1
4	Flattering shirt	This shirt is very flattering to all due to th...	1

In [9]:
```
text_df['Review'] = text_df['Title'] + ' ' + text_df['Review Text']
text_df = text_df.drop(labels=['Title','Review Text'] , axis=1)
text_df.head()
```

Out[9]:

	Recommended IND	Review
0	1	NaN
1	1	NaN
2	0	Some major design flaws I had such high hopes ...
3	1	My favorite buy! I love, love, love this jumps...
4	1	Flattering shirt This shirt is very flattering...

- target value

In [11]:
```
text_df = text_df[~text_df.Review.isna()]
text_df = text_df.rename(columns={"Recommended IND": "Recommended"})
print("My data's shape is:", text_df.shape)
text_df.head()
```

```
My data's shape is: (19675, 2)
```

Out[11]:

	Recommended	Review
2	0	Some major design flaws I had such high hopes ...
3	1	My favorite buy! I love, love, love this jumps...
4	1	Flattering shirt This shirt is very flattering...
5	0	Not for the very petite I love tracy reese dre...
6	1	Cagrcoal shimmer fun I aded this in my basket ...

In [12]:
```
text_df['Recommended'].unique()
```

Out[12]:
```
array([0, 1], dtype=int64)
```

In [13]:
```
text_df['Recommended'].value_counts(normalize=True)
```

Out[13]:
```
1    0.818297
0    0.181703
Name: Recommended, dtype: float64
```

추천 비율이 더 높음
Imbalance한 데이터

EWHA EURON

#7.2 텍스트 전처리

- adding feature
  1) Review_length: 리뷰의 길이에 관한 변수

```
In [14]: text_df['Review_length'] = text_df['Review'].apply(len)
         print(text_df.shape)
         text_df.head()
```

```
(19675, 3)
```

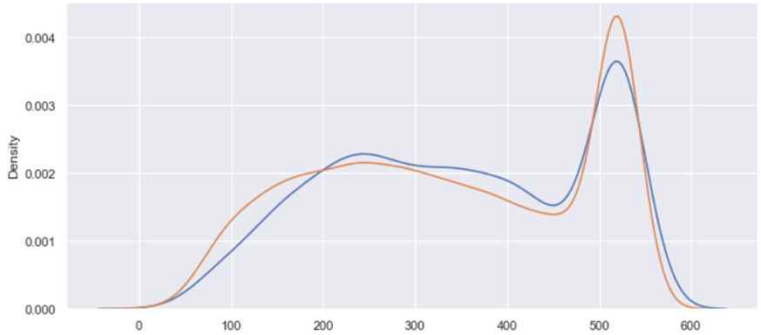
Out[14]:

	Recommended	Review	Review_length
2	0	Some major design flaws I had such high hopes ...	524
3	1	My favorite buy! I love, love, love this jumps...	141
4	1	Flattering shirt This shirt is very flattering...	209
5	0	Not for the very petite I love tracy reese dre...	512
6	1	Cagrcoal shimmer fun I aded this in my basket ...	517

```
In [17]: df_zero = text_df[text_df['Recommended']==0]
         df_one = text_df[text_df['Recommended']==1]
```

```
In [18]: sns.distplot(df_zero[['Review_length']] ,hist=False)
         sns.distplot(df_one[['Review_length']], hist=False)
```

```
Out[18]: <AxesSubplot:ylabel='Density'>
```



추천하는 리뷰와 추천하지 않는 리뷰간의 길이는 비슷한 분포를 지님

- 2) count_exc: 느낌표 개수에 관한 변수

```
In [19]: def count_exclamation_mark(string_text):
             count = 0
             for char in string_text:
                 if char == '!':
                     count += 1
             return count
```

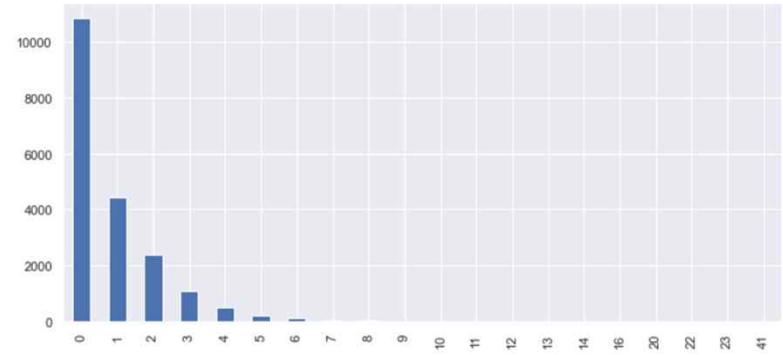
```
In [20]: text_df['count_exc'] = text_df['Review'].apply(count_exclamation_mark)
         text_df.head(5)
```

Out[20]:

	Recommended	Review	Review_length	count_exc
2	0	Some major design flaws I had such high hopes ...	524	1
3	1	My favorite buy! I love, love, love this jumps...	141	3
4	1	Flattering shirt This shirt is very flattering...	209	3
5	0	Not for the very petite I love tracy reese dre...	512	0
6	1	Cagrcoal shimmer fun I aded this in my basket ...	517	0

```
In [22]: text_df['count_exc'].value_counts().sort_index().plot(kind='bar')
```

```
Out[22]: <AxesSubplot:>
```



#7.2 텍스트 전처리

- 3) Polarity: TextBlob을 이용해서 극성 평가

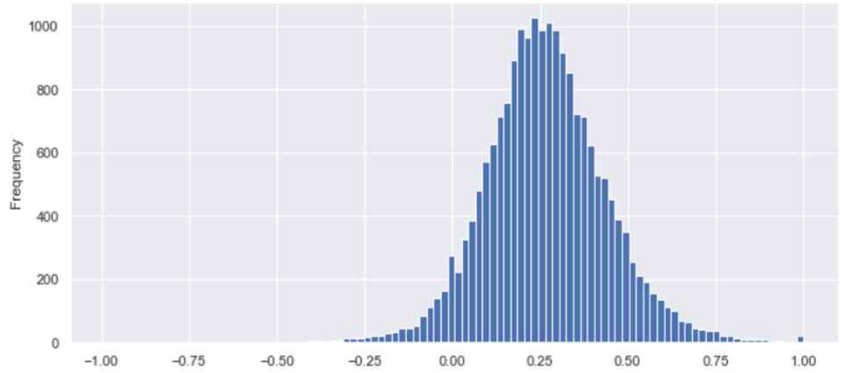
```
In [25]: text_df['Polarity'] = text_df['Review'].apply(lambda x: TextBlob(x).sentiment.polarity)
         text_df.head(5)
```

Out[25]:

	Recommended	Review	Review_length	count_exc	Polarity
2	0	Some major design flaws I had such high hopes ...	524	1	0.073209
3	1	My favorite buy! I love, love, love this jumps...	141	3	0.560714
4	1	Flattering shirt This shirt is very flattering...	209	3	0.512891
5	0	Not for the very petite I love tracy reese dre...	512	0	0.181111
6	1	Cagrcoal shimmer fun I aded this in my basket ...	517	0	0.157500

*polarity: 문장의 긍정, 부정의 정도에 따라 -1~1사이로 표현하는 것

```
In [26]: text_df['Polarity'].plot(kind='hist', bins=100)
Out[26]: <AxesSubplot:ylabel='Frequency'>
```



0.25를 중심으로 정규분포 형태로 분포됨
부정적인 표현보다는 긍정적인 표현이 많음

- drop punctuation

```
In [29]: def punctuation_removal(messy_str):
             clean_list = [char for char in messy_str if char not in string.punctuation]
             clean_str = ''.join(clean_list)
             return clean_str
```

```
In [30]: text_prep['Review'] = text_prep['Review'].apply(punctuation_removal)
         text_prep['Review'].head()
```

```
Out[30]: 2    Some major design flaws I had such high hopes ...
         3    My favorite buy I love love love this jumpsuit...
         4    Flattering shirt This shirt is very flattering...
         5    Not for the very petite I love tracy reese dre...
         6    Cagrcoal shimmer fun I aded this in my basket ...
         Name: Review, dtype: object
```

EWHA EURON

#7.2 텍스트 전처리

- stop word 제거

In [38]:
```
stop = stopwords.words('english')
stop.append("i'm")
```

In [39]:
```
stop_words = []

for item in stop:
    new_item = punctuation_removal(item)
    stop_words.append(new_item)
print(stop_words[::12])
```

```
['i', 'youd', 'hers', 'which', 'were', 'a', 'at', 'above', 'again', 'both', 'own', 'dont', 'aren', 'haven',
'shant']
```

- stemming

In [49]:
```
porter = PorterStemmer()
```

In [50]:
```
text_prep['Review'] = text_prep['Review'].apply(lambda x: x.split())
text_prep['Review'].head()
```

Out[50]:
```
2    [major, high, wanted, work, ordered, small, us...
3          [favorite, love, love, fabulous, get, great]
4    [flattering, flattering, due, adjustable, perf...
5    [petite, love, reese, petite, tall, long, full...
6    [aded, last, see, look, went, pale, gorgeous, ...
Name: Review, dtype: object
```

In [51]:
```
def stem_update(text_list):
    text_list_new = []
    for word in text_list:
        word = porter.stem(word)
        text_list_new.append(word)
    return text_list_new
```

In [52]:
```
text_prep['Review'] = text_prep['Review'].apply(stem_update)
text_prep['Review'].head()
```

Out[52]:
```
2    [major, high, want, work, order, small, usual,...
3          [favorit, love, love, fabul, get, great]
4    [flatter, flatter, due, adjust, perfect, pair,...
5    [petit, love, rees, petit, tall, long, full, o...
6    [ade, last, see, look, went, pale, gorgeou, tu...
Name: Review, dtype: object
```

In [53]:
```
text_prep['Review'] = text_prep['Review'].apply(lambda x: ' '.join(x))
text_prep['Review'].head()
```

Out[53]:
```
2    major high want work order small usual found s...
3                favorit love love fabul get great
4     flatter flatter due adjust perfect pair cardigan
5    petit love rees petit tall long full overwhelm...
6    ade last see look went pale gorgeou turn mathc...
Name: Review, dtype: object
```

wanted -> want
ordered -> order

EWHA EURON

#7.3 워드 클라우드

워드 클라우드: 자료의 빈도를 시각적으로 나타내는 시각화 방법 중 하나

```
In [55]: pos_df = text_prep[text_prep.Recommended== 1]
         neg_df = text_prep[text_prep.Recommended== 0]
         pos_df.head(3)
```

Recommended=1은 positive word
Recommended=0은 negative word로 분류

<Positive word 시각화>

```
In [57]: !pip install wordcloud
         from wordcloud import WordCloud

         wordcloud = WordCloud().generate(pos_words)

         wordcloud = WordCloud(background_color="white",max_words=len(pos_words),₩
                               max_font_size=40, relative_scaling=.5, colormap='summer').generate(pos_words)
         plt.figure(figsize=(13,13))
         plt.imshow(wordcloud)
         plt.axis("off")
         plt.show()
```

<Negative word 시각화>

```
In [58]: wordcloud = WordCloud().generate(neg_words)

         wordcloud = WordCloud(background_color="white",max_words=len(neg_words),₩
                               max_font_size=40, relative_scaling=.5, colormap='gist_heat').generate(neg_words)
         plt.figure(figsize=(13,13))
         plt.imshow(wordcloud)
         plt.axis("off")
         plt.show()
```

#7.4 텍스트 벡터화 - bow

```
In [61]: bow_transformer = CountVectorizer(text_vectorizing_process)

In [62]: bow_transformer.fit(text_prep['Review'])
Out[62]: CountVectorizer(input=<function text_vectorizing_process at 0x000001DB774D8310>)
```

- tf-idf 기반 벡터화

```
In [67]: tfidf_transformer = TfidfTransformer().fit(Reviews)

         tfidf_example = tfidf_transformer.transform(example)
         print (tfidf_example)
         #3507=Love
         #4438=petit
```

```
  (0, 7159)     0.18474832182622425
  (0, 6201)     0.2570248132639302
  (0, 6196)     0.2365171129379326
  (0, 5601)     0.1442941606293562
  (0, 5383)     0.355252429679757
  (0, 5004)     0.1888712992199643
  (0, 4875)     0.29314893108651 7
  (0, 4438)     0.4065409531496632
  (0, 4302)     0.3214748504588952
  (0, 3881)     0.3072123012448639
  (0, 3507)     0.21175884431125436
  (0, 3438)     0.20250236542769373
  (0, 2416)     0.2403172376641177
```

```
In [70]: messages_tfidf = tfidf_transformer.transform(Reviews)
         messages_tfidf.shape
Out[70]: (19675, 7277)

In [71]: print(messages_tfidf[:1])
         #tuple(index_num, word_num), tfidf_proba
```

```
  (0, 7273)     0.21472646647428087
  (0, 7159)     0.12712196996857394
  (0, 6930)     0.12017260570324308
  (0, 6813)     0.1513403250714934
  (0, 6383)     0.14060642619885752
  (0, 5601)     0.29785858576976637
  (0, 5290)     0.18845614450376966
  (0, 4438)     0.13986672876492434
  (0, 4260)     0.1585817815069749 7
  (0, 4210)     0.1001757163619286
  (0, 4138)     0.20901503445074096
  (0, 3944)     0.30397328049150685
  (0, 3584)     0.5250571838807988
```

EWHA EURON

#7.5 데이터 전처리

- X/y split, train/test split, scaling

In [74]:
```
X = df_all.drop('Recommended', axis=1)
y = df_all.Recommended

X.head()
```

In [77]:
```
X_train, X_test, y_train, y_test = split(X,y, test_size=0.3, stratify=y, random_state=111)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[77]: ((11536, 7280), (4945, 7280), (11536,), (4945,))

In [80]:
```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

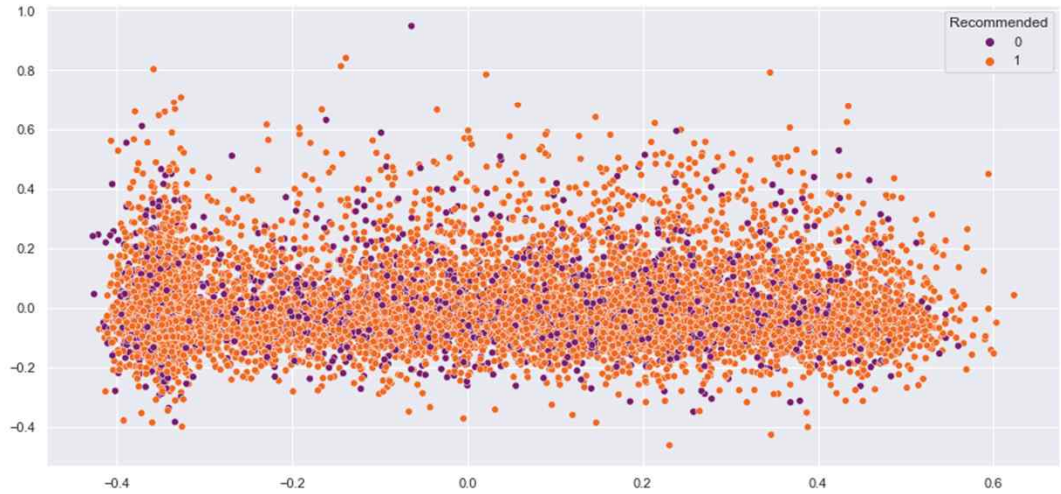
- PCA를 통해 Data visualization

In [82]:
```
pca_transformer = PCA(n_components=2).fit(X_train_scaled)
X_train_scaled_pca = pca_transformer.transform(X_train_scaled)
X_test_scaled_pca = pca_transformer.transform(X_test_scaled)
X_train_scaled_pca[:1]
```

Out[82]: array([[-0.13600042, -0.05000232]])

In [83]:
```
plt.figure(figsize=(15,7))
sns.scatterplot(x=X_train_scaled_pca[:, 0],
                y=X_train_scaled_pca[:, 1],
                hue=y_train,
                sizes=100,
                palette="inferno")
```

Out[83]: <AxesSubplot:>



#7.6 모델링 & 평가

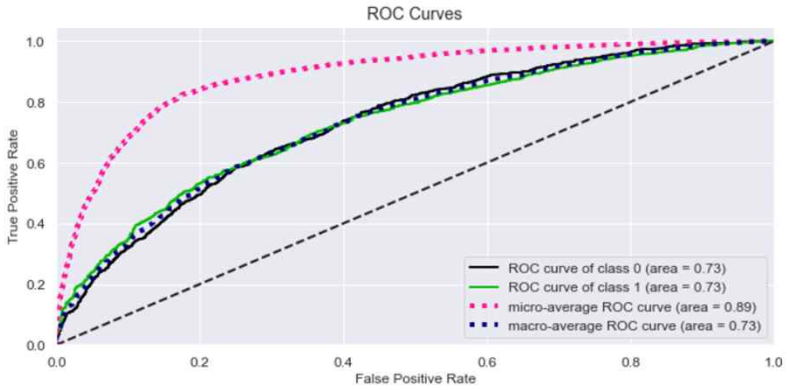
1. SVM (SVC)

```
In [86]: svc_model = SVC(C=1.0,
                         kernel='linear',
                         class_weight='balanced',
                         probability=True,
                         random_state=111)
         svc_model.fit(X_train_scaled, y_train)

Out[86]: SVC(class_weight='balanced', kernel='linear', probability=True,
             random_state=111)

In [87]: test_predictions = svc_model.predict(X_test_scaled)
         print(report(y_test, test_predictions, svc_model.classes_ ))
```

```
Confusion Matrix:
     0     1
0  517   380
1  995  3053

Classification Report:
              precision    recall  f1-score   support

           0       0.34      0.58      0.43       897
           1       0.89      0.75      0.82      4048

    accuracy                           0.72      4945
   macro avg       0.62      0.67      0.62      4945
weighted avg       0.79      0.72      0.75      4945
```

- F1 micro score=0.72

```
In [88]: skplt.metrics.plot_roc(y_test, svc_model.predict_proba(X_test_scaled))

Out[88]: <AxesSubplot:title={'center':'ROC Curves'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>
```



2. Logistic Regression

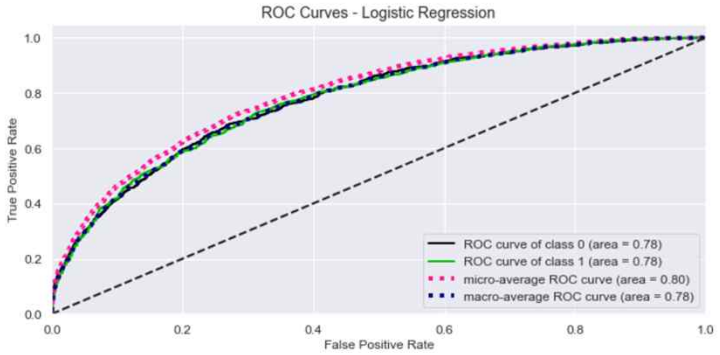
```
In [89]: lr_model = LogisticRegression(class_weight='balanced',
                                       random_state=111,
                                       solver='lbfgs',

         gs_lr_model = GridSearchCV

         gs_lr_model.fit(X_train_sc

Out[89]: GridSearchCV(cv=5,
                      estimator=Log

                      param_grid={'
```

```
In [91]: test_predictions = gs_lr_model.predict(X_test_scaled)
         print(report(y_test, test_predictions, gs_lr_model.classes_ ))
```

```
Confusion Matrix:
     0     1
0  617   280
1 1100  2948

Classification Report:
              precision    recall  f1-score   support

           0       0.36      0.69      0.47       897
           1       0.91      0.73      0.81      4048

    accuracy                           0.72      4945
   macro avg       0.64      0.71      0.64      4945
weighted avg       0.81      0.72      0.75      4945
```

- F1 micro score=0.72

```
In [92]: skplt.metrics.plot_roc(y_test, gs_lr_model.predict_proba(X_test_scaled),
                                title='ROC Curves - Logistic Regression')

Out[92]: <AxesSubplot:title={'center':'ROC Curves - Logistic Regression'}, xlabel='False Positive Rate', ylabel='True
         Positive Rate'>
```



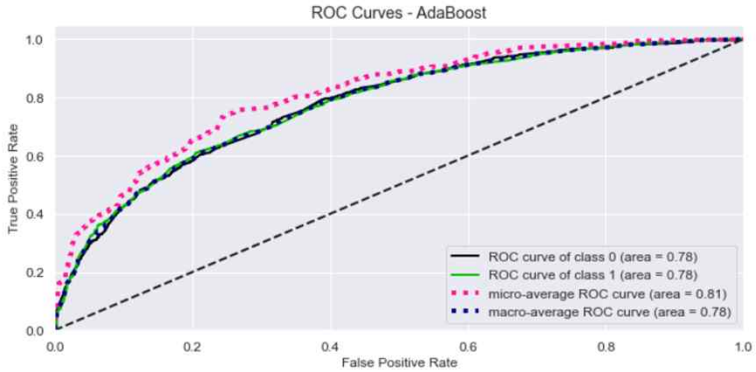
#7.6 모델링 & 평가

3. AdaBoost

```
In [93]: dt = DecisionTreeClassifier(max_depth=5, class_weight='balanced'

         ada_model = AdaBoostClassifier(base_estimator=dt, learning_rate=
         ada_model.fit(X_train ,y_train)

Out[93]: AdaBoostClassifier(base_estimator=DecisionTreeClassifier(class_w
                                                                  max_dep
                                                                  random_
                            learning_rate=0.001, n_estimators=1000, rando

In [94]: test_predictions = ada_model.predict(X_test)
         print(report(y_test, test_predictions, ada_model.classes_ ))

         Confusion Matrix:
              0     1
         0  549   348
         1  916  3132

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.37      0.61      0.46       897
                    1       0.90      0.77      0.83      4048

             accuracy                           0.74      4945
            macro avg       0.64      0.69      0.64      4945
         weighted avg       0.80      0.74      0.77      4945
```

- F1 micro score=0.74

```
In [95]: skplt.metrics.plot_roc(y_test, ada_model.predict_proba(X_test),
                                title='ROC Curves - AdaBoost')

Out[95]: <AxesSubplot:title={'center':'ROC Curves - AdaBoost'}, xlabel='False Positive Rate', ylabel='True Positive R
         ate'>
```

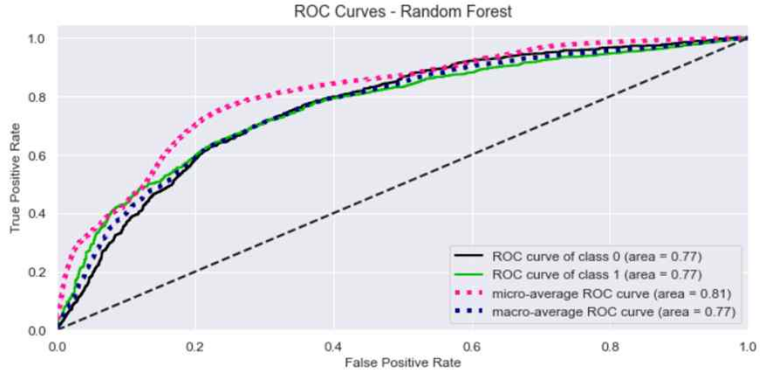


4. Randomforest

```
In [96]: rf_model = RandomForestClassifier(n_estimators=1000, max_dept
                                           class_weight='balanced', ra
         rf_model.fit(X_train, y_train)

Out[96]: RandomForestClassifier(class_weight='balanced', max_depth=5,
                                random_state=3)

In [97]: test_predictions = rf_model.predict(X_test)
         print(report(y_test, test_predictions, rf_model.classes_ ))

         Confusion Matrix:
              0     1
         0  535   362
         1  828  3220

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.39      0.60      0.47       897
                    1       0.90      0.80      0.84      4048

             accuracy                           0.76      4945
            macro avg       0.65      0.70
         weighted avg       0.81      0.76      0.78      4945
```

- F1 micro score=0.76

```
In [98]: skplt.metrics.plot_roc(y_test, rf_model.predict_proba(X_test),
                                title='ROC Curves - Random Forest')

Out[98]: <AxesSubplot:title={'center':'ROC Curves - Random Forest'}, xlabel='False Positive Rate', ylabel='True Posit
         ive Rate'>
```



8. 복습과제 소개: 토픽모델링 - 호텔 리뷰 데이터

EWHA EURON

#8.1 데이터 소개

데이터 다운로드: https://www.kaggle.com/datasets/andrewmvd/trip-advisor-hotel-reviews

In [6]:
```
data = pd.read_csv('tripadvisor_hotel_reviews.csv', encoding='utf-8')
data.head()
```

Out[6]:

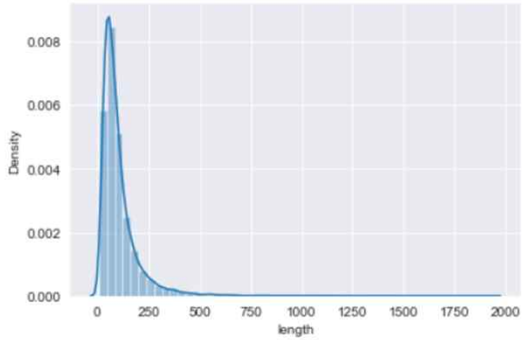
	Review	Rating
0	nice hotel expensive parking got good deal sta...	4
1	ok nothing special charge diamond member hilto...	2
2	nice rooms not 4* experience hotel monaco seat...	3
3	unique, great stay, wonderful time hotel monac...	5
4	great stay great stay, went seahawk game aweso...	5

#8.2 텍스트 전처리

https://wikidocs.net/21692

- adding feature: length

```
In [7]: data['length'] = data.Review.apply(lambda row: len(row.split()))
        print('Mean length: ', data['length'].mean())

        Mean length:  104.37582353228247

In [8]: import seaborn as sns
        sns.set_style(style="darkgrid")

        sns.distplot(data['length'])

Out[8]: <AxesSubplot:xlabel='length', ylabel='Density'>
```



LDA 모델에서는 짧은 리뷰를 안 좋게
평가하기 때문에 중요한 변수가 될 수 있음

- regular expressions 제거 & 토큰화

```
In [30]: data['review_list'] = data.Review.values.tolist()

         import re

         # remove characters
         data['review_list'] = [re.sub('₩s+', ' ', sent) for sent in data['review_list']]
         data['review_list'] = [re.sub("₩'", "", sent) for sent in data['review_list']]

         def sent_to_words(sentences):
             for sentence in sentences:
                 yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))

         tokens_reviews = list(sent_to_words(data['review_list']))
```

- N-grams 생성

```
In [31]: # create N-grams
         def make_n_grams(texts):
             bigram = gensim.models.Phrases(texts, min_count=5, threshold=100)  # higher threshold fewer phrases.
             bigram_mod = gensim.models.phrases.Phraser(bigram)
             trigram = gensim.models.Phrases(bigram[texts], threshold=100)
             trigram_mod = gensim.models.phrases.Phraser(trigram)
             bigrams_text = [bigram_mod[doc] for doc in texts]
             trigrams_text =  [trigram_mod[bigram_mod[doc]] for doc in bigrams_text]
             return trigrams_text
```

*N-gram: 언어모델 중 하나로, 뒤에 올 단어를 예측할 때 앞 단어 중
임의의 개수 (n개의 연속적인 단어)만 고려하는 방법

EWHA EURON

#8.2 텍스트 전처리

- lemmatization & stop word 제거

In [33]:
```
import spacy
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

# I use gensim stop-words and add me own stop-words, based on texts
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in gensim.parsing.preprocessing.STOPWORDS.union(set(['also', 'meanwhile','howe
                                                                                                                          'hour', 'soon', 'day', 'book
                                                                                                                          'there', 'hotel', 'room', 'l
                                                                                                                          'arrive',
                                                                                                                          'place', 'stay', 'staff', 'l
                                                                                                                         'service', 'come', 'check',
                                                                                                                         'ask', 'lot', 'thing',
                                                                                                                         'soooo', 'add', 'rarely',
                                                                                                                         'use', 'look', 'minute',
                                                                                                                         'bring', 'need', 'world',
                                                                                                                         'think', 'value', 'include'])
def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

In [34]:
```
# do lemmatization keeping only noun, vb, adv
# because adj is not informative for reviews topic modeling
reviews_lemmatized = lemmatization(tokens_reviews, allowed_postags=['NOUN', 'VERB', 'ADV'])

# remove stop words after lemmatization
reviews_lemmatized = remove_stopwords(reviews_lemmatized)
```

EWHA EURON

#8.3 모델링 - GSDMM

GSDMM: LDA의 확장된 버전으로 문서당 하나의 주제를 가정하기 때문에 짧은 텍스트 주제 모델링에 적합한 모델

클러스터 수를 자동으로 추론할 수 있고 텍스트의 희소하고 고차원적인 문제를 해결할 수 있음

- model fitting

```
In [92]: np.random.seed(0)

In [93]: mgp = MovieGroupProcess(K=6, alpha=0.01, beta=0.01, n_iters=30)

         vocab = set(x for review in reviews_lemmatized for x in review)
         n_terms = len(vocab)
         model = mgp.fit(reviews_lemmatized, n_terms)
```

```
In stage 0: transferred 15831 clusters with 6 clusters populated
In stage 1: transferred 8795 clusters with 6 clusters populated
In stage 2: transferred 5403 clusters with 6 clusters populated
In stage 3: transferred 4504 clusters with 6 clusters populated
In stage 4: transferred 3759 clusters with 6 clusters populated
In stage 5: transferred 3175 clusters with 6 clusters populated
In stage 6: transferred 2891 clusters with 6 clusters populated
In stage 7: transferred 2710 clusters with 6 clusters populated
In stage 8: transferred 2667 clusters with 6 clusters populated
In stage 9: transferred 2537 clusters with 6 clusters populated
In stage 10: transferred 2458 clusters with 6 clusters populated
In stage 11: transferred 2400 clusters with 6 clusters populated
```

K=최대 주제 개수
n_iters=반복횟수
-> 6개의 cluster가 생성됨

- 각 데이터 세트에 주제 할당

```
In [96]: topic_dict = {}
         topic_names = ['type 1',
                        'type 2',
                        'type 3',
                        'type 4',
                        'type 5',
                        'type 6',
                       ]
         for i, topic_num in enumerate(top_index):
             topic_dict[topic_num]=topic_names[i]
```

```
In [97]: def create_topics_dataframe(data_text=data.Review,  mgp=mgp, threshold=0.3, topic_dict=topic_dict, lemma_te
             result = pd.DataFrame(columns=['Text', 'Topic', 'Rating', 'Lemma-text'])
             for i, text in enumerate(data_text):
                 result.at[i, 'Text'] = text
                 result.at[i, 'Rating'] = data.Rating[i]
                 result.at[i, 'Lemma-text'] = lemma_text[i]
                 prob = mgp.choose_best_label(reviews_lemmatized[i])
                 if prob[1] >= threshold:
                     result.at[i, 'Topic'] = topic_dict[prob[0]]
                 else:
                     result.at[i, 'Topic'] = 'Other'
             return result
```

```
In [98]: result = create_topics_dataframe(data_text=data.Review, mgp=mgp, threshold=0.3, topic_dict=topic_dict, lemm
         result.head(5)
```

Out[98]:

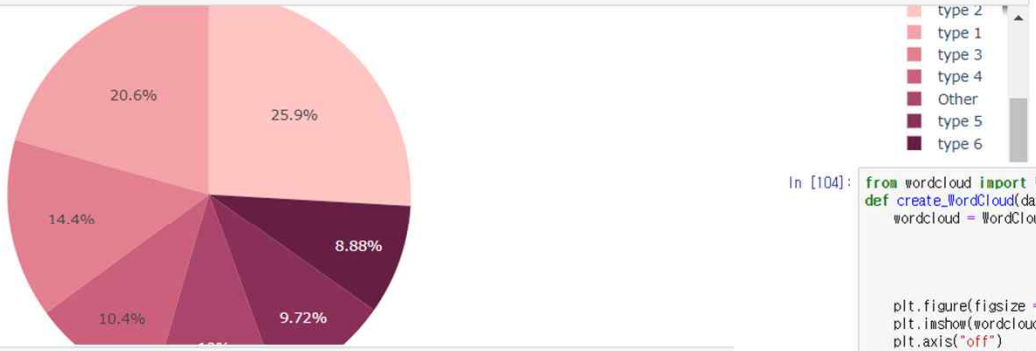
	Tex	Topic	Rating	Lemma-text
0	nice hotel expensive parking got good deal sta...	type 6	4	[parking, deal, anniversary, evening, advice, ...
1	ok nothing special charge diamond member hilto...	Other	2	[charge, diamond_member, decide, chain, shoot,...
2	nice rooms not 4* experience hotel monaco seat...	type 1	3	[experience, positive, bathroom, suite, bed, h...
3	unique, great stay, wonderful time hotel monac...	type 3	5	[stroll, downtown, shopping, area, pet, sign, ...

EWHA EURON

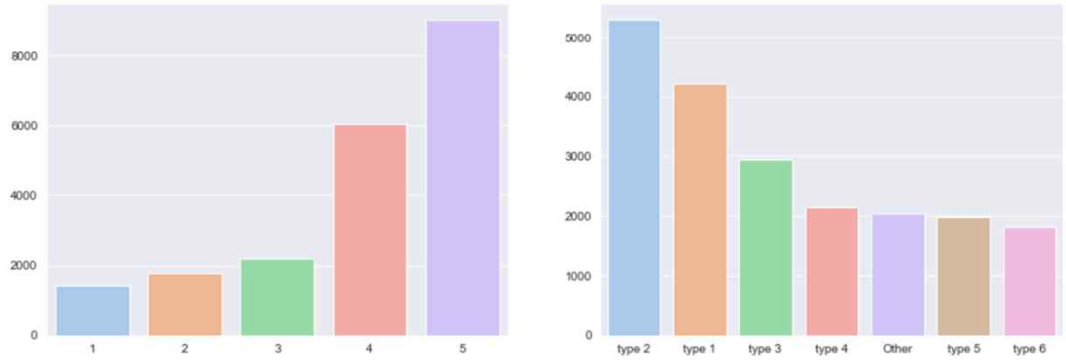
#8.3 모델링 - GSDMM 시각화

- Topic 비율 시각화

In [99]:
```
import plotly.express as px

fig = px.pie(result, names='Topic',  title='Topics', color_discrete_sequence=px.colors.sequential.Burg)
fig.show()
```



In [100]:
```
import matplotlib.pyplot as plt

rating_counts = result.Rating.value_counts()
types_counts = result.Topic.value_counts()
fig, ax = plt.subplots(1, 2, figsize=(15,5))
rating = sns.barplot(x = rating_counts.index, y = rating_counts.values, palette="pastel", ax=ax[0])
types = sns.barplot(x = types_counts.index, y = types_counts.values, palette="pastel", ax=ax[1])
```



- Word clouds

In [104]:
```
from wordcloud import WordCloud
def create_WordCloud(data, title=None):
    wordcloud = WordCloud(width = 500, height = 500,
                          background_color ='white',
                          min_font_size = 15
                         ).generate(" ".join(data.values))

    plt.figure(figsize = (5, 5), facecolor = None)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title(title,fontsize=20)
    plt.show()
```

In [105]:
```
create_WordCloud(result['Lemma_text'].loc[result.Topic == 'type 1'], title="Most used words in topic 1")
```

8.3 모델링 - LDA

- model fitting

In [106]:
```
from gensim import corpora
id2word = corpora.Dictionary(reviews_lemmatized)
texts = reviews_lemmatized
corpus = [id2word.doc2bow(text) for text in texts]
```

In [107]:
```
from gensim import models

tfidf = models.TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]
```

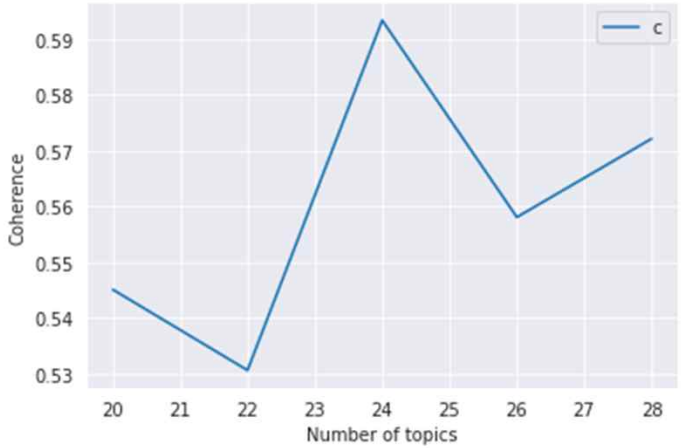
- coherence_values로 평가

In [108]:
```
from gensim.models.ldamulticore import LdaMulticore
from gensim.models.coherencemodel import CoherenceModel

def calc_coherence_values(dictionary, corpus, texts, limit = 12, start = 1, step = 1):
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = LdaMulticore(corpus=corpus,id2word = dictionary, num_topics = num_topics, alph
        model_list.append(model)
        print('model created')
        coherencemodel = CoherenceModel(model = model, texts = texts, dictionary = dictionary,
        print(coherencemodel.get_coherence())
        coherence_values.append(coherencemodel.get_coherence())
    return model_list, coherence_values

model_list, coherence_values = calc_coherence_values(dictionary = id2word, corpus = corpus_tfi
```

```
model created
0.31955594214985983
model created
0.31927718174424063
model created
0.3172096431107219
```

- coherece value 시각화

In [110]:
```
limit, start, step = 30, 20, 2
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Number of topics")
plt.ylabel("Coherence")
plt.legend(("coherence_values"), loc='best')
plt.show()
```



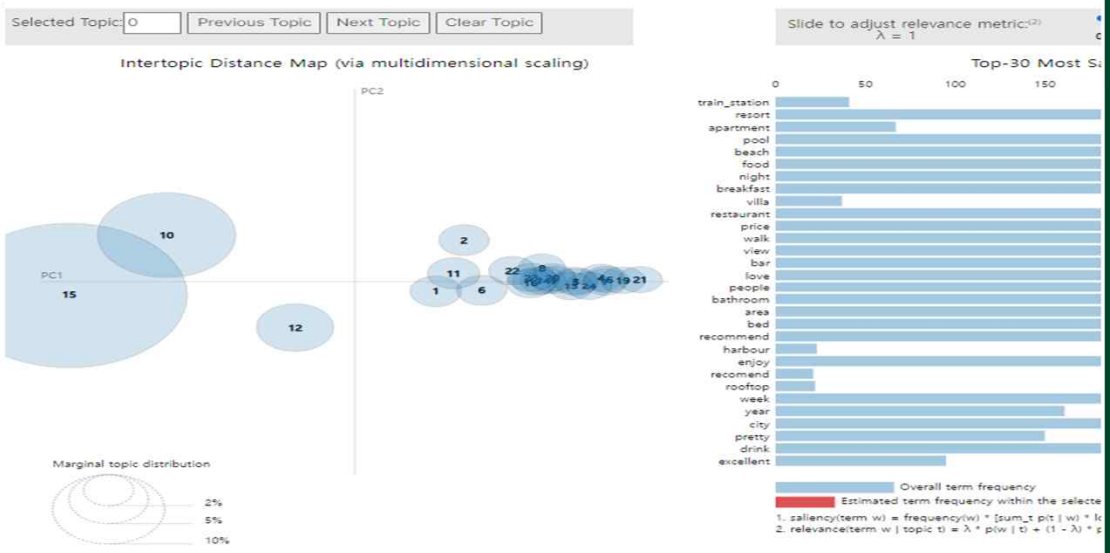
24가 가장 best topic

EWHA EURON

#8.3 모델링 - LDA

- 주제 시각화: pyLDAvis.gensim_models

In [82]:
```
!pip install pyLDAvis
import pyLDAvis.gensim_models

lda_display = pyLDAvis.gensim_models.prepare(model_list[index], corpus_tfidf, id2word
pyLDAvis.display(lda_display)
```



- 주요 주제, 기여도 (topic_perc_contrib), 키워드 출력

In [83]:
```
def format_topics_sentences(lda_model, corpus, data):
    sent_topics_df = pd.DataFrame()
    for i, row in enumerate(lda_model[corpus]):
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = lda_model.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(pr
            else:
                break
    sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']

    # Add original text to the end of the output
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)
```

Out[84]:

	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	Text
0	0	21.0	0.6552	resort, beach, night, walk, restaurant, pool, ...	[parking, deal, anniversary, evening, advice, ...
1	1	0.0	0.4911	pool, beach, resort, food, night, area, bed, b...	[charge, diamond_member, decide, chain, shoot,...
2	2	21.0	0.3915	resort, beach, night, walk, restaurant, pool, ...	[experience, positive, bathroom, suite, bed, h...
3	3	27.0	0.6623	view, night, walk, city, breakfast, restaurant...	[stroll, downtown, shopping, area, pet, sign, ...
4	4	9.0	0.5260	beach, food, restaurant, resort, night, pay, p...	[game, downfall, view, building, complain, web...

EWHA EURON

#8.3 모델링 - LSI

- model fitting & coherence_values로 평가

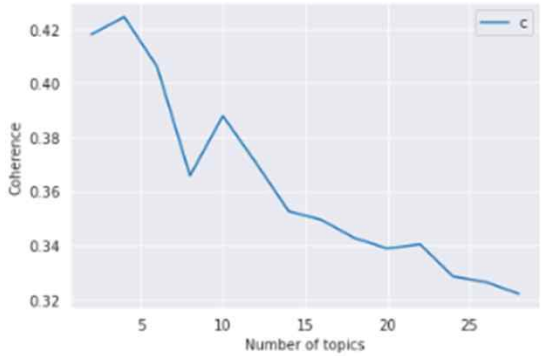
In [88]:
```
from gensim.models import LsiModel
```

In [89]:
```
def calc_coherence_values_Lsi(dictionary, corpus, texts, limit, start = 2, step = 2):
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = LsiModel(corpus=corpus, id2word = dictionary, num_topics = num_topics)
        print('model created')
        model_list.append(model)
        coherencemodel = CoherenceModel(model = model, texts = texts, dictionary = dictionary, coherence = 'c_v')
        print(coherencemodel.get_coherence())
        coherence_values.append(coherencemodel.get_coherence())
    return model_list, coherence_values


model_list, coherence_values_Lsi = calc_coherence_values_Lsi(dictionary = id2word, corpus=corpus_tfidf, texts=texts, start = 2, limit = 30, step =2)
```

```
model created
0.41399489526044675
model created
0.40535055326371
model created
0.3740852189148906
model created
0.3756386549220688
model created
0.36357356448443706
model created
0.36447667735404526
```

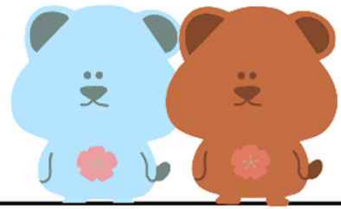
- coherece value 시각화

In [90]:
```
limit, start, step = 30, 2, 2
x = range(start, limit, step)
plt.plot(x, coherence_values_Lsi)
plt.xlabel("Number of topics")
plt.ylabel("Coherence")
plt.legend(("coherence_values"), loc='best')
plt.show()
```



4가 가장 best topic

THANK YOU



EWHA EURON