



12주차 발표

DA팀 김예진 박보영 오연재

목차

#01 텍스트 분석의 이해

#02 텍스트 전처리

#03 Bag of Words

#04 감성분석

#05 토픽모델링

#06 감성분석 노트북

#07 토픽모델링 노트북



01. 텍스트 분석의 이해



1.1 텍스트 분석

NLP (National Language Processing, 자연어처리)

- 머신이 인간의 언어를 이해하고 해석하는 데 더 중점

텍스트 분석

- 비정형 텍스트 데이터에서 의미 있는 정보를 추출하는 것에 더 중점

NLP task 사례

- 텍스트 요약 : 카카오 PORORO
- 기계 번역 : 구글번역, 네이버 파파고번역 등 + 딥딥, 클로버 등 음성번역
- 질의응답 (Question Answering, QA) : KorQuAD 2.0

1.2 NLP task 사례

텍스트 요약 : 카카오 PORORO

샘플 데이터 생성

```
input_text1 = """가수 김태연은 걸 그룹 소녀시대, 소녀시대-태티서 및 소녀시대-Oh!GG의 리더이자 메인보컬이다  
input_text2 = """목성과 토성이 약 400년 만에 가장 가까이 만났습니다. 국립과천과학관 등 천문학계에 따르면 2
```

샘플 데이터에 대하여 출력 값을 확인합니다.

```
print(input_text1)
```

```
'가수 김태연은 걸 그룹 소녀시대, 소녀시대-태티서 및 소녀시대-Oh!GG의 리더이자 메인보컬이다. 2004년 SM에서 주최한 청소년'
```

다음은 문서 요약(text summarization)에 대한 예제입니다.

```
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'
```

1.2 NLP task 사례

기계 번역 : 구글번역, 네이버 파파고번역 등 + 딥딥, 클로버 등 음성번역

The image shows two side-by-side screenshots of translation interfaces. On the left is Google Translate and on the right is Papago Translate.

Google Translate Interface:

- Top bar: ≡ Google 번역
- Input field: 텍스트 (Text) and 문서 (Document) options.
- Language selection: 언어 감지 (Language detection), 영어 (English), 한국어 (Korean), 독일어 (German), ↗ (Translate button), 한국어 (Korean), 영어 (English), 일본어 (Japanese).
- Text area: 번역 (Translation) placeholder.
- Bottom: 0 / 5,000, a microphone icon, and 의견 보내기 (Send feedback).
- Bottom navigation: 기록 (History), 저장된 번역 (Saved Translations), and 참여 (Participate).

Papago Translate Interface:

- Top bar: papago logo, 새창으로 띄우기 (Open in new window) button.
- Language selection: 영어 (English) and 한국어 (Korean).
- Text area: 번역할 내용을 입력하세요. (Enter text to translate). Includes audio, copy, and microphone icons.
- Bottom: 높임말 (Pronunciation) toggle switch and 선택된 텍스트 자동 번역 (Automatic translation of selected text) toggle switch.

1.2 NLP task 사례

질의응답 (Question Answering, QA) : KorQuAD 2.0

The screenshot shows the homepage of the KorQuAD 2.0 dataset. At the top, there is a dark header bar with the text "KorQuAD" on the left and "1.0 (ENG)" and "1.0 (한국어)" on the right. Below the header is a large teal banner with the text "KorQuAD 2.0" in white, bold letters, followed by "The Korean Question Answering Dataset". The main content area has two sections: "What is KorQuAD 2.0?" on the left and "Leaderboard" on the right. The "What is KorQuAD 2.0?" section contains a detailed description of the dataset, mentioning its size (over 20,000+ questions and answers), the source of documents (Wikipedia articles), and specific challenges like reading comprehension across multiple documents. It also includes two green buttons: "KORQUAD 2.0 소개 (SLIDE)" and "KORQUAD 2.0 소개 (PAPER)". The "Leaderboard" section displays a table of top-performing models based on Exact Match (EM) and F1 scores. The table includes columns for Rank, Reg. Date, Model, EM, and F1. The top entry is "Human Performance" from 2019.09.05, followed by "LittleBird-large (single model)" from 2022.03.08, and "SDS-NET v1.3 (single model)" from 2020.09.21.

KorQuAD

1.0 (ENG) 1.0 (한국어)

KorQuAD 2.0

The Korean Question Answering Dataset

What is KorQuAD 2.0?

KorQuAD 2.0은 KorQuAD 1.0에서 질문답변 20,000+ 쌍을 포함하여 총 100,000+ 쌍으로 구성된 한국어 Machine Reading Comprehension 데이터셋입니다. KorQuAD 1.0과는 다르게 1~2 문단이 아닌 Wikipedia article 전체에서 답을 찾아야 합니다. 매우 긴 문서들이 있기 때문에 탐색 시간에 대한 고려가 필요할 것 입니다. 또한 표와 리스트도 포함되어 있기 때문에 HTML tag를 통한 문서의 구조 이해도 필요합니다. 이 데이터셋을 통해서 다양한 형태와 길이의 문서들에서도 기계독해가 가능해질 것 입니다.

KORQUAD 2.0 소개 (SLIDE)

KORQUAD 2.0 소개 (PAPER)

Leaderboard

KorQuAD 2.0의 Test set으로 평가한 Exact Match(EM) 및 F1 score입니다.

Rank	Reg. Date	Model	EM	F1
-	2019.09.05	Human Performance	68.82	83.86
1	2022.03.08	LittleBird-large (single model) KakaoEnterprise - Minchul Lee, DongHyun Choi, Seung Woo Cho, Ae Lim Ahn	78.70	90.22
2	2020.09.21	SDS-NET v1.3 (single model) Samsung SDS AI Research	77.86	89.82

1.3 피처 벡터화

텍스트 = 비정형 데이터

머신러닝 알고리즘 = 숫자형의 피처 기반 데이터만 사용 가능

→ 비정형 텍스트 데이터를 어떻게 피처 형태로 추출, 추출된 피처에 의미 있는 값을 부여하는지가 중요!!!

피처 벡터화 (or 피처 추출)

- BoW (Bag of Words)
- Word2Vec
- GloVe
- FastText

1.3 피처 벡터화

Word2Vec

- 단어의 의미에 따라 벡터 공간에 projecting
- 비슷한 의미를 가진 단어들끼리 모이고, 다른 의미를 가진 단어들끼리 멀어지도록

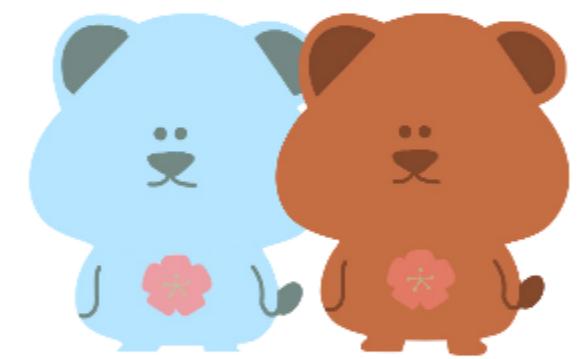
GloVe

- Word2Vec과 마찬가지로 단어들을 벡터 공간으로
- Local한 문맥에만 의존하는 것이 아니라 word co-occurrence를 바탕으로 global한 의미 고려

FastText

- Modified version of Word2Vec

02. 텍스트 전처리



2.1 텍스트 토큰화

문장 토큰화

- 문장의 마침표(.), 개행문자(\n) 등의 기호에 따라 분리
- NTLK의 sent_tokenize()

```
from nltk import sent_tokenize

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)

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

```

2.1 텍스트 토큰화

단어 토큰화

- 공백, 콤마(,), 마침표(.). 개행문자 등으로 분리
- 각 문장이 가지는 문맥이 중요하지 않은 경우 단어의 순서가 의미가 없기 때문에 문장 토큰화 없이도 충분
- NLTK의 word_tokenize()

```
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)

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

2.1 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)

<class 'list'> 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', 'fee',
l', 'it', 'when', 'you', 'go', 'to', 'work', ',', 'or', 'go', 'to', 'church', 'or', 'pay', 'your', 'taxes',
'.']]
```

2.1 텍스트 토큰화

정규표현식

- 정규표현식 모듈 re (import re)
- 특정 규칙이 있는 텍스트 데이터를 빠르게 정제할 수 있다

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

2.1 텍스트 토큰화

.기호: 1개의 임의의 문자

ex) 정규표현식 a.c

```
r = re.compile("a.c")
```

```
r.search("kkk")
```

```
r.search("abc")
```

```
<re.Match object; span=(0, 3), match='abc'>
```

+기호: 바로 앞의 문자가 1개 이상

ex) 정규표현식 ab+c

```
r = re.compile("ab+c")
```

```
r.search("ac")
```

```
r.search("abc")
```

```
<re.Match object; span=(0, 3), match='abc'>
```

```
r.search("abbbbc")
```

```
<re.Match object; span=(0, 6), match='abbbbc'>
```

2.1 텍스트 토큰화

정규표현식을 이용한 토큰화 및 텍스트 전처리

```
text = """100 John    PROF  
101 James   STUD  
102 Mac     STUD"""
```

re.split('\s+', text) **공백**

```
['100', 'John', 'PROF', '101', 'James', 'STUD', '102', 'Mac', 'STUD']
```

re.findall('\d+', text) **숫자**

```
['100', '101', '102']
```

re.findall('[A-Z][a-z]+', text) **대문자 + 소문자 여러번**

```
['John', 'James', 'Mac']
```

NLTK의 RegexpTokenizer()

```
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  
  
tokenizer1 = RegexpTokenizer("[\w]+")  
tokenizer2 = RegexpTokenizer("\s+", gaps=True)  
  
print(tokenizer1.tokenize(text))  
print(tokenizer2.tokenize(text))
```

문자 또는 숫자

공백

```
['Don', 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'Mr', 'Jone', 's', 'Orphanage', 'is', 'a  
s', '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']
```

2.2 불용어 제거

불용어 (Stop word) 제거

- 불용어 = 분석에 큰 의미가 없는 단어 (ex. Is, the, a, will 등)
- NLTK는 다양한 언어의 stop word를 제공한다

```
import nltk

stopwords = nltk.corpus.stopwords.words('english')
all_tokens = []
# 위 예제의 3개의 문장별로 얻은 word_tokens list에 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', 'pay', 'taxes', '.']]
```

2.3 어근 추출

어근 추출

- 문법적 또는 의미적으로 변화하는 단어의 원형 찾기

Stemming

- 단순화된 방법을 사용하여 원형 단어에서 일부 철자가 훼손된 단어 추출
- NLTK가 제공하는 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
```

2.3 어근 추출

Lemmatization

- 문법적 요소와 의미적인 부분 감안하여 정확한 철자의 어근 단어 추출
- 단어의 품사를 입력해줘야 함
- NLTK가 제공하는 WordNetLemmatizer

```
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'))
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]      C:\Users\lovel\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

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

2.4 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

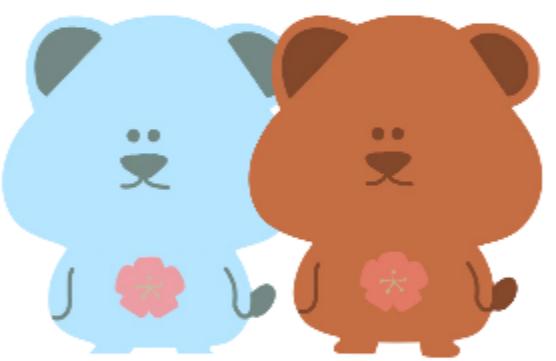
SOYNLP

- 품사 태깅, 단어 토큰화 등을 지원
- 비지도학습으로 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) ‘은경이는’ → ‘은’, ‘경이’, ‘는’으로 토큰화되는 문제를
twitter.add_dictionary(‘은경이’, ‘Noun’) 사전 추가를 통해 해결 → ‘은경이’, ‘는’

03. BoW



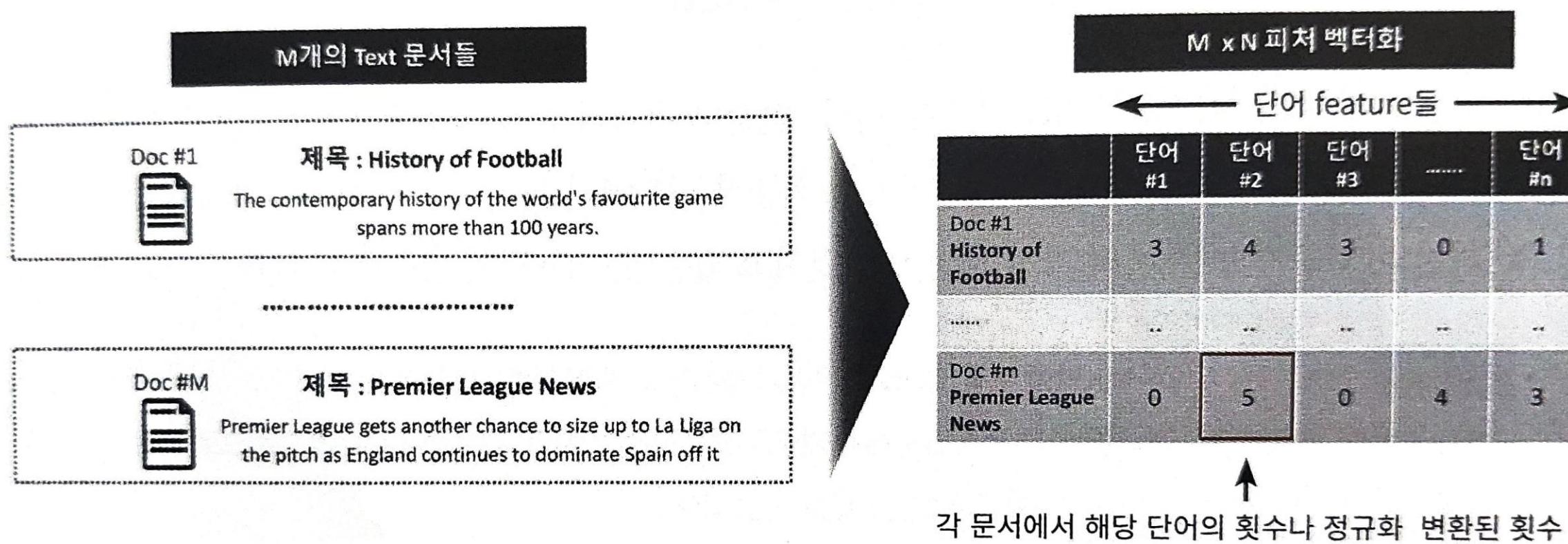
3.1 BoW (Bag of Words)

Bag of Words 모델

- 문서가 가지는 모든 단어를 문맥이나 순서를 무시하고 일괄적으로 단어에 대한 빈도 값을 부여해 피처 값 추출
- 장점: 쉽고 빠른 구축
- 단점: 문맥 의미 반영 부족, 희소 행렬 문제

BoW 피처 벡터화

- 모든 문서에서 모든 단어를 칼럼 형태로 나열
→ 각 문서에서의 해당 단어 횟수나 정규화된 빈도를 값으로 부여 = 데이터 세트 모델로 변경



3.2 카운트 벡터화

카운트 벡터화

- 각 문서에서 해당 단어가 나타나는 횟수로 단어 피처에 값 부여
- 카운트 값이 높을수록 중요한 단어로 인식
- 언어의 특성상 문서에서 자주 사용될 수 밖에 없는 단어에도 높은 값 부여하는 문제 발생

사이킷런의 CountVectorizer 클래스

max_df	너무 높은 빈도수를 가지는 단어 피처 제외
min_df	너무 낮은 빈도수를 가지는 단어 피처 제외
max_features	추출하는 피처의 개수를 제한
stop_words	추출에서 제외할 단어를 스톱 워드로 지정
n_gram_range	N-gram 기법 사용할 범위 설정. 튜플 형태로 (minimum, maximum)
anlayzer	피처 추출 수행할 단위 지정. 디폴트 = word
token_pattern	토큰화 진행하는 정규 표현식 패턴. 디폴트 변경할 경우 거의 X.
tokenizer	토큰화를 별도의 커스텀 함수로 이용 시 적용
binary	True로 지정 시 개수 카운트가 아니라 단어가 있는지 없는지 여부를 1, 0으로

3.2 카운트 벡터화

```
from sklearn.feature_extraction.text import CountVectorizer

sample_bow = CountVectorizer()

sample_corpus = ["the cat sat", "the cat sat in the hat", "the cat with the hat"]

sample_bow.fit(sample_corpus)

def get_bow_representation(text):
    return sample_bow.transform(text)

print(f"Vocabulary mapping for given sample corpus : \n {sample_bow.vocabulary_}")
print("\nBag of word Representation of sentence 'the cat cat sat in the hat'")
print(get_bow_representation(["the cat cat sat in the hat"]).toarray())

Vocabulary mapping for given sample corpus :
{'the': 4, 'cat': 0, 'sat': 3, 'in': 2, 'hat': 1, 'with': 5}

Bag of word Representation of sentence 'the cat cat sat in the hat'
[[2 1 1 1 2 0]]
```

```
# Bag of 2-gram (bigram)
from sklearn.feature_extraction.text import CountVectorizer

sample_boN = CountVectorizer(ngram_range = (2, 2))

sample_corpus = ["the cat sat", "the cat sat in the hat", "the cat with the hat"]

sample_boN.fit(sample_corpus)

def get_boN_representation(text):
    return sample_boN.transform(text)

print(f"Bigram Vocabulary mapping for given sample corpus : \n {sample_boN.vocabulary_}")
print("\nBag of 2-gram (bigram) Representation of sentence 'the cat cat sat in the hat'")
print(get_boN_representation(["the cat cat sat in the hat"]).toarray())

Bigram Vocabulary mapping for given sample corpus :
{'the cat': 4, 'cat sat': 0, 'sat in': 3, 'in the': 2, 'the hat': 5, 'cat with': 1, 'with the': 6}

Bag of 2-gram (bigram) Representation of sentence 'the cat cat sat in the hat'
[[1 0 1 1 1 1 0]]
```

3.3 TF-IDF

TF-IDF 벡터화

- Term Frequency – Inverse Document Frequency
- 개별 문서에서 자주 나타나는 단어에 높은 가중치 → 해당 문서를 특징 짓는 중요 단어
- 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 페널티 → 언어 특성상 범용적으로 자주 사용되는 단어

TF

- 개별 문서에서 해당 단어의 빈도

$$TF(t, d) = \frac{\text{(Number of occurrences of term } t \text{ in document } d)}{\text{(Total number of terms in the document } d)}$$

IDF

- 해당 단어의 중요도

$$IDF(t) = \log_e \frac{\text{(Total number of documents in the corpus } N)}{\text{(Number of documents with term } t \text{ in them } DF)}$$

3.3 TF-IDF

“the car is driven on the road”

“the truck is driven on the highway”

Word	TF		IDF	TF * IDF	
The	2/7	2/7	$\log(2/2) = 0$	0	0
car	1/7	0	$\log(2/1) = 0.3$	0.043	0
Truck	0	1/7	$\log(2/1) = 0.3$	0	0.043
Is	1/7	1/7	$\log(2/2) = 0$	0	0
Driven	1/7	1/7	$\log(2/2) = 0$	0	0
On	1/7	1/7	$\log(2/2) = 0$	0	0
Road	1/7	0	$\log(2/1) = 0.3$	0.043	0
highway	0	1/7	$\log(2/1) = 0.3$	0	0.043

3.3 TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

sample_corpus = ["the car is driven on the road", "the truck is driven on the highway"]
tfidf_rep = tfidf.fit_transform(sample_corpus)
print(f"Vocabulary mapping for given sample corpus : \n {tfidf.vocabulary_}")
print(f"IDF Values for sample corpus : {tfidf.idf_}")

print("TF-IDF Representation for sentence 'the car is driven on the road' :")
print(tfidf.transform(["the car is driven on the road"]).toarray())
```

Vocabulary mapping for given sample corpus :

```
{'the': 6, 'car': 0, 'is': 3, 'driven': 1, 'on': 4, 'road': 5, 'truck': 7, 'highway': 2}
```

IDF Values for sample corpus : [1.40546511 1. 1.40546511 1. 1. 1.40546511
1. 1.40546511]

TF-IDF Representation for sentence 'the car is driven on the road' :

```
[[0.42471719 0.30218978 0.          0.30218978 0.30218978 0.42471719  
 0.60437955 0.          0.          ]]
```

3.4 회소행렬

회소행렬

- 대규모의 칼럼으로 구성된 행렬에서 대부분의 값이 0으로 채워지는 행렬
- 문서마다 서로 다른 단어로 구성 → 매우 많은 단어의 개수 / 하나의 문서에 있는 단어는 이 중 극히 일부
- 머신러닝 알고리즘의 수행 시간과 예측 성능 떨어뜨림 → 회소행렬을 위한 기법 필요 (COO 형식, CSR 형식)

밀집행렬

- 반대로 대부분의 값이 0이 아닌 의미 있는 값으로 채워져 있는 행렬

3.4 회소행렬

COO 형식

Dense 형식의 원본 데이터

```
[ [0,0,1,0,0,5],  
[1,4,0,3,2,5],  
[0,6,0,3,0,0],  
[2,0,0,0,0,0],  
[0,0,0,7,0,8],  
[1,0,0,0,0,0]]
```

0이 아닌 데이터 값 배열

```
[1, 5, 1, 4, 3, 2, 5, 6, 3, 2, 7, 8, 1]  
  
→ 0이 아닌 데이터 값의 행과 열 위치  
(0, 2), (0, 5)  
(1, 0), (1, 1), (1, 3), (1, 4), (1, 5)  
(2, 1), (2, 3)  
(3, 0)  
(4, 3), (4, 5)  
(5, 0)
```

행 위치 배열

```
[0, 0, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5]
```

열 위치 배열

```
[2, 5, 0, 1, 3, 4, 5, 1, 3, 0, 3, 5, 0]
```

```
dense = np.array([[0,0,1,0,0,5],  
[1,4,0,3,2,5],  
[0,6,0,3,0,0],  
[2,0,0,0,0,0],  
[0,0,0,7,0,8],  
[1,0,0,0,0,0]])
```

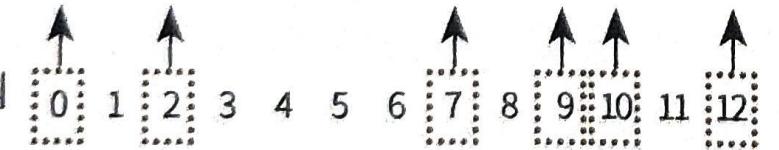
```
coo = sparse.coo_matrix(dense)  
csr = sparse.csr_matrix(dense2)  
  
print(coo.toarray())  
print(csr.toarray())
```

CSR 형식

행 위치 배열

```
[0, 0, 1, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5]
```

행 위치 배열의
인덱스



행 위치 배열의 고유값
시작 인덱스 배열

```
[0, 2, 7, 9, 10, 12]
```



```
[13]
```

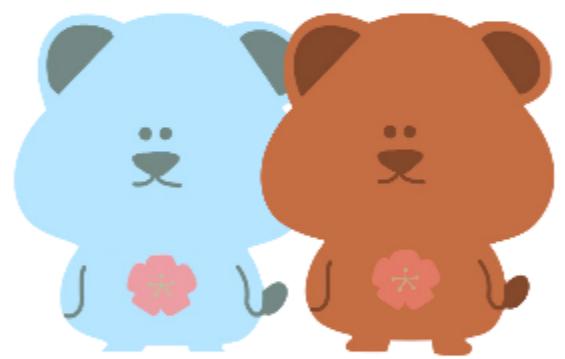
총 항목 개수 배열

```
[0, 2, 7, 9, 10, 12, 13]
```

행 위치 배열의 고유값
시작 인덱스 배열 최종

```
[[0 0 1 0 0 5]  
[1 4 0 3 2 5]  
[0 6 0 3 0 0]  
[2 0 0 0 0 0]  
[0 0 0 7 0 8]  
[1 0 0 0 0 0]]  
  
[[0 0 1 0 0 5]  
[1 4 0 3 2 5]  
[0 6 0 3 0 0]  
[2 0 0 0 0 0]  
[0 0 0 7 0 8]  
[1 0 0 0 0 0]]
```

04. 20 뉴스그룹 분류



4. 20 뉴스그룹 분류

5.6.2. The 20 newsgroups text dataset

The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics split in two subsets: one for training (or development) and the other one for testing (or for performance evaluation). The split between the train and test set is based upon messages posted before and after a specific date.

This module contains two loaders. The first one, `sklearn.datasets.fetch_20newsgroups`, returns a list of the raw texts that can be fed to text feature extractors such as `sklearn.feature_extraction.text.CountVectorizer` with custom parameters so as to extract feature vectors. The second one, `sklearn.datasets.fetch_20newsgroups_vectorized`, returns ready-to-use features, i.e., it is not necessary to use a feature extractor.

https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html



진행 순서

텍스트 정규화 ► 피처 벡터화 (Count, TF-IDF) 및 성능 비교 ► 희소행렬형태 (*) ► 로지스틱회귀 분류 수행 ► Pipeline, GridSearchCV

희소행렬 분류를 잘 처리할 수 있는 알고리즘은(*) [로지스틱회귀](#), [선형서포트벡터머신](#), [나이브베이즈](#)

4. 20 뉴스그룹 분류

희소행렬 분류를 잘 처리할 수 있는 알고리즘은 로지스틱회귀, 선형서포트벡터머신, 나이브베이즈

나이브베이즈(naïve bayse classifier)

<https://wikidocs.net/22892>

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad P(A|B) = \frac{P(A \cap B)}{P(B)}$$

데이터가 각 클래스에 속할 특징 확률을 계산하는 조건부 확률 기반의 분류 방법

데이터셋의 모든 특징들이 독립적이라고 가정 (=naïve)

Ex) 비 내리는 날에는 ‘시간’ 보다 ‘습도’ 가 더 중요할 수 있지만 이를 무시
단어의 순서가 중요하지 않기 때문에 무시하고 **빈도수**만 고려한다(bow)

장점: 데이터 크기에 상관없이 잘 작동하며 빠르고 효율적이다

단점: 가정의 오류. 수치 특징이 많은 데이터셋에는 성능이 떨어진다

사용처: 스팸 필터링, 질병 진단

```
from sklearn.naive_bayes import MultinomialNB  
  
mod = MultinomialNB()  
mod.fit(X_train_cnt_vect, y_train)  
pred=mod.predict(X_test_cnt_vect)  
print("정확도:", accuracy_score(y_test, pred))
```

정확도: 0.5431492299522039

4.1. 텍스트 정규화

```
from sklearn.datasets import fetch_20newsgroups  
  
news_data=fetch_20newsgroups(subset='all', random_state=156)  
  
print(news_data.keys())  
  
dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])  
  
target 클래스의 값과 분포도  
0    799  
1    973  
2    985  
3    982  
4    963  
5    988  
6    975  
7    990  
8    996  
9    994  
10   999  
11   991  
12   984  
13   990  
14   987  
15   997  
16   910  
17   940  
18   775  
19   628  
dtype: int64  
  
target 클래스의 이름들  
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',  
'comp.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.basi-  
c.space', 'soc.religion.christian', 'talk.politics.guns', 't-']
```

```
#개별 데이터가 텍스트로 어떻게 구성돼 있는지 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
```

뉴스그룹 제목, 작성자, 소속, 이메일,
뉴스그룹 기사 내용 등...

```
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...
```

순수한 텍스트만으로 구성된 기사내용으로 어떤 뉴스그룹에 속하는지 분류할 것이기 때문에 나머지 삭제해야 한다(header, footer 등)

4.1. 텍스트 정규화

```
from sklearn.datasets import fetch_20newsgroups  
  
train_news=fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'), random_state=156)  
X_train=train_news.data  
y_train=train_news.target  
  
test_news=fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'), random_state=156)  
X_test=test_news.data  
y_test=test_news.target  
  
print('train data size {0}, test data size {1}'.format(len(train_news.data), len(test_news.data)))  
  
train data size 11314, test data size 7532
```

subset : {'train', 'test', 'all'}, default='train'

remove : tuple, default=()

Fetch_20newsgroups()의 **subset**파라미터를 이용해 학습/테스트 데이터 세트 분리
Remove 파라미터를 이용해 header, footer, quote 삭제

4.2. 피처 벡터화 변환과 ML모델 학습/예측/평가

CountVectorizer

학습 데이터만을 이용해 fit()이 수행된 countvectorizer객체를 이용해 테스트 데이터를 변환해야 한다
이는 설정된 피처개수와 변환할 피처개수 동일시하기 위해서이다. 따라서 fit_transform()사용 불가.

```
from sklearn.feature_extraction.text import CountVectorizer

cnt_vect=CountVectorizer()
cnt_vect.fit(X_train)
X_train_cnt_vect=cnt_vect.transform(X_train)

X_test_cnt_vect=cnt_vect.transform(X_test)

print('학습 데이터 텍스트의 countVectorizer shape: ', X_train_cnt_vect.shape)
```

학습 데이터 텍스트의 countVectorizer shape: (11314, 101631)

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

lr_clf=LogisticRegression(max_iter=500)
lr_clf.fit(X_train_cnt_vect, y_train)
pred=lr_clf.predict(X_test_cnt_vect)
print('countvectorized logistic regression의 예측 정확도는 {:.3f}'.format(accuracy_score(y_test, pred)))
```

countvectorized logistic regression의 예측 정확도는 0.597

4.2. 피처 벡터화 변환과 ML모델 학습/예측/평가

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect=TfidfVectorizer()
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)
pred=lr_clf.predict(X_test_tfidf_vect)
print('TF-IDF logistic regression의 예측 정확도는 {:.3f}'.format(accuracy_score(y_test, pred)))
```

TF-IDF logistic regression의 예측 정확도는 0.674

TF-IDF(with parameter)

Stop-words, ngram_range, max_df

stop_words : {'english'}, list, default=None

ngram_range : tuple (min_n, max_n), default=(1, 1)

max_df : float or int, default=1.0

```
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)

pred=lr_clf.predict(X_test_tfidf_vect)
print('TF-IDF vectorized logistic regression 예측 정확도는 {:.3f}'.format(accuracy_score(y_test, pred)))
```

TF-IDF vectorized logistic regression 예측 정확도는 0.692

4.2. 피처 벡터화 변환과 ML모델 학습/예측/평가

로지스틱 회귀의 최적의 C값을 찾은 후 학/예/평

```
from sklearn.model_selection import GridSearchCV  
  
params={'C':[0.01,0.1,1,5,10]}  
  
lr_clf=LogisticRegression(max_iter=200)  
lr_clf.fit(X_train_tfidf_vect, y_train)  
  
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_)  
  
pred=grid_cv_lr.predict(X_test_tfidf_vect)  
print('TF-IDF vectorized logistic regression 예측 정확도는 {:.3f}'.format(accuracy_score(y_test, pred)))
```

Best c parameter c=10
Accuracy_score=0.704

4.3. Pipeline & GridSearchCV

Pipeline을 이용해 피처벡터화와 ML 알고리즘 학습/예측 한번에 진행



Pipeline- TF-IDF, LogisticRegression

```
from sklearn.pipeline import Pipeline
pipeline=Pipeline([('tfidf_vect', TfidfVectorizer(stop_words='english')), ('ngram_range', ngram_range(1,2)), ('max_df', max_df=300), ('lr_clf', LogisticRegression(C=10))])
pipeline.fit(X_train, y_train)
pred=pipeline.predict(X_test)
print('pipeline 을 통한 logistic regression의 예측정확도는 {:.3f}'.format(accuracy_score(y_test, pred)))
```

Accuracy_score=0.704

GridSearchCV에 Pipeline 입력- TF-IDF, LogisticRegression 함께 최적화 - 시간오래걸린다

```
from sklearn.pipeline import Pipeline
pipeline=Pipeline([('tfidf_vect', TfidfVectorizer(stop_words='english')), ('lr_clf', LogisticRegression())])
params={'tfidf_vect_ngram_range': [(1,1),(1,2),(1,3)],
        'tfidf_vect_max_df': [100,300,700],
        'lr_clf_C':[1,5,10]}
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('pipeline 을 통한 logistic regression의 예측정확도는 {:.3f}'.format(accuracy_score(y_test, pred)))
```

Lr_clf_C=10
Tfidf_vect_max_df=700
Tfidf_vect_ngram_range=(1,2)
Grid_cv_pipe.best_score_=0.755
Pieline logistic regression accuracy=0.702

05. 감성분석



5. 감성분석(sentiment Analysis)

감성분석이란 텍스트의 정보를 추출하는 텍스트 마이닝과는 달리, 어떤 주제에 대한 주관적인 인상, 감정, 태도, 개인의 의견들을 텍스트로부터 뽑아내는 분석이다

문서 내 텍스트가 나타내는 여러가지 주관적인 단어와 문맥을 기반으로 감성수치를 계산하는 방법을 이용한다. 긍정과 부정으로 나눠진다 (polarity)

분노, 슬픔, 기쁨, 등의 감정의 상태도 분석할 수 있다(beyond polarity)

지도학습	학습/타깃 데이터를 기반으로 감성 분석 학습을 수행한 뒤, 다른 데이터의 감성분석을 예측하는 방법 (일반적인 텍스트 기반의 분류)
비지도학습	감성 어휘 사전(Lexicon)을 이용

5.1. 지도학습 감정분석

사이트 IMDB의 영화평

GettingStarted Prediction Competition

Bag of Words Meets Bags of Popcorn

Use Google's Word2Vec for movie reviews

Kaggle · 577 teams · 7 years ago

```
import pandas as pd  
  
review_df=pd.read_csv('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..."

```
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 insight into this guy who i thought was really cool in the eighties just to maybe make up my mind whether he is guilty or innocent. Moonwalker is part biography, part feature film which i remember going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's feeling towards the press and also the obvious message of drugs are bad m'kay.
>
Visually impressive but of course this is all about Michael Jackson so unless you remotely like MJ in anyway then you are going to hate this and find it boring. Some may call MJ an egotist for consenting to the making of this movie BUT MJ and most of his fans would say that he made it for the fans which if true is really nice of him.
>
The actual feature film bit when it finally starts is only on for 20 minutes or so excluding the Smooth Criminal sequence and Joe Pesci is convincing as a psychopathic all powerful drug lord. Why he wants MJ dead so bad is beyond me. Because MJ overheard his plans? Nah, Joe Pesci's character ranted that he wanted people to know it is he who is supplying drugs etc so i dunno, maybe he just hates MJ's music.
>
Lots of cool things in this like MJ turning into a car and a robot and the whole Speed Demon sequence. Also, the director must have had the patience of a saint when it came to filming the kiddy Bad sequence as usually directors hate working with one kid let alone a whole bunch of them performing a complex dance scene.
>
Bottom line, this movie is for people who like MJ on one level or another (which i think is most people). If not, then stay away. It does try and give off a wholesome message and ironically MJ's bestest buddy in this movie is a girl! Michael Jackson is truly one of the most talented people ever to grace this planet but is he guilty? Well, with all the attention i've gave this subject....hmmm well i don't know because people can be different behind closed doors, i know this for a fact. He is either an extremely nice but stupid guy or one of the most sickest liars. I hope he is not the latter."

Id
Sentiment(target)
긍정평가1 부정평가0
Review 텍스트

5.1. 지도학습 감정분석

진행 순서

텍스트 정규화 ▶ 피처 벡터화 (Count, TF-IDF) 로지스틱회귀 ▶ Pipeline

정규 표현식 모듈 `re`

영어가 아닌 모든 문자 == `[^a-zA-Z]` 필요없으므로 제거

```
import re
review_df['review']=review_df['review'].str.replace('<br />',' ')
review_df['review']=review_df['review'].apply(lambda x: re.sub('[^a-zA-Z]', " ", x))

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))
```

remember going to see at the cinema when it was originally released. Some
ess and also the obvious message of drugs are bad m'kay.

Visual
so unless you remotely like MJ in anyway then you are going to hate this
ng to making of this movie BUT MJ and most of his fans would say that

The actual feature film bit when it finally starts is only on
and Joe Pesci is convincing as a psychopathic all powerful drug lord. Why
is plans? Nah, Joe Pesci's character ranted that he wanted people to know
t hates MJ's music.

Lots of cool things in this like MJ turning

5.1. 지도학습 감정분석

CountVectorizer

```
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

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

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

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

예측 정확도는 0.8860, ROC-AUC는 0.9503

TF-IDF

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

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

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

예측 정확도는 0.8936, ROC-AUC는 0.9598

5.2. 비지도학습 감정분석

감성 지수(polarity score) 감성의 정도를 나타내는 수치
감성 수치는 단어의 위치, 주변 단어, 문맥 pos(품사) 등을 참고해 결정된다

NLTK(Natural Language Toolkit) 패키지는 자연어 처리 및 문서 분석용으로 개발됐다
감성사전인 Lexicon 모듈을 가진다

$$StSc = \frac{\text{number of positive words} - \text{number of negative words}}{\text{total number of words}}$$

Sentiment Lexicons for 81 Languages

Sentiment Polarity Lexicons (Positive vs. Negative)

다양한 언어의 감성사전 존재

Korean Sentiment Analysis Corpus

About

The KOrean Sentiment Analysis Corpus, KOSAC, is built for capturing sentiment expressions and their patterns in Korean and representing their meaning to be interpretable for a computer. To represent the meaning, a fine-grained annotation scheme called KSML (Shin et al., 2012) is developed identifying key components and properties of sentiments based on solid theoretical background. The annotation scheme has been employed in the manual annotation of a 7,713-sentence corpus of 332 news articles from the Sejong syntactic parsed corpus.

모듈 설명

형태소 분석기(morphemeServer.py)

처음 실행 후 사전을 불러오는 시간이 많이 필요하므로 형태소 분석기 서버를 만들어 요청이 왔을 때 최초 1번 사전을 불러온 후 다음 요청 부터는 로드된 사전 데이터를 이용한다.

감성 분석기(sentiment.js)

KOSAC에서 제공하는 plarity.csv 데이터를 이용해서 감성분석을 한다. 해당 데이터를 로드한 후 한글 텍스트를 형태소 분석기로 파싱한 후 감성 사전에 해당하는 감성 데이터를 가져온다.

트위터 감성 분석기(analyzer.js)

SimpleTwitterScraper 패키지를 이용해 트윗 데이터를 다운로드하고 해당 파일을 data/YYYY-MM-DD.json로 저장한 후 감성분석을 시행한 결과를 results/YYYY-MM-DD.json에 저장한다.

<https://github.com/mrlee23/KoreanSentimentAnalyzer>

5.2. 비지도학습 감정분석

NLP 패키지의 **WordNet** 모듈=semantic 분석을 제공하는 어휘 사전이다
문맥상 의미 분석을 위해 각각의 품사로 구성된 개별 단어를 **Synset**(Sets of cognitive synonyms, 단어묶음)로 표현한다

NLTK 예측 성능이 좋지 않기 때문에 NLTK를 포함한 다른 감성 사전을 이용해야 한다

- ☞ **SentiWordNet** Synset 개념을 감성분석에 적용한 것. 긍정.부정.객관성 지수.
VADER 소셜 미디어의 텍스트에 주로 사용. 뛰어난 성능과 빠른 수행시간
Pattern 뛰어난 예측성능

5.2. 비지도학습 감정분석

```
import nltk
nltk.download('all')
from nltk.corpus import wordnet as wn

term='present'

synsets=wn.synsets(term)
print('synsets() 변환 type:', type(synsets))
print('synsets() 변환 값 개수:', len(synsets))
print('synsets() 변환 값:', synsets)

synsets() 변환 type: <class 'list'>
synsets() 변환 값 개수: 18
synsets() 변환 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'), Synset('present.v.04'), Synset('present.v.05'), Synset('award.v.01'), Synset('give.v.08'), Synset('deliver.v.01'), Synset('introduce.v.01'), Synset('portray.v.04'), Synset('confront.v.03'), Synset('present.v.12'), Synset('salute.v.06'), Synset('present.a.01'), Synset('present.a.02')]
```

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

```
##### 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']
```

(present의 미. n명사 품사. 의미 구분 인덱스)

lexname 품사/ **definition** 단어의 시맨틱적 의미/
lemma_names 단어의 부명제, 다른 대체 단어

이외에도...
Synset.pos() 품사. Synset.example() 예제.
Synset.lemmas() 동의어. Synset.hypernyms() 상위어. Synset.hyponyms() 하위어

5.2. 비지도학습 감정분석

단어간의 유사도 path_similarity() == 의미가 얼마나 유사한지, 0~1사이의 값으로 반환

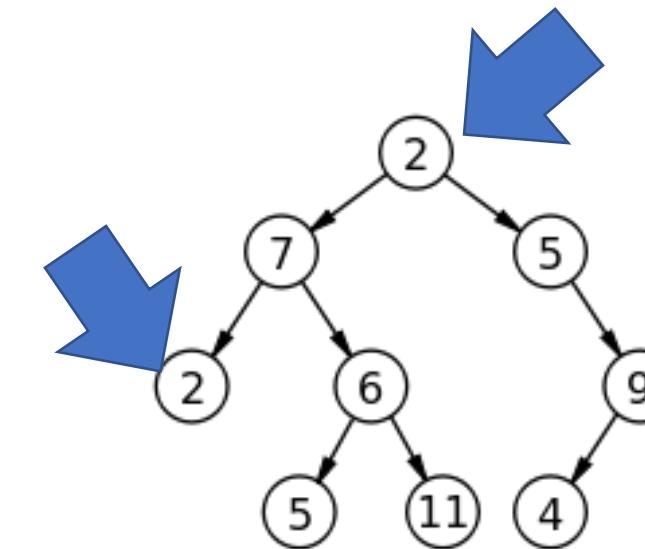
```
tree=wn.synset('tree.n.01')
lion=wn.synset('lion.n.01')
tiger=wn.synset('tiger.n.02')
cat=wn.synset('cat.n.01')
dog=wn.synset('dog.n.01')

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

for entity in entities:
    similarity=[round(entity.path_similarity(compared_entity),2) for compared_entity in entities]
    similarities.append(similarity)

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



5.2. 비지도학습 감정분석

SentiWordNet은 Senti_Synset 을 가진다

감성지수(긍정, 부정)와 비감성지수(객관성)을 가지고 있다

어떤 단어가 전혀 감성적이지 않으면 감성지수 0 비감성지수1 이 된다

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

senti_synsets=list(swn.senti_synsets('slow'))
print('senti_synsets() 변환 type:', type(senti_synsets))
print('senti_synsets() 변환 값 개수:', len(senti_synsets))
print('senti_synsets() 변환 값:', senti_synsets)

senti_synsets() 변환 type: <class 'list'>
senti_synsets() 변환 값 개수: 11
senti_synsets() 변환 값: [SentiSynset('decelerate.v.01'), SentiSynset('slow.v.02'), SentiSynset('slow.v.03'), SentiSynset('slow.a.01'), SentiSynset('slow.a.02'), SentiSynset('dense.s.04'), SentiSynset('slow.a.04'), SentiSynset('slow.r.01'), SentiSynset('slowly.r.01'), SentiSynset('behind.r.03')]
```

```
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())
print('fabulous 객관성 지수:', fabulous.obj_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
fabulous 객관성 지수: 0.0
```

5.2. 비지도학습 감정분석

진행 순서

1. 문서를 문장단위로 분해
2. 다시 문장을 단어 단위로 토큰화하고 품사 태깅
3. 품사 태깅된 단어 기반으로 synset 객체와 senti_synset 객체를 생성
4. Senti_synset에서 긍정/부정 감성 지수를 구하고 이를 모두 합산해 특정 임계치 값(0) 이상일 때 긍정감성으로 그렇지 않을 는 부정감성으로 결정

WordNet을 이용해 문서를 다시 단어로 토큰화한 뒤
어근 추출(lemmatization)과 품사 태깅(POS tagging)을 적용해야 한다

5.2. 비지도학습 감정분석

WordNet 기반의 품사 tag로 변환

문장 → 단어 토큰 → 품사 태깅 → SentiSynset 생성 → Polarity Score 합산

```
from nltk.corpus import wordnet as wn

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
```

```
from nltk import sent_tokenize, word_tokenize, pos_tag

def swn_polarity(text):
    sentiment=0.0
    tokens_count=0
    lemmatizer=WordNetLemmatizer()
    raw_sentences=sent_tokenize(text)
    for raw_sentence in raw_sentences:
        tagged_sentence=pos_tag(word_tokenize(raw_sentence))
        for word, tag in tagged_sentence:
            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
            synsets=wn.synsets(lemma, pos=wn_tag)
            if not synsets:
                continue
            synset=synsets[0]
            swn_synset=swn.senti_synset(synset.name())
            sentiment+=(swn.senti_pos_socre()-swn_synset.neg_score())

            tokens_count+=1

    if not tokens_count:
        return 0

    if sentiment>=0:
        return 1

    return 0
```

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

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

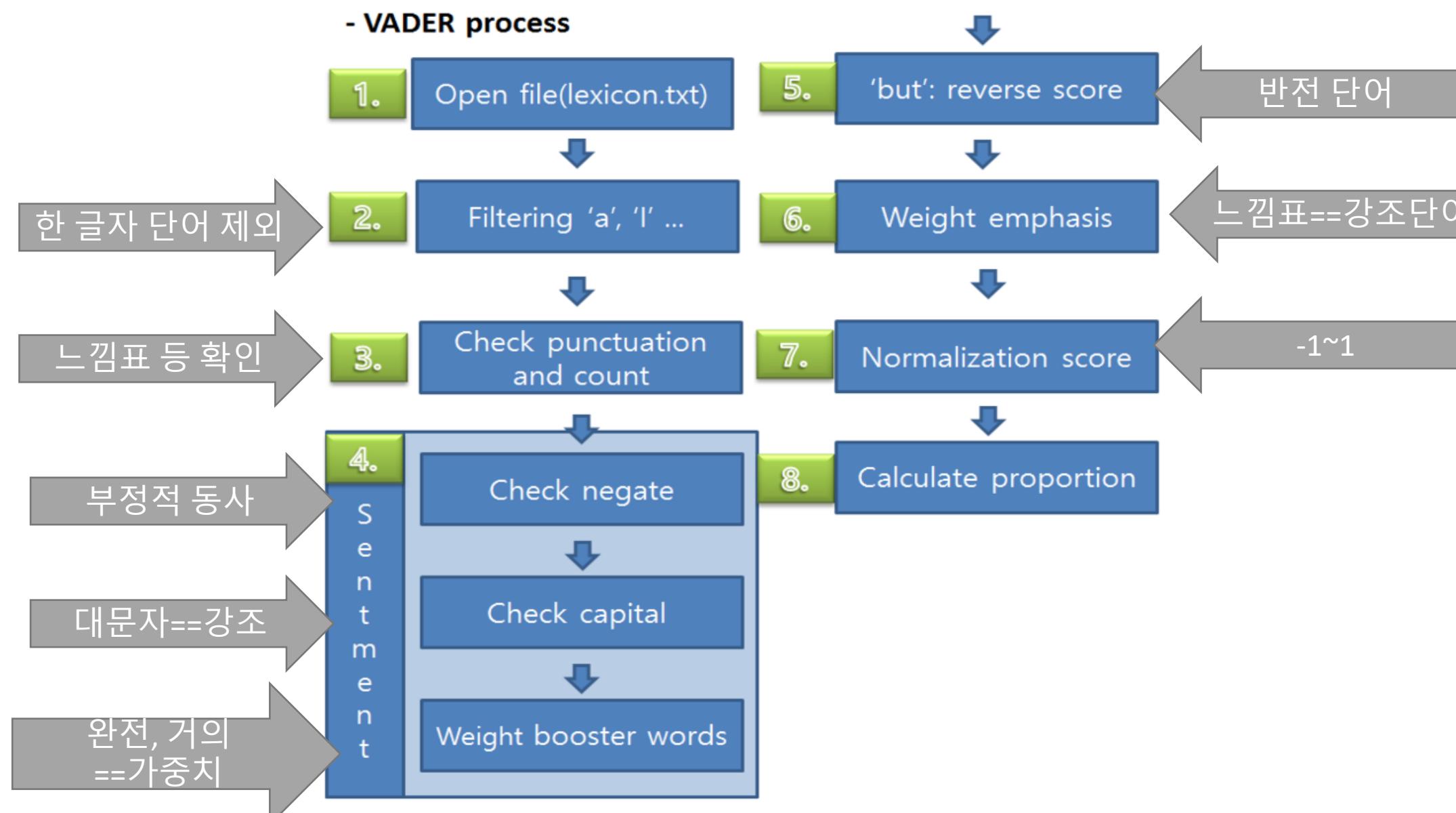
print(confusion_matrix(y_target,preds))
print("정확도: ",np.round(accuracy_score(y_target,preds),4))
print("정밀도: ",np.round(precision_score(y_target,preds),4))
print("재현율: ",np.round(recall_score(y_target,preds),4))
```

```
[[7668 4832]
 [3636 8864]]
정확도: 0.6613
정밀도: 0.6472
재현율: 0.7091
```

5.2. 비지도학습 감정분석

VADER Valence Aware Dictionary and sEntiment Reasoner lexicon

소설 미디어의 감성 분석 목적. SentimentIntensityAnalyzer 클래스를 이용해 감성 분석을 제공한다



```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sent_i_analyzer = SentimentIntensityAnalyzer()
sent_i_scores = sent_i_analyzer.polarity_scores(review_df['review'][0])
print(sent_i_scores)

def vader_polarity(review, threshold=0.1):
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review)

    agg_score = scores['compound']
    final_sentiment = 1 if agg_score >= threshold else 0
    return final_sentiment

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, preds), 4))
print("재현율:", np.round(recall_score(y_target, preds), 4))

{'neg': 0.13, 'neu': 0.743, 'pos': 0.127, 'compound': -0.7943}
[[ 6747  5753]
 [ 1858 10642]]
정확도: 0.6956
정밀도: 0.6472
재현율: 0.7091
```

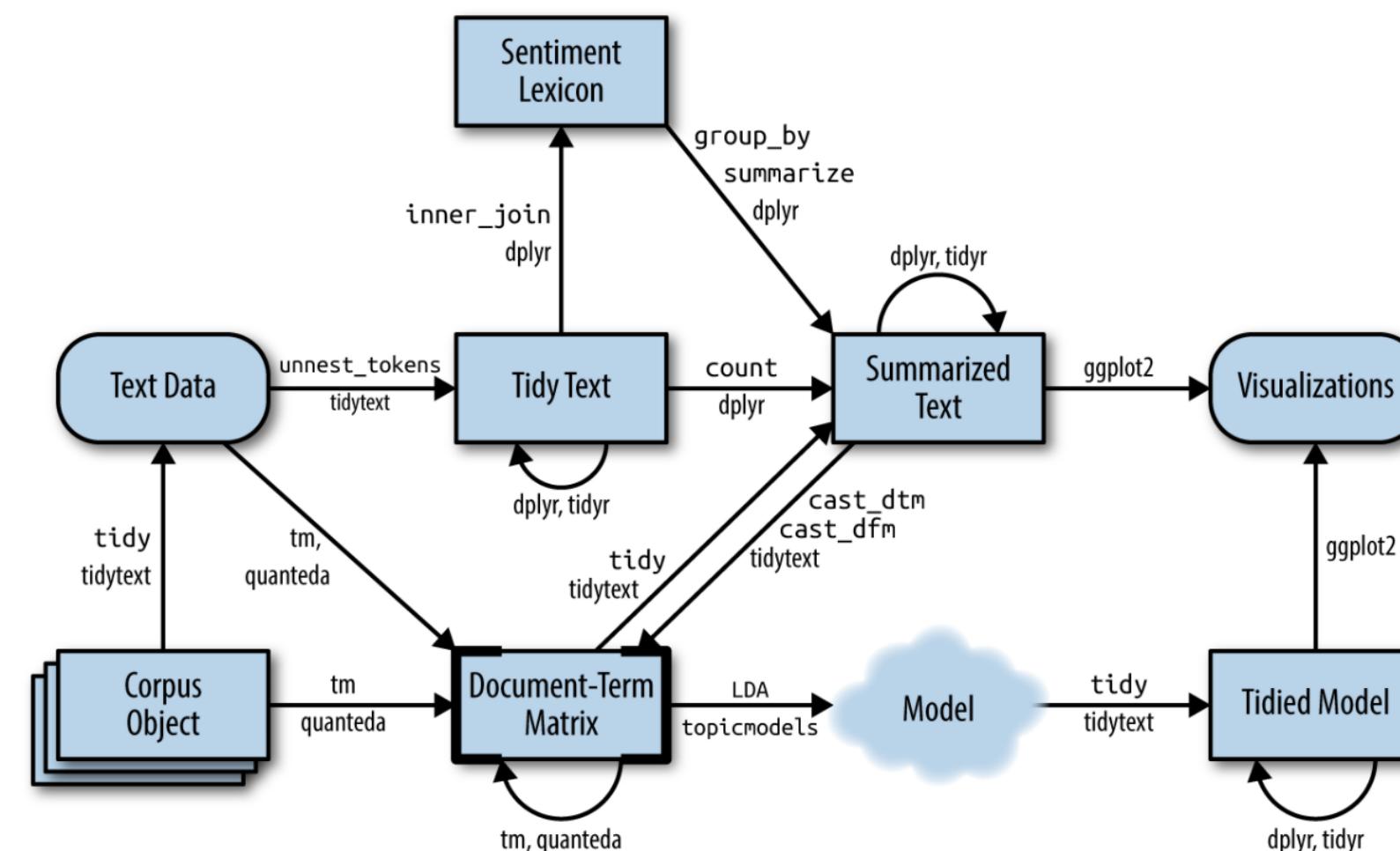
06. 토퍽모델링



6. 토픽 모델링(Topic Modeling)

토픽 모델링이란 문서 집합에 숨어 있는 (중요, 핵심) 주제를 찾아내는 것이다. 비지도학습 분류.

사람의 경우, 함축적인 의미로 문장을 요약하는 반면, 머신러닝의 경우, 함축적인 중심단어로 추출한다



LSA(Latent Semantic Analysis), LDA(Latent Dirichlet Allocation)

6.1. LDA(latent dirichlet allocation)

2가지 가정

- 모든 문서는 토픽이 혼합되어 있다 (문서 1은 토픽A 90%, 토픽B 10% 로 구성됨)
- 모든 토픽은 단어가 혼합되어 있다 (정치 토픽에서 사용되는 단어는 국회, 정부이고, 엔터테인먼트 토픽에서 사용되는 단어는 영화, TV 이다. 두 토픽에서 동등하게 사용되는 단어는 예산이다)

목표

각 토픽과 관련된 단어의 조합을 찾는 동시에 각 문서를 설명하는 토픽의 조합을 결정한다

LDA는 Count벡터화만 적용가능하다

Unsupervised Generative Topic Model

6.1. LDA(latent dirichlet allocation)

[Components_](#)는 개별 토픽별로 각 word 피처가 얼마나 많이 할당됐는 지에 관한 수치를 가진다
높을 수록 해당 토픽의 중심 word가 된다

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

cats=['rec.motorcycles', 'rec.sport.baseball', 'comp.graphics', 'comp.windows.x', 'talk.pol.

news_df=fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'), categorie
count_vect=CountVectorizer(max_df=0.95, max_features=1000, min_df=2, stop_words='english',
feat_vect=count_vect.fit_transform(news_df.data)
print('CountVectorizer Shape:', feat_vect.shape)
```

CountVectorizer Shape: (7862, 1000)

```
lda=LatentDirichletAllocation(n_components=8, random_state=0)
lda.fit(feat_vect)

LatentDirichletAllocation(n_components=8, random_state=0)

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]])
```

6.1. LDA(latent dirichlet allocation)

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

        topic_word_indexes=topic.argsort()[:-1]
        top_indexes=topic_word_indexes[:no_top_words]

        feature_concat=' '.join([feature_names[i] for i in top_indexes])
        print(feature_concat)

feature_names=count_vect.get_feature_names()

display_topics(lda, feature_names, 15)
```

야구, 그래픽스, 윈도우, 중동, 기독교, 전자공학, 의학

Topic # 0
year 10 game medical health team 12 20 disease cancer 1993 games years patients good
Topic # 1
don just like know people said think time ve didn right going say 11 way
Topic # 2
image file jpeg program gif images output format files color entry 00 use bit 03
Topic # 3
like know don think use does just good time book read information people used post
Topic # 4
armenian israel armenians jews turkish people israeli jewish government war dos dos turkey arab armenia 000
Topic # 5
edu com available graphics ftp data pub motif mail widget software mit information version sun
Topic # 6
god people jesus church believe christ does christian say think christians bible faith sin life
Topic # 7
use dos thanks windows using window does display help like problem server need know run

6.2. LSA(Latent Semantic Analysis)

BOW에 기반한 TF-IDF등은 단어의 빈도수를 이용하지만 단어의 의미는 고려하지 못한다는 단점이 있다.
이에 대한 대안으로 Truncated SVD를 사용하여 차원을 축소시키고, 단어의 잠재된 의미를 끌어내는 방법이 바로 LSA이다

$$A_{mxn} \approx U_{mxr} \Sigma_{rxr} V^T_{rxn}$$

장점으로 쉽고 빠르고 좋은 성능을 지닌다는 특징이 있지만 단점으로는 SVD특성상 이미 계산된 LSA에 새로운 데이터를 추가하여 계산하려고 하면 처음부터 다시 계산해야 하기 때문에 새로운 정보에 관한 업데이트가 어렵다는 특징을 가진다

LSA의 단점을 개선하여 탄생한 것이 LDA로, 보다 더 적합한 알고리즘이다

6.2. LSA(Latent Semantic Analysis)

```
from sklearn.feature_extraction.text import TfidfVectorizer  
  
tfidf_vect = TfidfVectorizer(max_df=0.95, max_features=1000, min_df=2, stop_words='english', ngram_range=(1,2))  
feat_vect=tfidf_vect.fit_transform(news_df.data)  
print('TF-IDF 행렬의 크기 :',feat_vect.shape)
```

TF-IDF 행렬의 크기 : (7862, 1000)

```
from sklearn.decomposition import TruncatedSVD
```

```
svd = TruncatedSVD(n_components=8, random_state=0)  
svd.fit(feat_vect)
```

TruncatedSVD(n_components=8, random_state=0)

```
display_topics(svd, feature_names, 15)
```

Topic # 0
don like know just think people does god good time use ve way thanks say
Topic # 1
god people jesus think israel believe christ say church jews said did christians christian sin
Topic # 2
god jesus does thanks christ church sin faith christians christian bible does know know lord advance
Topic # 3
israel armenian jews armenians israeli jewish turkish arab people government war arabs edu muslim armenia
Topic # 4
know does thanks does know don advance thanks advance bike israel anybody don know hi jews like israeli
Topic # 5
bike just ve ride use window like dod riding israel want right used car work
Topic # 6
bike armenian edu armenians thanks turkish god armenia genocide mail new turkey turks advance ride
Topic # 7
edu israel soon bike israeli com cs article university dod arab email don jews jewish

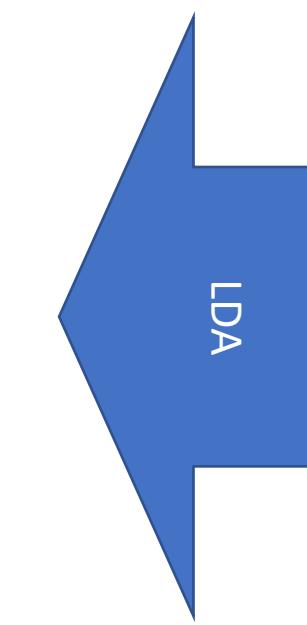
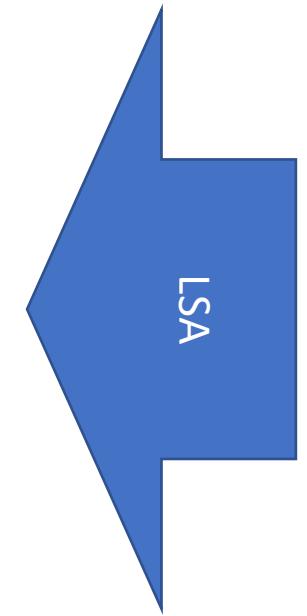
야구, 그래픽스, 윈도우, 중동, 기독교, 전자공학, 의학

6.2. LSA(Latent Semantic Analysis)

Topic # 0
don like know just think people does god good time use ve way thanks say
Topic # 1
god people jesus think israel believe christ say church jews said did christians christian sin
Topic # 2
god jesus does thanks christ church sin faith christians christian bible does know know lord advance
Topic # 3
israel armenian jews armenians israeli jewish turkish arab people government war arabs edu muslim armenia
Topic # 4
know does thanks does know don advance thanks advance bike israel anybody don know hi jews like israeli
Topic # 5
bike just ve ride use window like dod riding israel want right used car work
Topic # 6
bike armenian edu armenians thanks turkish god armenia genocide mail new turkey turks advance ride
Topic # 7
edu israel soon bike israeli com cs article university dod arab email don jews jewish

야구, 그래픽스, 윈도우, 중동, 기독교, 전자공학, 의학

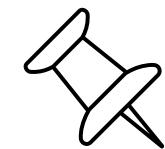
Topic # 0
year 10 game medical health team 12 20 disease cancer 1993 games years patients good
Topic # 1
don just like know people said think time ve didn right going say ll way
Topic # 2
image file jpeg program gif images output format files color entry 00 use bit 03
Topic # 3
like know don think use does just good time book read information people used post
Topic # 4
armenian israel armenians jews turkish people israeli jewish government war dos dos turkey arab armenia 000
Topic # 5
edu com available graphics ftp data pub motif mail widget software mit information version sun
Topic # 6
god people jesus church believe christ does christian say think christians bible faith sin life
Topic # 7
use dos thanks windows using window does display help like problem server need know run



07. 감성분석



7.1 Womens Clothing E-Commerce Reviews



노트북 설명: 여성 의류 쇼핑몰 리뷰 데이터를 분석

→ positive review, negative review로 데이터를 정리

→ 다양한 ML 모델에 적합

(변수 종류)

Clothing ID

Age

Title

} 감성분석에 이용

Review Text

Rating (1~5로 나타냄)

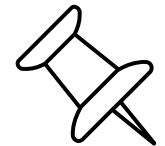
Recommended IND (리뷰어의 제품 추천 여부를 0, 1로 표현)

Positive Feedback Count

Division Name

Department Name

7.1 Womens Clothing E-Commerce Reviews



감성분석: 자연어 처리, 텍스트 분석, 컴퓨터 언어학 및 생체 인식을 사용하여 정서적 상태와 주관적 정보를 체계적으로 식별, 추출, 수량화 및 연구하는 것

```
from wordcloud import WordCloud
```

워드 클라우드란 문서의 키워드, 개념 등을 직관적으로 파악할 수 있도록 핵심 단어를 시각화하는 기법이다. 예를 들면 많이 언급될수록 단어를 크게 표현해 한눈에 들어올 수 있게 하는 기법 등이 있다. 주로 빅데이터(big data)를 분석할 때 데이터의 특징을 도출하기 위해 활용.

```
from textblob import TextBlob
```

TextBlob을 통해 "Simplified Text Processing"을 실습해보자.

Noun phrase extraction

Part-of-speech tagging

Sentiment analysis

Classification (Naive Bayes, Decision Tree)

Language translation and detection powered by Google Translate

Tokenization (splitting text into words and sentences)

Word and phrase frequencies

Parsing

n-grams

Word inflection (pluralization and singularization) and lemmatization

Spelling correction

Add new models or languages through extensions

Add new models of
WordNet integration



7.1 감성분석 – preprocessing (punctuation삭제)

```
text_prep = text_df.copy()
```

Preprocessing - text features

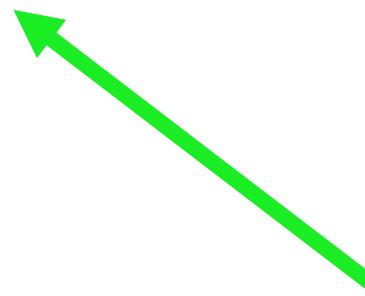
```
string.punctuation
```

```
'!"#$%&₩'()*+,-./:;<=>?@[₩₩₩]^_`{|}~'
```

```
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
```

```
text_prep['Review'] = text_prep['Review'].apply(punctuation_removal) Review에 해당하는 값에서 punctuation을 모두 삭제  
text_prep['Review'].head()
```

```
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 Cagrocoal shimmer fun I aded this in my basket ...  
Name: Review, dtype: object
```



```
Some major design flaws I had such high hopes ...
```

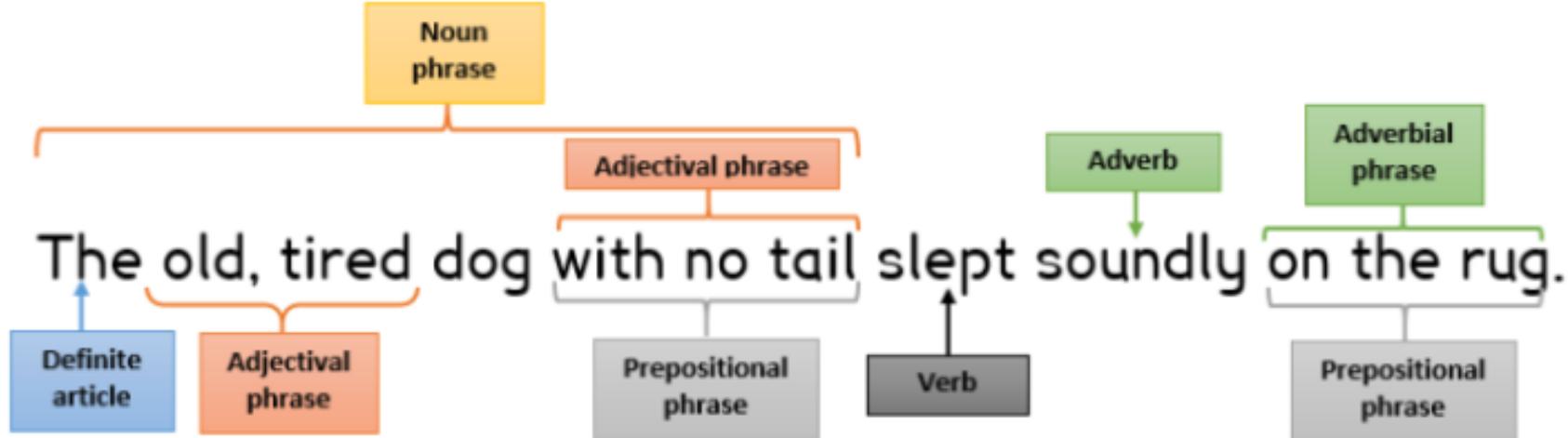
```
My favorite buy! I love, love, love this jumps...
```

```
Flattering shirt This shirt is very flattering...
```

```
Not for the very petite I love tracy reese dre...
```

```
Cagrocoal shimmer fun I aded this in my basket ...
```

7.1 감성분석 – preprocessing (verb, adjective 추출)



동사, 형용사가 문장의 감성을 잘 드러냄

```
def adj_collector(review_string):
    new_string = []
    review_string = word_tokenize(review_string)
    tup_word = nltk.pos_tag(review_string)
    for tup in tup_word:
        if 'VB' in tup[1] or tup[1] == 'JJ': #Verbs and Adjectives
            new_string.append(tup[0])
    return ' '.join(new_string)
```

동사, 형용사를 추출하는 함수를 설정

```
text_prep['Review'] = text_prep['Review'].apply(adj_collector)
text_prep['Review'].head(7)
```

```
2 major had such high wanted work i ordered smal...
3 favorite love love fabulous wear i get great
4 Flattering is flattering due adjustable is per...
5 petite love reese is petite am tall wear was i...
6 aded last see look i went am pale is gorgeous ...
7 goes ordered had try used top pair went is nic...
8 Flattering love get runs little ordered flatte...
Name: Review, dtype: object
```

7.1 감성분석 – preprocessing (stopwords 제거)

- 불용어(Stopword)

분석을 하는 것에 있어서는 큰 도움이 되지 않는…

예를 들면, I, my, me, over, 조사, 접미사 같은 제거해야함.

NLTK에서는 불용어로 패키지 내에서 미리 정의하고 있지만 개인이 직접 정의도 가능

```
stop = stopwords.words('english')      'English'에 이미 저장된 stopwords
stop.append("i'm")                    Stopwords에 "I'm"을 추가

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']

clothes_list =['dress', 'top','sweater','shirt', 'skirt','material', 'white', 'black',
'jeans', 'fabric', 'color','order', 'wear']      "clothes_list"라는 불용어를 추가로 지정한다. (감성분석에 필요 없음)

def stopwords_removal(messy_str):                  "stop_words", "clothes_list"에 있는 단어 모두 제외
    messy_str = word_tokenize(messy_str)
    return [word.lower() for word in messy_str
            if word.lower() not in stop_words and word.lower() not in clothes_list]
```

7.1 감성분석 – preprocessing (숫자 제거)

Removing all numbers (weight, size etc.)

```
def drop_numbers(list_text):
    list_text_new = []
    for i in list_text:
        if not re.search('\d', i):
            list_text_new.append(i)
    return ''.join(list_text_new)
```

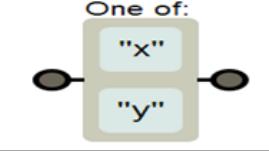
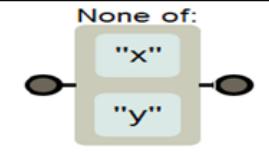
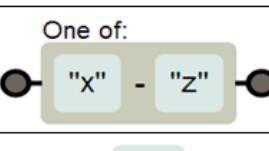
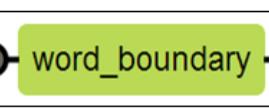
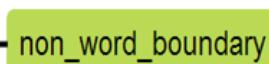
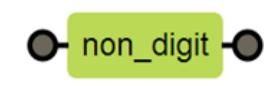
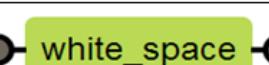
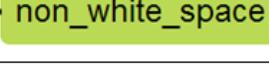
```
text_prep['Review'] = text_prep['Review'].apply(drop_numbers)
text_prep['Review'].head()
```

```
2 major high wanted work ordered small usual fou...
3 favorite love love fabulous get great
4 flattering flattering due adjustable perfect p...
5 petite love reese petite tall long full overwh...
6 aded last see look went pale gorgeous turns ma...
Name: Review, dtype: object
```

숫자 제외, 문자만 추출함

결과

- ① punctuation 삭제
- ② 동사 / 형용사 만 추출
- ③ stopwords 삭제
- ④ 숫자 삭제

정규표현식	표현	설명
[xy]		x,y중 하나를 찾습니다.
[^xy]		x,y를 제외하고 문자 하나를 찾습니다. (문자 클래스 내의 ^는 not을 의미합니다.)
[x-z]		x~z 사이의 문자중 하나를 찾습니다.
\^		^(특수문자)를 식에 문자 자체로 포함합니다. (escape)
\w\B		문자와 공백사이의 문자를 찾습니다.
\w\W\B		문자와 공백사이가 아닌 값을 찾습니다.
\w\d		숫자를 찾습니다.
\w\W\d		숫자가 아닌 값을 찾습니다.
\w\s		공백문자를 찾습니다.
\w\W\s		공백이 아닌 문자를 찾습니다.
\w\t		Tab 문자를 찾습니다.
\w\v		Vertical Tab 문자를 찾습니다.
\w\w		알파벳 + 숫자 + _ 를 찾습니다.
\w\W\w		알파벳 + 숫자 + _ 을 제외한 모든 문자를 찾습니다.

7.2 감성분석 – stemming(어간추출)

stemming

```
porter = PorterStemmer()
```

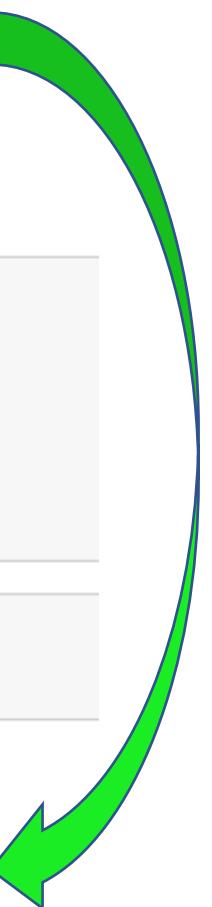
```
text_prep['Review'] = text_prep['Review'].apply(lambda x: x.split())
text_prep['Review'].head()
```

```
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
```

```
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
```

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

```
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
```



7.3 감성분석 – WordCloud

WordCloud - Repetition of words

```
pos_df = text_prep[text_prep.Recommended== 1] 추천했을 경우를 pos_df로,  
neg_df = text_prep[text_prep.Recommended== 0] 추천하지 않을 경우를 neg_df로 지정  
pos_df.head(3)
```

Recommended	Review	Review_length	count_exc	Polarity
3	1 favorit love love fabul get great	141	3	0.560714
4	1 flatter flatter due adjust perfect pair cardigan	209	3	0.512891
6	1 ade last see look went pale gorgeou turn mathc...	517	0	0.157500

```
pos_words =[]  
neg_words = []  
  
for review in pos_df.Review:  
    pos_words.append(review)  
pos_words = ' '.join(pos_words)  
pos_words[:40]  
  
for review in neg_df.Review:  
    neg_words.append(review)  
neg_words = ' '.join(neg_words)  
neg_words[:400]
```

'major high want work order small usual found small small zip reorder petit ok overal comfort fit bottom tight sever cheap imo major net c petit love rees petit tall long full overwhelm small shorten narrow take love work return look cheap run small run order fit tight cheap pull caus rip disappoint go say go look style side purchas knew larg next imposs second look cheap awkward tight look describ'

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

7.3 감성분석 – WordCloud

Positive reviews

```
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 Reviews

```
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 감성분석 – (TF-IDF)

```
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) 0.393148931086517  
(0, 4438) 0.4065409531496632  
(0, 4302) 0.3214748504588952  
(0, 3881) 0.3072123012448639  
(0, 3507) 0.21175884431125436  
(0, 3438) 0.20250236542769373  
(0, 2416) 0.2403172376641177
```

TF(Term Frequency):: 특정 문서의 특정 단어 빈도수
DF(document frequency) : 특정 단어가 등장한 문서의 수
IDF (inverse document frequency): DF의 역수
TF-IDF: TF와 IDF의 곱으로 나타낼 수 있음

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

7.5 감성분석 – PCA

Data Visualization (PCA)

```
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]
array([-0.13554902, -0.05376844])
```

```
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")
```

```
<AxesSubplot :>
```



```
X_train_scaled = scipy.sparse.csr_matrix(X_train_scaled)
X_test_scaled = scipy.sparse.csr_matrix(X_test_scaled)
X_train = scipy.sparse.csr_matrix(X_train.values)
X_test = scipy.sparse.csr_matrix(X_test.values)
X_test
```

```
<4945x7297 sparse matrix of type '<class 'numpy.float64'>'  
with 71999 stored elements in Compressed Sparse Row format>
```

7.5 감성분석 – model(SVC)

Models

```
def report(y_true, y_pred, labels):
    cm = pd.DataFrame(confusion_matrix(y_true=y_true, y_pred=y_pred),
                       index=labels, columns=labels)
    rep = classification_report(y_true=y_true, y_pred=y_pred)
    return (f'Confusion Matrix:\n{cm}\nClassification Report:\n{rep}' )
```

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

```
SVC(class_weight='balanced', kernel='linear', probability=True,
     random_state=111)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
test_predictions = svc_model.predict(X_test_scaled)
print(report(y_test, test_predictions, svc_model.classes_ ))
```

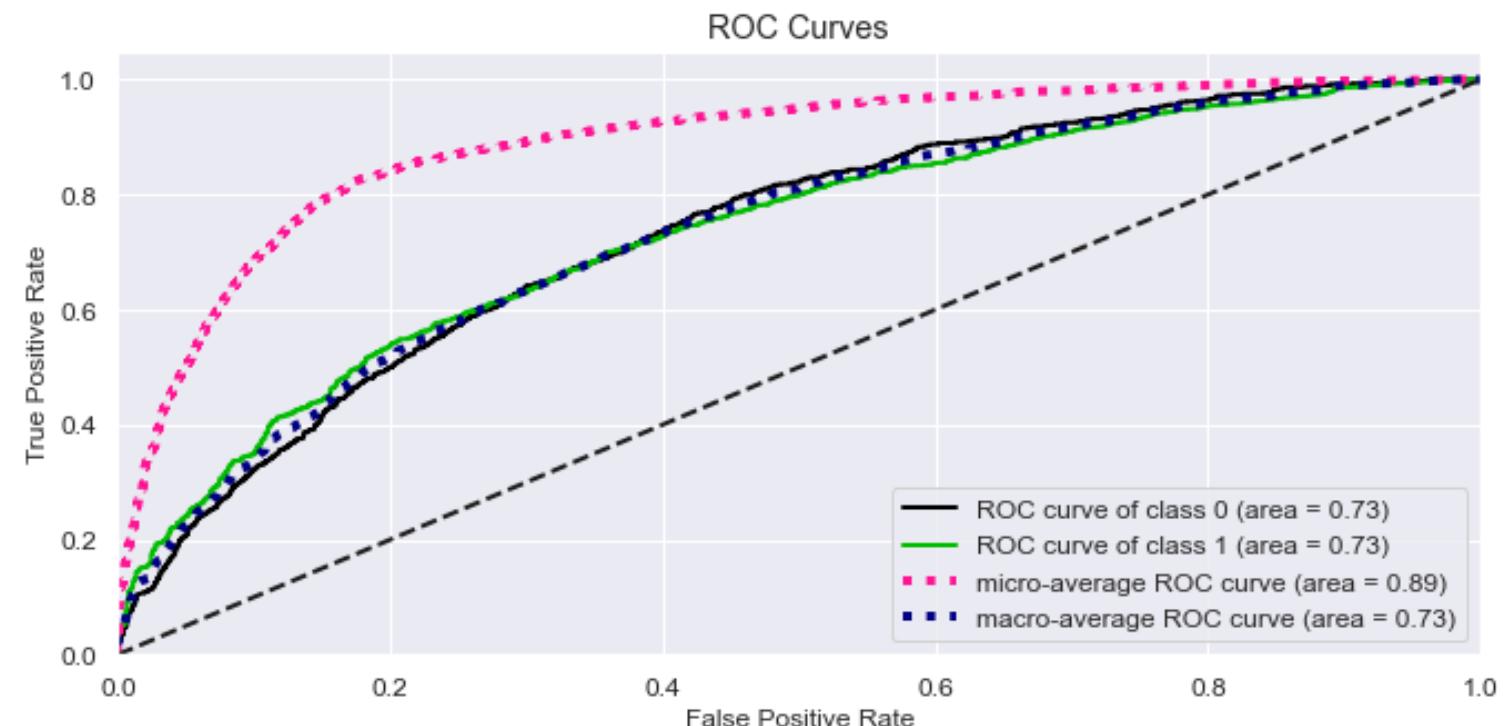
Confusion Matrix:

	0	1
0	513	384
1	1009	3039

Classification Report:

	precision	recall	f1-score	support
0	0.34	0.57	0.42	897
1	0.89	0.75	0.81	4048
accuracy			0.72	4945
macro avg	0.61	0.66	0.62	4945
weighted avg	0.79	0.72	0.74	4945

```
skplt.metrics.plot_roc(y_test, svc_model.predict_proba(X_test_scaled))
<AxesSubplot:title={'center':'ROC Curves'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>
```



7.5 감성분석 – model(LR)

Logistic Regression

```
lr_model = LogisticRegression(class_weight='balanced',
                             random_state=111,
                             solver='lbfgs',
                             C=1.0)

gs_lr_model = GridSearchCV(lr_model,
                           param_grid={'C': [0.01, 0.1, 0.5, 1.0, 5.0]},
                           cv=5,
                           scoring='roc_auc')

gs_lr_model.fit(X_train_scaled, y_train)
```

```
GridSearchCV(cv=5,
             estimator=LogisticRegression(class_weight='balanced',
                                           random_state=111),
             param_grid={'C': [0.01, 0.1, 0.5, 1.0, 5.0]}, scoring='roc_auc')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
gs_lr_model.best_params_
```

```
{'C': 0.1}
```

```
test_predictions = gs_lr_model.predict(X_test_scaled)
print(report(y_test, test_predictions, gs_lr_model.classes_ ))
```

Confusion Matrix:

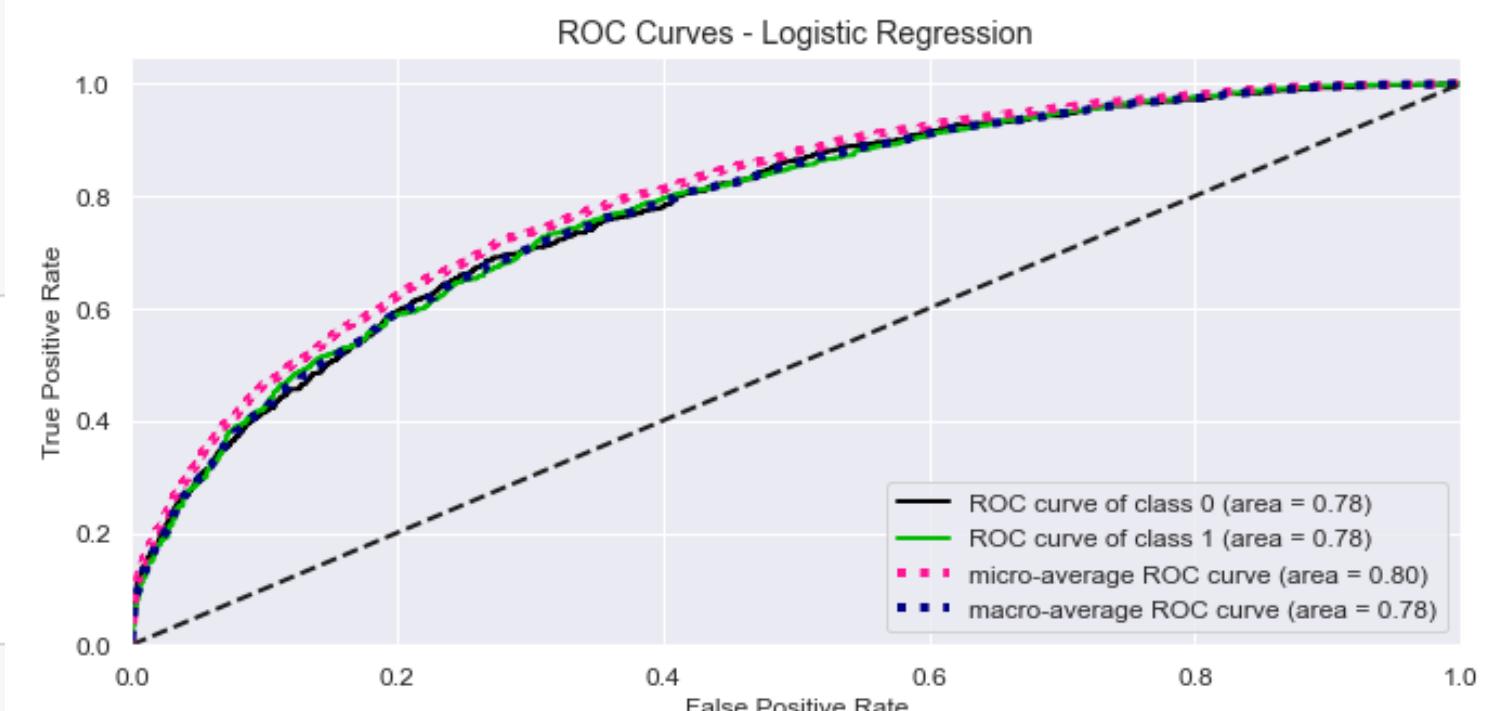
	0	1
0	619	278
1	1101	2947

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

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

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



7.5 감성분석 – model(Ada Boost)

3. AdaBoost

```
dt = DecisionTreeClassifier(max_depth=5, class_weight='balanced', random_state=555)
ada_model = AdaBoostClassifier(base_estimator=dt, learning_rate=0.001, n_estimators=1000, random_state=222)
ada_model.fit(X_train, y_train)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(class_weight='balanced',
                                                       max_depth=5,
                                                       random_state=555),
                  learning_rate=0.001, n_estimators=1000, random_state=222)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
test_predictions = ada_model.predict(X_test)
print(report(y_test, test_predictions, ada_model.classes_))
```

Confusion Matrix:

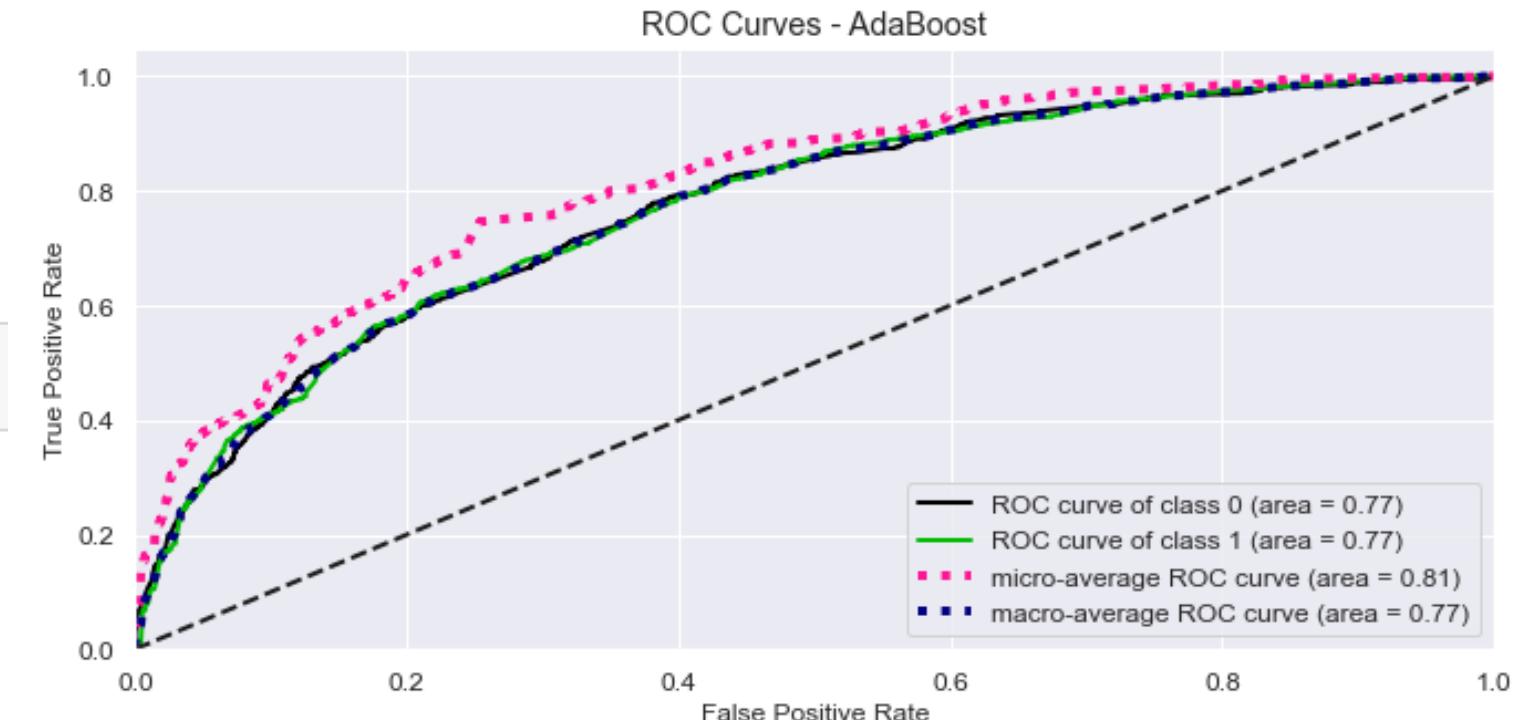
	0	1
0	547	350
1	906	3142

Classification Report:

	precision	recall	f1-score	support
0	0.38	0.61	0.47	897
1	0.90	0.78	0.83	4048
accuracy	0.64	0.69	0.75	4945
macro avg	0.64	0.69	0.75	4945
weighted avg	0.80	0.75	0.77	4945

```
skplt.metrics.plot_roc(y_test, ada_model.predict_proba(X_test),
                      title='ROC Curves - AdaBoost')

<AxesSubplot:title={'center':'ROC Curves - AdaBoost'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>
```



7.5 감성분석 – model(Ada Boost)

4. Random Forest

```
rf_model = RandomForestClassifier(n_estimators=1000, max_depth=5,
                                  class_weight='balanced', random_state=3)
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(class_weight='balanced', max_depth=5, n_estimators=1000,
                      random_state=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
test_predictions = rf_model.predict(X_test)
print(report(y_test, test_predictions, rf_model.classes_ ))
```

Confusion Matrix:

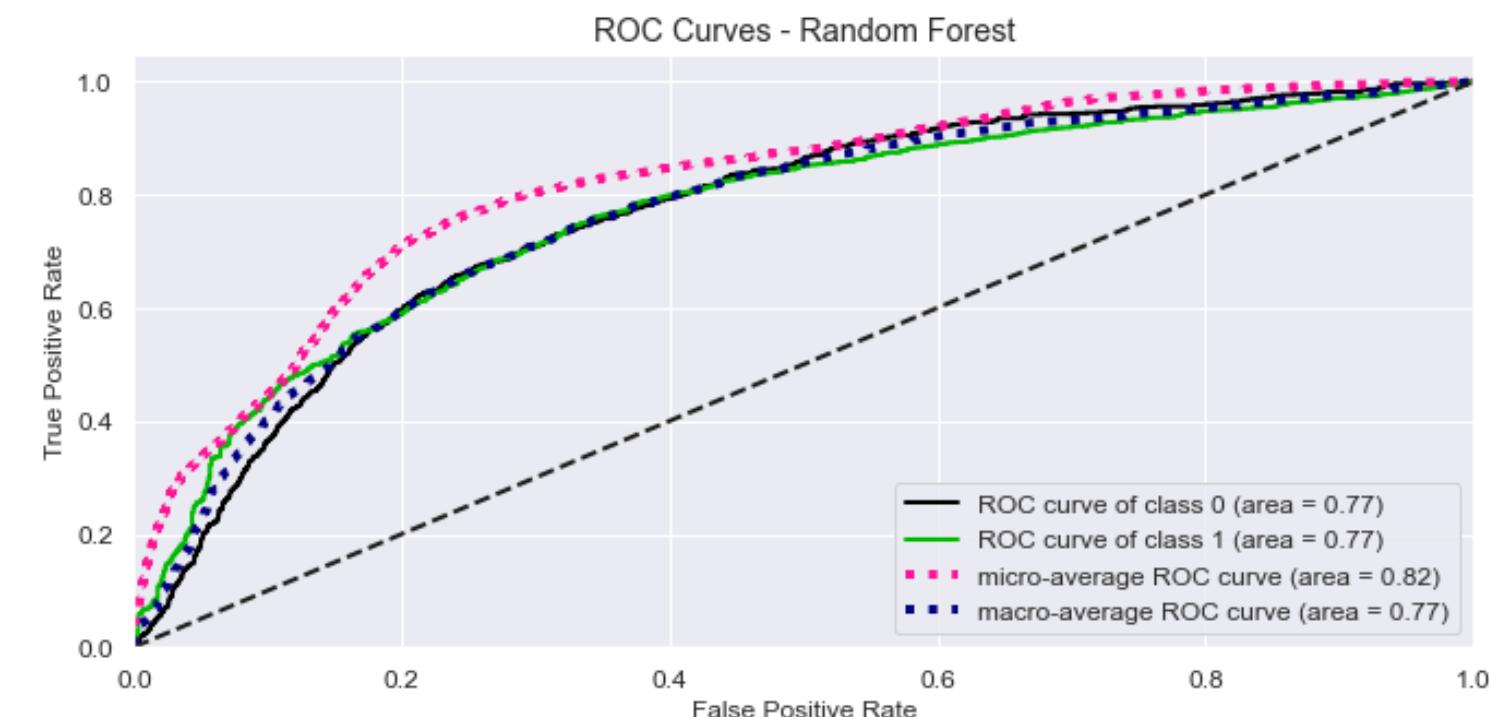
	0	1
0	549	348
1	848	3200

Classification Report:

	precision	recall	f1-score	support
0	0.39	0.61	0.48	897
1	0.90	0.79	0.84	4048
accuracy			0.76	4945
macro avg	0.65	0.70	0.66	4945
weighted avg	0.81	0.76	0.78	4945

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

```
<AxesSubplot:title={'center':'ROC Curves - Random Forest'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>
```



- SVM - f1 micro score of 0.72
- Logistic Regression - f1 micro score of 0.72
- AdaBoost - f1 micro score of 0.75
- Random Forest - f1 micro score of 0.76

08. Topic modeling



8.1 trip hotel review - GSMD

What is the GSMD?

- 자동적으로 cluster의 수를 추론
- completeness / homogeneity 의 균형을 맞춤
- sparse / high-dimensional에서도 가능
- 각 cluster의 대표어(representative word) 나타내기 가능

토픽 모델링(Topic Modeling)이란 기계 학습 및 자연어 처리 분야에서 토픽이라는 문서 집합의 추상적인 주제를 발견하기 위한 통계적 모델 중 하나로, 텍스트 본문의 숨겨진 의미 구조를 발견하기 위해 사용되는 텍스트 마이닝 기법.

parameters

- K = 6 (cluster의 수)
- alpha = 0.01 and beta = 0.01.
- n_iters = 30.

```
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)
```

top_words 설정

```
def top_words(cluster_word_distribution, top_cluster, values):
    for cluster in top_cluster:
        sort_dicts = sorted(mgp.cluster_word_distribution[cluster].items(), key=lambda k: k[1], reverse=True)[:values]
        print("\nCluster %s : %s" % (cluster, sort_dicts))
```

```
doc_count = np.array(mgp.cluster_doc_count)
print('Number of documents per topic :', doc_count)
```

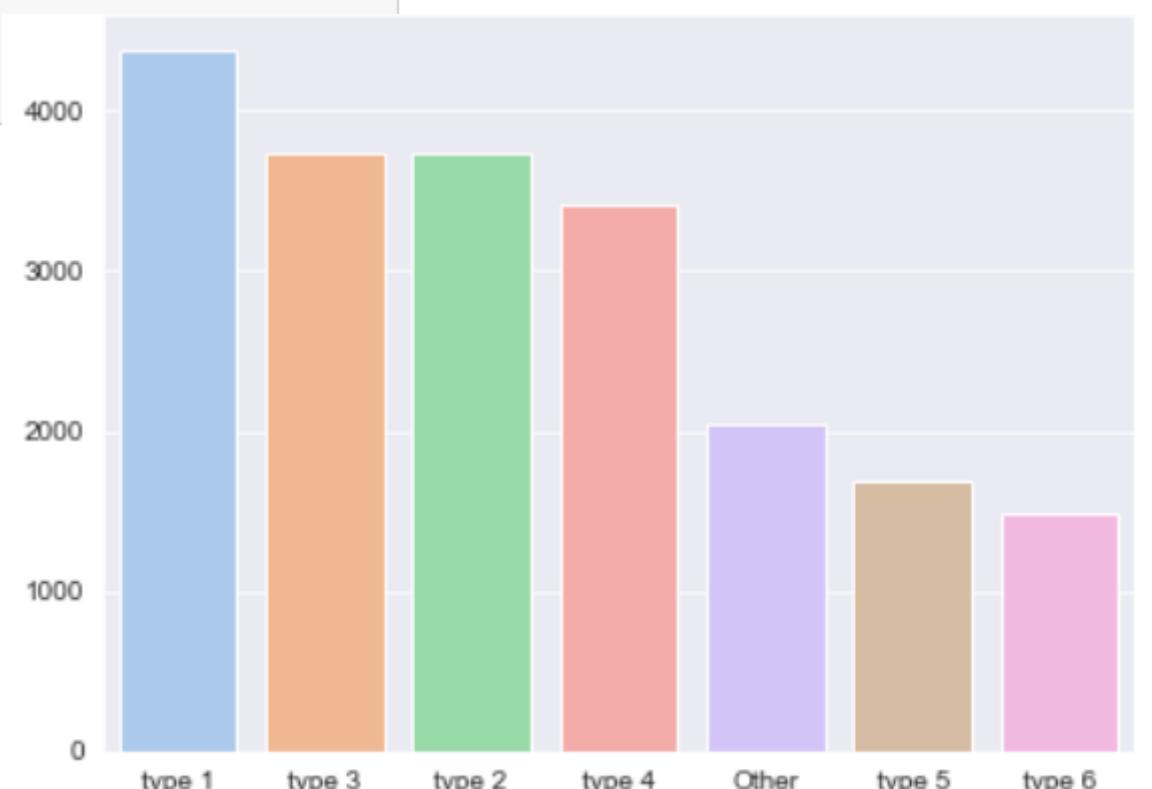
```
# topics sorted by the number of document they are allocated to
top_index = doc_count.argsort()[-10:][::-1]
print("\nMost important clusters (by number of docs inside):", top_index)
# show the top 5 words in term frequency for each cluster
top_words(mgp.cluster_word_distribution, top_index, 10) 가장 자주 나오는 5개의 단어 추출
```

8.1 trip hotel review - GSDM

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

```
def create_topics_dataframe(data_text=data.Review, mpg=mpg, threshold=0.3, topic_dict=topic_dict, lemma_text=reviews_lemmatized):  
    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 = mpg.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
```

	Text	Topic	Rating	Lemma-text
0	nice hotel expensive parking got good deal sta...	type 2	4	[parking, deal, anniversary, evening, advice, ...]
1	ok nothing special charge diamond member hilton...	Other	2	[ok, charge, decide, chain, shoot, anniversary...]
2	nice rooms not 4* experience hotel monaco seat...	type 5	3	[experience, level, positive, bathroom, suite,...]
3	unique, great stay, wonderful time hotel monac...	type 1	5	[stroll, downtown, shopping, area, pet, sign, ...]
4	great stay great stay, went seahawk game aweso...	type 1	5	[game, view, building, complain, website, pack...]



8.2 trip hotel review – LDA

Lda model

```
: id2word = corpora.Dictionary(reviews_lemmatized)    LDA모델에 들어갈 객체를 생성
texts = reviews_lemmatized
corpus = [id2word.doc2bow(text) for text in texts] id2 word: Dictionary에 list of list of str 형식의 documents를 입력하면 Dictioanry 가 학습. 전체 말뭉치에서 단어가 겹치지 않도록 하나씩 dictionary에 저장.
corpus : 트윗 리스트 안의 단어를 bag-of-words 형태-> list of (token_id, token_count) 2-tuples로 변환.
```

```
: # Use TF-IDF
tfidf = models.TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]
```

```
: from gensim.models.ldamulticore import LdaMulticore

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, alpha=.1, eta=0.1, random_state = 42)
        model_list.append(model)
        print('model created')
        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

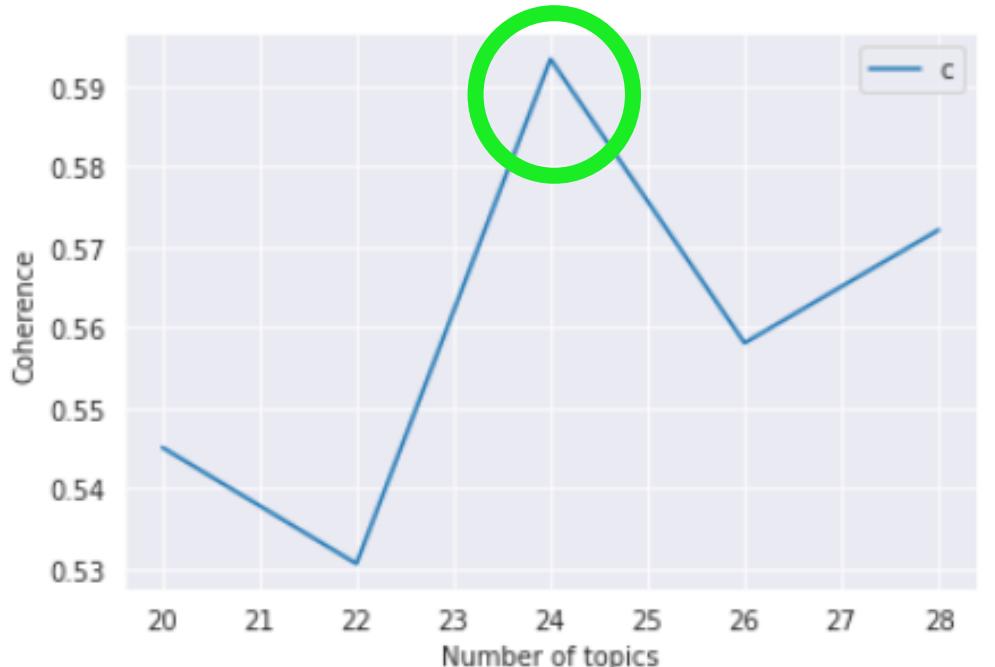
model_list, coherence_values = calc_coherence_values(dictionary = id2word, corpus = corpus_tfidf, texts = texts, start = 20, limit
```

```
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()
```

Means of LDA coherence score:

- 0.3 is bad
- 0.4 is low
- 0.55 is okay
- 0.65 might be as good as it is going to get
- 0.7 is nice
- 0.8 is unlikely
- 0.9 is probably wrong

Topic Coherence	
토픽이 얼마나 의미론적으로 일관성 있는지. 높을수록 의미론적 일관성 높음	
해당 모델이 얼마나 실제로 의미있는 결과를 내는지 확인하기 위해 사용	
평가를 진행하기 위해 다른 외부 데이터(코퍼스, 시소러스 등)가 필요	



8.3 trip hotel review – LSA (Latent Semantic Analysis)

Latent Semantic Analysis (LSA 또는 LSI라고 부름)

DTM이나 TF-IDF는 기본적으로 단어의 빈도 수를 이용한 수치화 방법이기 때문에 단어의 의미를 고려하지 못한다는 단점이 존재. (토픽을 고려하지 못함)

이를 위한 대안으로 DTM의 잠재된(Latent) 의미를 이끌어내는 방법으로 잠재 의미 분석(Latent Semantic Analysis, LSA)이라는 방법을 이용.

```
from gensim.models import LsiModel

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, limi
```



LSA: DTM을 차원축소하여 축소 차원에서 근접 단어들을 토픽으로 묶음

LDA: 단어가 특정 토픽에 존재할 확률과 특정 토픽이 존재할 확률을 결합확률로 추정하여 토픽을 추출

THANK YOU

