

# 자연어처리 Natural Language Processing

김진숙

---

# [ 텍스트 분류 ]

## 텍스트 분류

### ■ 텍스트 분류(Text Classification)

- 자연어 처리 문제 중 가장 대표적이고 많이 접하는 문제
- 자연어 처리 기술을 활용해 특정 텍스트를 사람들이 정한 몇 가지 범주(Class) 중 어느 범주에 속하는지 분류하는 문제
- 이진 분류(Binary Classification), 다중 범주 분류(Multi Class Classification)

### ■ 텍스트 분류의 예시

#### • 스팸 분류

- 일반 메일과 스팸 메일을 분류하는 문제
- 분류 범주(Class): 스팸 메일, 일반 메일 2가지

#### • 감정 분류

- 주어진 글에 대해 긍정적인지 부정적인지 판단하는 문제
- 분류 범주(Class): 긍정, 부정, 중립

#### • 뉴스 기사 분류

- 스포츠, 경제, 사회, 연예 등 다양한 주제의 기사를 주제에 맞게 분류하는 문제

## 텍스트 분류

### ■ 지도학습을 통한 텍스트 분류

- 글(데이터)에 대해 각각 속한 범주에 대한 값(라벨)이 이미 주어져 있고, 주어진 범주로 글을 학습한 후 학습한 결과를 이용해 새로운 글의 범주를 예측하는 방법
- 지도학습 문장 분류 모델
  - 나이브베이지스 분류
  - 서포트벡터머신
  - 신경망
  - 선형 분류
  - 로지스틱 분류
  - 랜덤포레스트

### ■ 비지도학습을 통한 텍스트 분류

- 비지도 학습에서는 데이터만 존재하고, 각 데이터는 범주로 미리 나뉘져 있지 않다.
- 비지도 학습은 특성을 찾아내서 비슷한 데이터끼리 적당한 범주로 만들어 각 데이터를 나눈다.
- 비지도학습 문장 분류 모델
  - k-평균 군집화(k-means Clustering)
  - 계층적 군집화(Hierarchical Clustering)

---

# [ 한글 텍스트 분류 ]

## 한글 텍스트 분류

### ■ 한글 텍스트 분류

- 언어마다 언어적인 특성이 매우 달라서 언어를 처리하는 과정이 다름
- 한글 텍스트를 다루는 방법과 한글 텍스트 분류하는 방법
- NLTK 라이브러리는 한글 지원 안됨, KoNLPy 라이브러리로 한글 텍스트 분류

### ■ 네이버 영화 리뷰 데이터

- 네이버 영화의 사용자 리뷰를 각 영화당 100개씩 모아서 만들어진 데이터
- Id, document(영화 리뷰), label(감정 값(긍정(1) 혹은 부정(0)))로 구성
- 데이터 이름 : Naver Sentiment movie corpus v1.0
- 데이터 용도 : 텍스트 분류 학습을 목적으로 사용
- 데이터 권한 : CCO 1.0
- 데이터 출처 : <https://github.com/e9t/nsmc>

### ■ 데이터 전처리

- 전처리한 데이터를 활용하여 리뷰가 긍정과 부정인지 분류

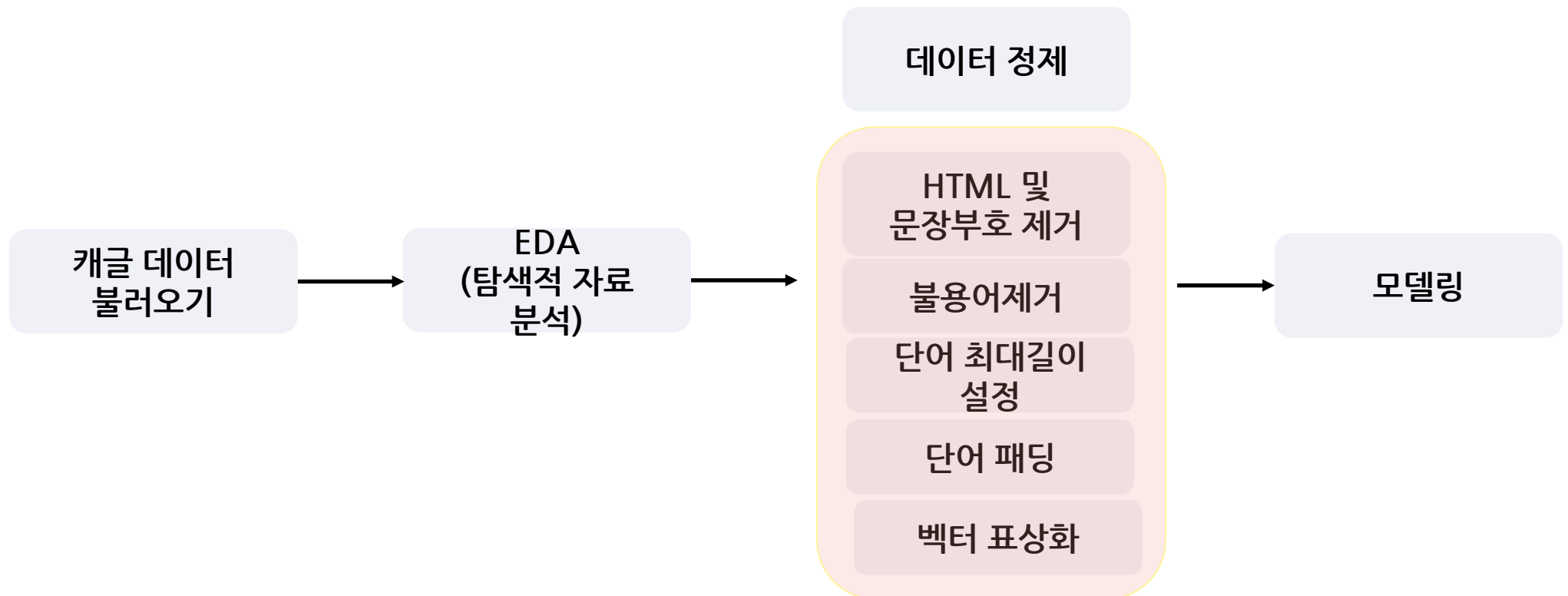
## 텍스트 분류

### ■ 데이터 분석 및 전처리

### • 데이터 처리 과정

#### • 탐색적 데이터 분석(EDA; Exploratory Data Analysis)

- 데이터에 대해 최대한 많은 정보를 뽑아내는 것
- 데이터의 평균값, 중앙값, 최소값, 최대값, 범위, 분포, 이상치(Outlier) 등
- 히스토그램, 그래프 등의 다양한 방법으로 시각화 하면서 데이터에 대한 직관을 얻어야 한다.



## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

- 네이버 영화 리뷰 데이터 다운로드 <https://github.com/e9t/nsmc>
- 데이터의 구성
  - ratings.txt : 전체 리뷰를 모아둔 데이터. 전체 20만개의 데이터로 구성
  - ratings\_train.txt : 학습 데이터. 총 15만개의 데이터로 구성
  - ratings\_test.txt : 테스트 데이터. 총 5만개의 데이터로 구성

#### 1) 데이터 탐색 분석(EDA)

##### 라이브러리 불러오기

```
1 import numpy as np
2 import pandas as pd
3 import os
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from wordcloud import WordCloud
7 %matplotlib inline
```

##### 데이터 불러오기

```
1 DATA_IN_PATH = './data_in/'
```

- 데이터 크기 확인

```
1 print("파일 크기 : ")
2 for file in os.listdir(DATA_IN_PATH):
3     if 'txt' in file :
4         print(file.ljust(30) + str(round(os.path.getsize(DATA_IN_PATH + file) / 1000000, 2)) + 'MB')
```

파일 크기 :	
ratings.txt	19.52MB
ratings_test.txt	4.89MB
ratings_train.txt	14.63MB



## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

- 데이터 불러오기

```
1 train_data = pd.read_csv(DATA_IN_PATH + 'ratings_train.txt', header = 0, delimiter = '₩t', quoting = 3)
2 train_data.head()
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1

- 데이터의 갯수 확인

```
1 print('전체 학습데이터의 개수: {}'.format(len(train_data)))
```

전체 학습데이터의 개수: 150000

## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

#### 리뷰의 문자 길이로 히스토그램 그리기

- 각 리뷰의 길이 확인

```
1 train_length = train_data['document'].astype(str).apply(len)
```

```
1 train_length.head()
```

```
0    19
```

```
1    33
```

```
2    17
```

```
3    29
```

```
4    61
```

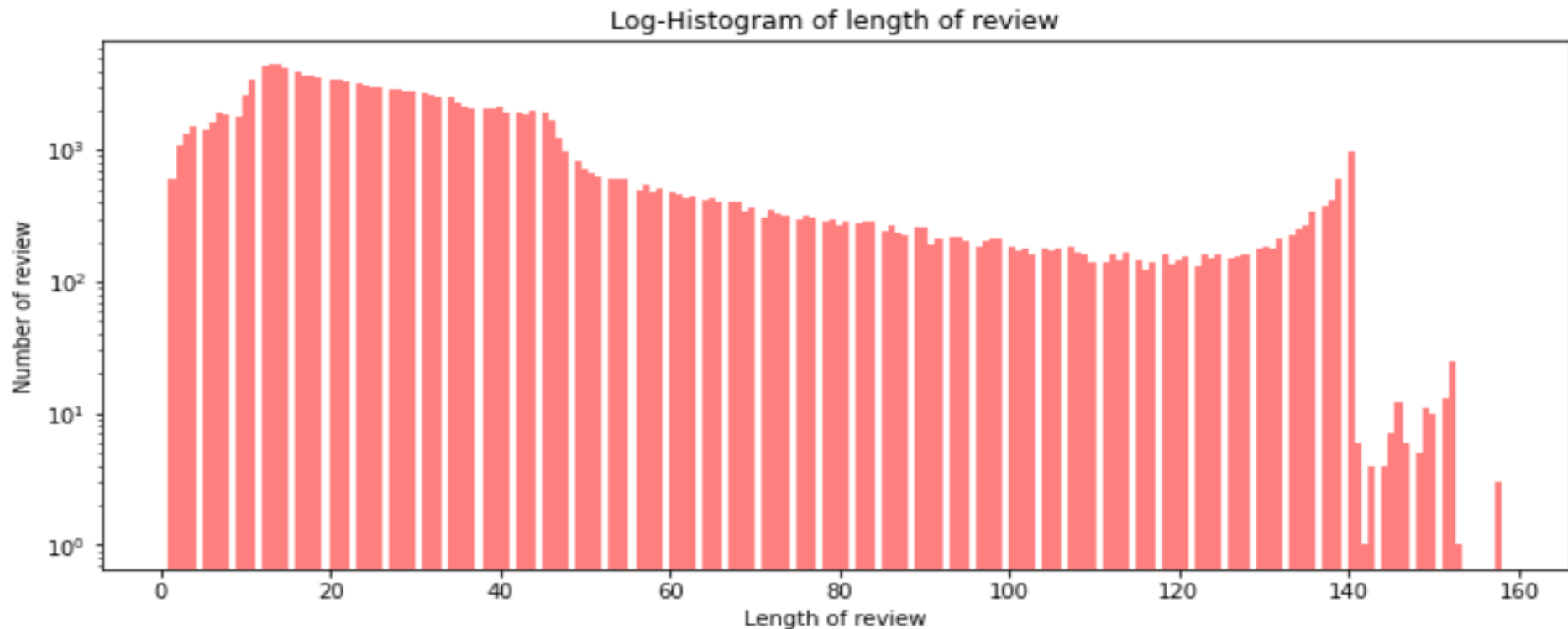
```
Name: document, dtype: int64
```

- 히스토그램 그리기

```
1 plt.figure(figsize=(12, 5))
2 plt.hist(train_length, bins=200, alpha=0.5, color='r', label='word')
3 plt.yscale('log', nonposy='clip')
4 # 그래프 제목
5 plt.title('Log-Histogram of length of review')
6 # 그래프 x 축 라벨
7 plt.xlabel('Length of review')
8 # 그래프 y 축 라벨
9 plt.ylabel('Number of review')
```

## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석



- 그래프 해석)
- 리뷰의 길이가 0부터 140까지 고르게 분포되어 있음
- 20자 이하에 많이 분포되어 있다가 길이가 길어질수록 점점 글자 수가 적어지는데 140자 부근에서 갑자기 글자수가 많아지는 모양
- 140자 글자 제한(한글기준)이 있는 데이터이기 때문에 최대 글자수에 조금 모여있는 형태임

## 한글 텍스트 분류

### 데이터 전처리 및 분석

#### 리뷰 글자 길이 기술통계

```
1 print('리뷰 길이 최대 값: {}'.format(np.max(train_length)))
2 print('리뷰 길이 최소 값: {}'.format(np.min(train_length)))
3 print('리뷰 길이 평균 값: {:.2f}'.format(np.mean(train_length)))
4 print('리뷰 길이 표준편차: {:.2f}'.format(np.std(train_length)))
5 print('리뷰 길이 중간 값: {}'.format(np.median(train_length)))
6 # 사분위의 대한 경우는 0~100 스케일로 되어있음
7 print('리뷰 길이 제 1 사분위: {}'.format(np.percentile(train_length, 25)))
8 print('리뷰 길이 제 3 사분위: {}'.format(np.percentile(train_length, 75)))
```

리뷰 길이 최대 값: 158  
리뷰 길이 최소 값: 1  
리뷰 길이 평균 값: 35.24  
리뷰 길이 표준편차: 29.58  
리뷰 길이 중간 값: 27.0  
리뷰 길이 제 1 사분위: 16.0  
리뷰 길이 제 3 사분위: 42.0

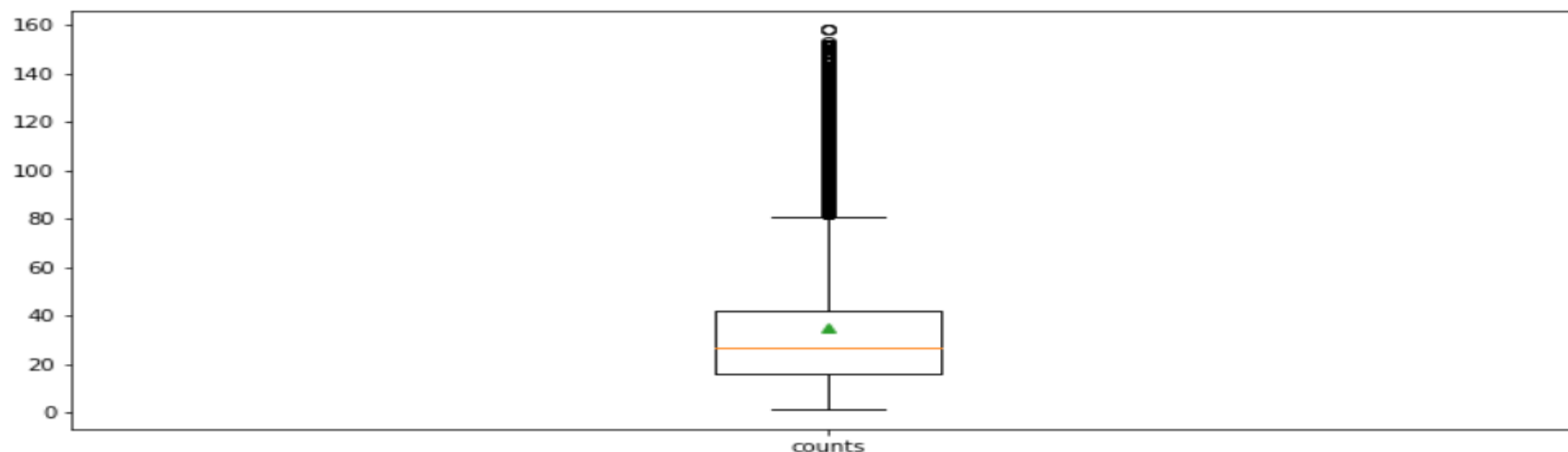
- 해석)
- 리뷰 길이의 최대값은 158
- 최대 글자수가 140자이긴 하지만 특수문자등으로 인해 좀 더 긴 데이터가 포함되어 있는 것으로 보임
- 리뷰 길이의 최소값은 1이고 평균은 35자 정도, 중간값은 27자로 평균보다 좀 더 작은 값을 가진다.

## 한글 텍스트 분류

### 데이터 전처리 및 분석

#### 리뷰 글자 길이의 박스 플롯

```
1 plt.figure(figsize=(12, 5))
2 # 박스플롯 생성
3 # 첫번째 파라미터: 여러 분포에 대한 데이터 리스트를 입력
4 # labels: 입력한 데이터에 대한 라벨
5 # showmeans: 평균값을 마크함
6
7 plt.boxplot(train_length,
8             labels=['counts'],
9             showmeans=True)
```



- 해석)
- 중간값과 평균은 아래쪽에 위치
- 일부 데이터가 긴 데이터가 꽤 존재한다.

## ■ 데이터 전처리 및 분석

- 리뷰에 해당하는 document 컬럼만 train\_review로 가져오기

```
1 train_review = [review for review in train_data['document'] if type(review) is str]
```

```
1 wordcloud = WordCloud(font_path = '"c:/windows/font/NanumGothic.ttf').generate(' '.join(train_review))
2
3 import matplotlib.pyplot as plt
4
5 plt.figure(figsize=(10,6))
6 plt.imshow(wordcloud, interpolation="bilinear")
7 plt.axis("off")
8 plt.show()
```

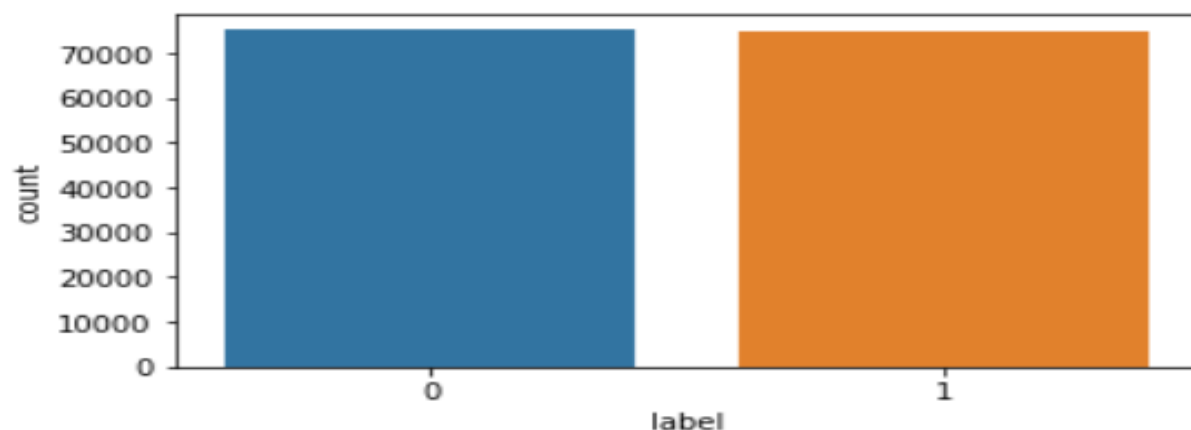


## 한글 텍스트 분류

### 데이터 전처리 및 분석

#### 긍정과 부정 라벨값 비율

```
1 fig, axe = plt.subplots(ncols=1)
2 fig.set_size_inches(6, 3)
3 sns.countplot(train_data['label'])
```



#### • 긍정과 부정 리뷰 개수

```
1 print("긍정 리뷰 개수: {}".format(train_data['label'].value_counts()[1]))
2 print("부정 리뷰 개수: {}".format(train_data['label'].value_counts()[0]))
```

긍정 리뷰 개수: 74827  
부정 리뷰 개수: 75173

## 한글 텍스트 분류

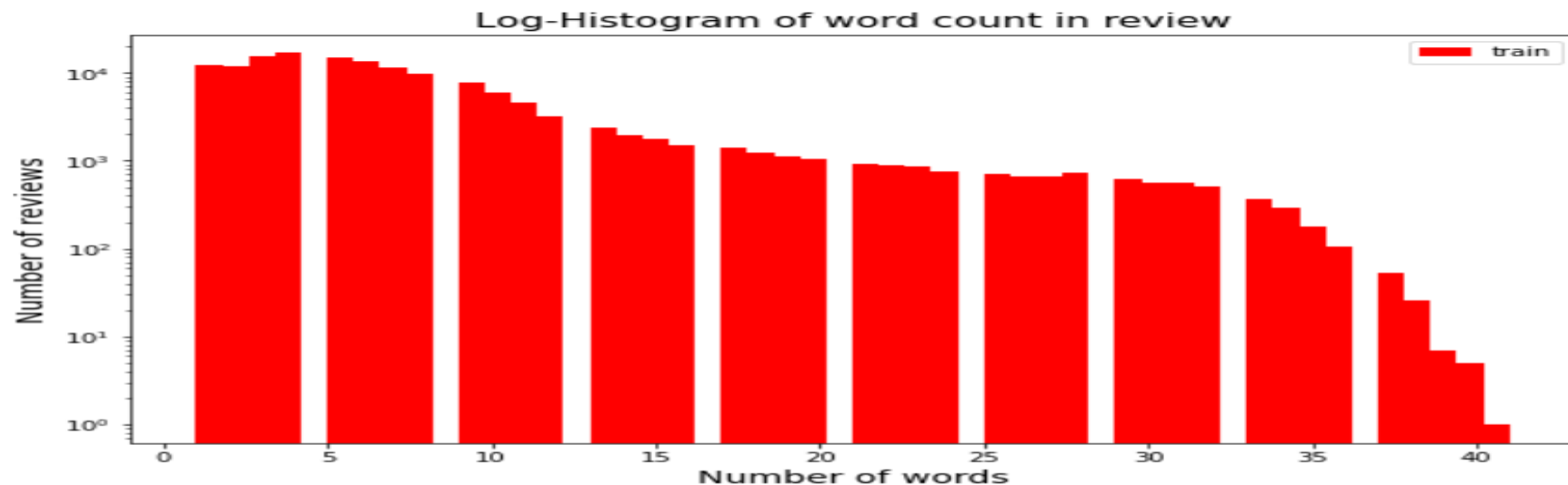
### 데이터 전처리 및 분석

#### 각 리뷰의 단어 수 확인

- 각 리뷰를 띄어쓰기 기준으로 나누고
- 그 갯수를 하나의 변수로 할당하고
- 그 값을 사용해 히스토그램으로 그리기

```
1 train_word_counts = train_data['document'].astype(str).apply(lambda x: len(x.split(' ')))
```

```
1 plt.figure(figsize=(10,6))
2 plt.hist(train_word_counts, bins=50, facecolor='r',label='train')
3 plt.title('Log-Histogram of word count in review', fontsize=15)
4 plt.yscale('log', nonposy='clip')
5 plt.legend()
6 plt.xlabel('Number of words', fontsize=14)
7 plt.ylabel('Number of reviews', fontsize=14)
```



- 그래프 해석)
- 각 리뷰의 단어 수는 대부분 5개 정도에 분포되어 있음
- 각 리뷰의 단어 수가 30개 이상 부터는 급격히 줄어듦



## 한글 텍스트 분류

### 데이터 전처리 및 분석

#### 리뷰 단어 개수 기술통계

```
1 print('리뷰 단어 개수 최대 값: {}'.format(np.max(train_word_counts)))
2 print('리뷰 단어 개수 최소 값: {}'.format(np.min(train_word_counts)))
3 print('리뷰 단어 개수 평균 값: {:.2f}'.format(np.mean(train_word_counts)))
4 print('리뷰 단어 개수 표준편차: {:.2f}'.format(np.std(train_word_counts)))
5 print('리뷰 단어 개수 중간 값: {}'.format(np.median(train_word_counts)))
6 # 사분위의 대한 경우는 0~100 스케일로 되어있음
7 print('리뷰 단어 개수 제 1 사분위: {}'.format(np.percentile(train_word_counts, 25)))
8 print('리뷰 단어 개수 제 3 사분위: {}'.format(np.percentile(train_word_counts, 75)))
```

리뷰 단어 개수 최대 값: 41  
리뷰 단어 개수 최소 값: 1  
리뷰 단어 개수 평균 값: 7.58  
리뷰 단어 개수 표준편차: 6.51  
리뷰 단어 개수 중간 값: 6.0  
리뷰 단어 개수 제 1 사분위: 3.0  
리뷰 단어 개수 제 3 사분위: 9.0

- 해석)
- 평균 7~8개의 단어 개수를 가짐
- 중간값의 경우 6개 정도
- 글자 수 제한이 있어서 영어 데이터에 비해 단어의 개수가 적은편
- 모델에 적용할 최대 단어수를 8개로 설정해도 크게 무리가 없을 것으로 보임

## 한글 텍스트 분류

### ▪ 데이터 전처리 및 분석

#### 리뷰에 특수 문자 유무 확인

```
1 qmarks = np.mean(train_data['document'].astype(str).apply(lambda x: '?' in x)) # 물음표가 구두점으로 쓰임
2 fullstop = np.mean(train_data['document'].astype(str).apply(lambda x: '.' in x)) # 마침표
3
4 print('물음표가있는 질문: {:.2f}%'.format(qmarks * 100))
5 print('마침표가 있는 질문: {:.2f}%'.format(fullstop * 100))
```

물음표가있는 질문: 8.25%

마침표가 있는 질문: 51.76%

- 해석)
- 영화리뷰 데이터라 물음표는 거의 포함되어 있지 않고
- 마침표의 경우는 절반 정도의 리뷰에 포함되어 있음

## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

#### 2) 데이터 전처리

#### 라이브러리 불러오기

```
1 import numpy as np
2 import pandas as pd
3 import re
4 import json
5 from konlpy.tag import Okt
6 from tensorflow.python.keras.preprocessing.sequence import pad_sequences
7 from tensorflow.python.keras.preprocessing.text import Tokenizer
```

#### 판다스 데이터프레임으로 데이터 불러오기

```
1 DATA_IN_PATH = './data_in/'
2
3 train_data = pd.read_csv(DATA_IN_PATH + 'ratings_train.txt', header=0, delimiter='#t', quoting=3)
4
5 print(train_data.head())
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1

## 한글 텍스트 분류

### 데이터 전처리 및 분석

#### 불러온 데이터 확인하기

```
1 train_data['document'][:5]
```

```
0                아 더빙.. 진짜 짜증나네요 목소리
1                흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나
2                너무재밌었다그래서보는것을추천한다
3                교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정
4 사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...
Name: document, dtype: object
```

- 영어와 달리 HTML 태그가 보이지 않음
- 특수 문자 혹은 숫자는 전처리가 필요해 보임

#### 첫번째 리뷰로 전처리 진행

- 정규표현식을 이용하여 한글이 아닌 것은 모두 제거

```
1 review_text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ#ws]", "", train_data['document'][0])
2 print(review_text)
```

아 더빙 진짜 짜증나네요 목소리

## 한글 텍스트 분류

### ▪ 데이터 전처리 및 분석

#### 불용어 제거

- 문장을 단어로 나누기: Okt 형태소 분석기 사용

```
1 from konlpy.tag import Okt
2
3 okt = Okt()
4 review_text = okt.morphs(review_text, stem=True)
5 print(review_text)
```

['아', '더빙', '진짜', '짜증나다', '목소리']

- 불용어 사전을 만들어서 불용어 제거

```
1 # 불용어 사전을 만들어서 불용어 제거
2 stop_words = set(['은', '는', '이', '가', '하', '아', '것', '들', '의', '있', '되', '수', '보', '주', '등', '한'])
3 clean_review = [token for token in review_text if not token in stop_words]
4 print(clean_review)
```

['더빙', '진짜', '짜증나다', '목소리']

## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

#### 전처리의 전과정을 함수로 생성: preprocessing()

- 함수의 인자
  - review : 전처리할 텍스트
  - okt : okt 객체를 반복적으로 생성하지 않고 미리 생성후 인자로 받는다.
  - remove\_stopword : 불용어를 제거할지 선택 기본값은 False
  - stop\_word : 불용어 사전은 사용자가 직접 입력해야함 기본값은 비어있는 리스트

```
1 stop_words = set(['은', '는', '이', '가', '하', '아', '것', '들', '의', '있', '되', '수', '보', '주', '등', '한'])
```

```
1 from konlpy.tag import Okt
2 okt = Okt()
3
4 def preprocessing(review, okt, remove_stopwords = False, stop_words = []):
5
6     # 1. 한글 및 공백을 제외한 문자 모두 제거.
7     review_text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣㅗㅛㅜㅝ]", "", review)
8
9     # 2. okt 객체를 활용해서 형태소 단위로 나눈다.
10    word_review = okt.morphs(review_text, stem=True)
11
12    if remove_stopwords:
13        # 불용어 제거(선택적)
14        word_review = [token for token in word_review if not token in stop_words]
15        # 단어 리스트를 공백을 붙여서 하나의 글로 합친다.
16        clean_review = ' '.join(word_review)
17
18    return clean_review
```

## 한글 텍스트 분류

### ▪ 데이터 전처리 및 분석

#### 학습 데이터 전체에 함수를 이용한 전처리 진행

```
1 stop_words = [ '은', '는', '이', '가', '하', '아', '것', '들', '의', '있', '되', '수', '보', '주', '등', '한' ]
2 okt = Okt()
3 clean_train_review = []
4
5 for review in train_data['document']:
6     # 비어있는 데이터에서 멈추지 않도록 string인 경우만 진행
7     if type(review) == str:
8         clean_train_review.append(preprocessing(review, okt, remove_stopwords = True, stop_words=stop_words))
9     else:
10        clean_train_review.append([]) #string이 아니면 비어있는 값 추가
```

```
1 clean_train_review[:4] # 앞의 4개 데이터 확인
```

```
['더빙 진짜 짜증나다 목소리',
 '흠 포스터 보고 초딩 영화 줄 오버 연기 조차 가볍다 않다',
 '너 무재 밌었 다그 래서 보다 추천 다',
 '교도소 이야기 구먼 솔직하다 재미 없다 평점 조정']
```

## 한글 텍스트 분류

### ▪ 데이터 전처리 및 분석

#### 테스트 데이터도 학습 데이터와 동일하게 전처리 진행

```
1 test_data = pd.read_csv(DATA_IN_PATH + 'ratings_test.txt', header=0, delimiter='##t', quoting=3 )
2
3 clean_test_review = []
4
5 for review in test_data['document']:
6     # 비어있는 데이터에서 멈추지 않도록 string인 경우만 진행
7     if type(review) == str:
8         clean_test_review.append(preprocessing(review, okt, remove_stopwords = True, stop_words=stop_words))
9     else:
10        clean_test_review.append([]) #string이 아니면 비어있는 값 추가
```

```
1 clean_test_review[:4]
```

['굳다 ㅋ', '', '뭐 야 평점 나쁘다 않다 점 짜다 리 더 더욱 아니다', '지루하다 않다 완전 막장 임 돈 주다 보기 예는']



## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

전처리가 끝난 데이터를 텍스트 데이터와 인덱스로 벡터화한 데이터로 저장

#### 1) 텍스트 형태로 저장 : 데이터프레임 --> csv 파일로 저장

```
1 # 전처리 과정이 끝난 전체 데이터 -> 데이터프레임으로 저장
2 # 학습 데이터
3 clean_train_df = pd.DataFrame({'id': train_data['id'], 'document': clean_train_review, 'label': train_data['label']})
4
5 # 테스트 데이터
6 clean_test_df = pd.DataFrame({'id': test_data['id'], 'document': clean_test_review, 'label': test_data['label']})
7 test_id = np.array(test_data['id'])
```

```
1 clean_train_df.head()
```

	id	document	label
0	9976970	더빙 진짜 짜증나다 목소리	0
1	3819312	흠 포스터 보고 초딩 영화 줄 오버 연기 조차 가볍다 않다	1
2	10265843	너 무재 밌었 다그 래서 보다 추천 다	0
3	9045019	교도소 이야기 구면 솔직하다 재미 없다 평점 조정	0
4	6483659	사이 문페 그 익살스럽다 연기 돋보이다 영화 스파이더맨 에서 늑다 보이다 하다 커스...	1

## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

```
1 # 정제된 텍스트를 csv 형태로 저장
2 TRAIN_CLEAN_DATA = 'nsmc_train_clean.csv'
3 TEST_CLEAN_DATA = 'nsmc_test_clean.csv'
4 clean_train_df.to_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA, index = False)
5 clean_test_df.to_csv(DATA_IN_PATH + TEST_CLEAN_DATA, index = False)
```

## 2) 학습 데이터와 테스트 데이터를 인덱스 벡터로 바꾼 뒤 패딩 처리

### 덴서플로의 전처리 모듈 사용

- 토큰나이징 객체 생성 후 학습 데이터에만 적용하고
- 해당 객체를 사용하여 학습 데이터와 테스트 데이터를 인덱스 벡터로 생성
- 해당 데이터들을 패딩

```
1 # 토큰나이징
2 tokenizer = Tokenizer()
3 tokenizer.fit_on_texts(clean_train_review)
4 train_sequences = tokenizer.texts_to_sequences(clean_train_review)
5 test_sequences = tokenizer.texts_to_sequences(clean_test_review)
6
7 word_vocab = tokenizer.word_index # 단어 사전 형태
8 word_vocab[ "<PAD>" ] = 0 # word_index에 <PAD>값이 정의되어 있지 않으므로 <PAD>의 인덱스 값을 0으로 입력
```

## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

```
1 # 인덱스로 구성된 벡터 첫번째 값 : 각 단어의 인덱스로 바뀜
2 print(train_sequences[0])
3 print(test_sequences[0])
```

```
[463, 20, 265, 664]
[648, 93]
```

```
1 # 각 인덱스가 어떤 단어를 의미하는 지 확인: 단어사전 필요
2 print(word_vocab)
```

```
{'영화': 1, '보다': 2, '하다': 3, '에': 4, '을': 5, '도': 6, '를': 7, '없다': 8, '이다': 9, '있다': 10, '다': 13, '정말': 14, '재밌다': 15, '되다': 16, '적': 17, '만': 18, '같다': 19, '진짜': 20, '으로': 21}
```

```
1 # 전체 데이터에서 사용된 단어의 총 개수
2 print("전체 단어 개수: ", len(word_vocab))
```

```
전체 단어 개수: 43757
```

## 한글 텍스트 분류

### ■ 데이터 전처리 및 분석

#### 패딩 과정 - 데이터의 길이 맞추기

- 각 데이터의 길이가 서로 다른데 이 길이를 하나로 통일해야 모델에 바로 적용 가능하기 때문에
- 특정 길이를 최대 길이로 정하고 더 긴 데이터의 경우는 뒷부분을 자르고
- 더 짧은 데이터의 경우에는 0 값으로 패딩하는 작업 진행
- 텐서플로우의 전처리 모듈 이용

```
: 1 MAX_SEQUENCE_LENGTH = 8 # 문장 최대 길이 - 데이터분석과정에서 단어의 평균개수가 약 8개 정도
2
3 train_inputs = pad_sequences(train_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post') # 학습 데이터를 벡터화
4 train_labels = np.array(train_data['label']) # 학습 데이터의 라벨
5
6 test_inputs = pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post') # 테스트 데이터를 벡터화
7 test_labels = np.array(test_data['label']) # 테스트 데이터의 라벨
```

```
: 1 print('Shape of train data: ', train_inputs.shape)
2 print('Shape of test data: ', test_inputs.shape)
```

Shape of train data: (150000, 8)

Shape of test data: (50000, 8)

## 한글 텍스트 분류

### ▪ 데이터 전처리 및 분석

#### 인덱스 벡터 + 패딩 처리한 데이터 저장

```
1 DATA_IN_PATH = './data_in/'
2 TRAIN_INPUT_DATA = 'nsmc_train_input.npy'
3 TRAIN_LABEL_DATA = 'nsmc_train_label.npy'
4 TEST_INPUT_DATA = 'nsmc_test_input.npy'
5 TEST_LABEL_DATA = 'nsmc_test_label.npy'
6 DATA_CONFIGS = 'data_configs.json'
7
8 data_configs = {}
9
10 data_configs['vocab'] = word_vocab
11 data_configs['vocab_size'] = len(word_vocab) # vocab size 추가
12
13 import os
14 # 저장하는 디렉토리가 존재하지 않으면 생성
15 if not os.path.exists(DATA_IN_PATH):
16     os.makedirs(DATA_IN_PATH)
17
18 # 전처리 된 학습 데이터를 넘파이 형태로 저장
19 np.save(open(DATA_IN_PATH + TRAIN_INPUT_DATA, 'wb'), train_inputs)
20 np.save(open(DATA_IN_PATH + TRAIN_LABEL_DATA, 'wb'), train_labels)
21
22 # 전처리 된 테스트 데이터를 넘파이 형태로 저장
23 np.save(open(DATA_IN_PATH + TEST_INPUT_DATA, 'wb'), test_inputs)
24 np.save(open(DATA_IN_PATH + TEST_LABEL_DATA, 'wb'), test_labels)
25
26 # 데이터 사전을 json 형태로 저장
27 json.dump(data_configs, open(DATA_IN_PATH + DATA_CONFIGS, 'w'), ensure_ascii=False)
```

## 텍스트 분류

### ■ 분류 모델링

- 머신러닝 모델 – 선형회귀모델, 랜덤포레스트모델
- 딥러닝 모델 – 합성곱 신경망(CNN), 순환 신경망(RNN)

### ■ 회귀 모델

#### • 선형 회귀 모델(Linear Regression)

- 종속변수(Y)와 한 개 이상의 독립변수(X)와의 선형 상관 관계를 모델링하는 회귀 분석 기법

1) 단순 선형 회귀:  $y = wx + b$   $w$ : 계수(가중치),  $b$ : 절편(편향)

2) 다중 선형 회귀:  $y = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n + b$

#### • 선형 회귀의 비용 함수

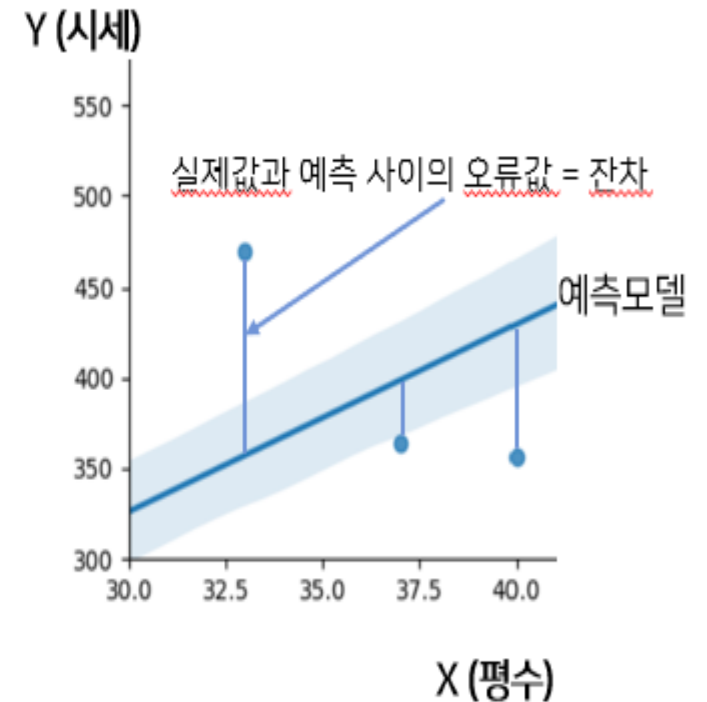
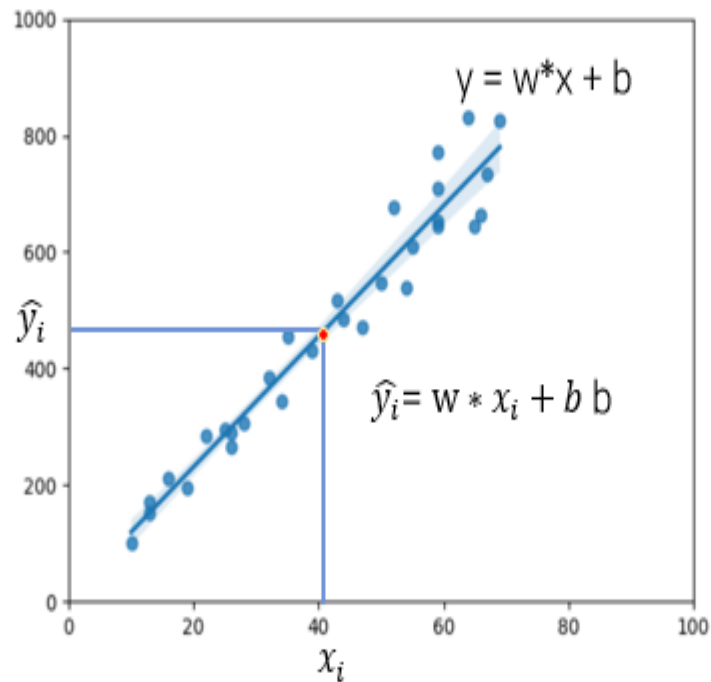
$$\text{Cost}_{\text{tr}} = \sum_i (y_i - \hat{y}_i)^2$$
$$\hat{y}_i = b + wx_i$$

- 실제 참값과 회귀 모델이 출력한 예측값 사이의 잔차의 제곱의 합을 최소화하는  $w$ (계수)를 구하는 것이 목적  
-> Least Square, 최소 제곱법

## 텍스트 분류

- 선형 회귀 모델
  - 주택 가격 예측: 주택 시세가 평수로만 결정된다고 가정

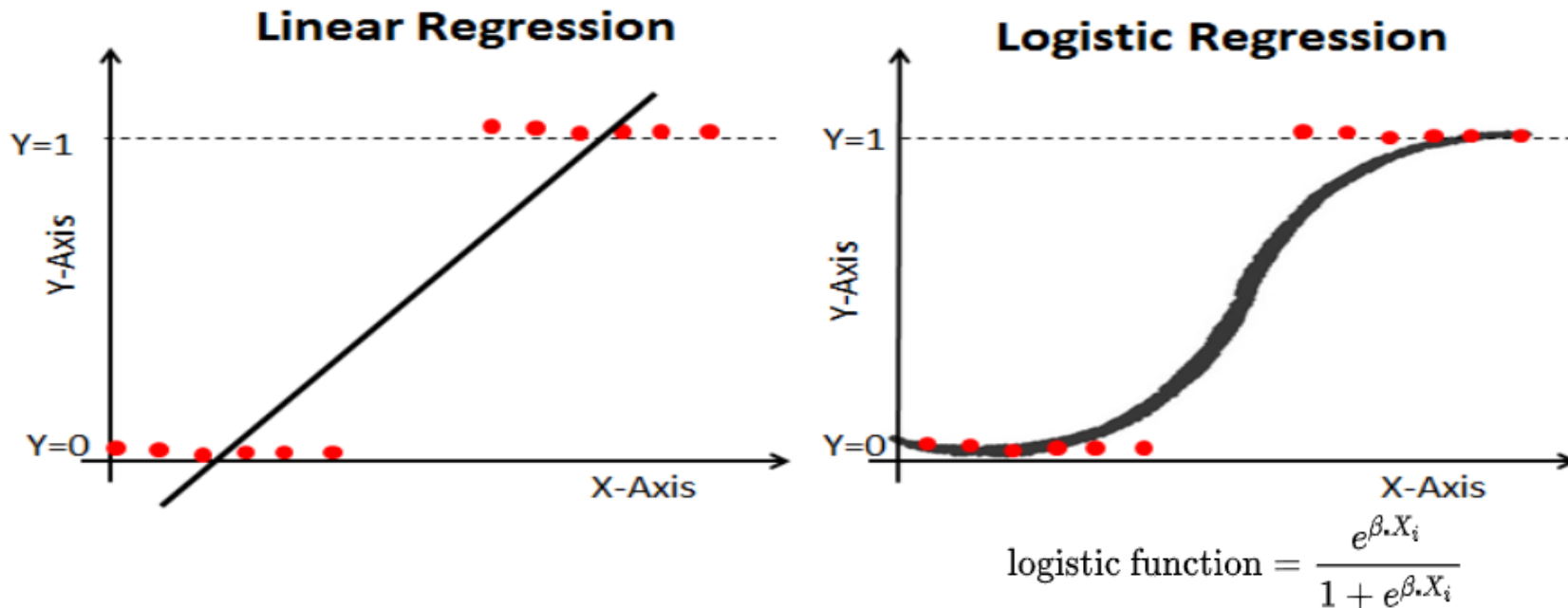
	Xi(평수)	Yi(시세)
0	10	102
1	22	284
2	32	384
3	64	832
4	50	547
<u>NewData</u>	<u>Xi</u>	<u>Yi</u>



## 텍스트 분류

### 로지스틱 회귀 모델

- 선형 모델의 결과값에 로지스틱 함수를 적용하여 0 ~ 1 사이의 값을 갖게 해서 확률로 표현한 모델
- 이렇게 나온 결과를 통해 1에 가까우면 정답이 1이라고 예측하고, 0에 가까울 경우 0으로 예측
- 로지스틱 회귀 모델에 가지고 텍스트 분류 진행
- 선형회귀 분석과 유사하지만 해당 데이터 결과가 특정 분류(Classification)로 나눔
- 로지스틱 회귀에는 종속변수가 이항적 문제에 사용



### 단어 임베딩 방법

- Word2Vec을 통한 단어 임베딩 벡터로 만드는 방법
- tf-idf를 통해 임베딩 벡터로 만드는 방법



## 한글 텍스트 분류

### ■ 분류 모델링

#### 로지스틱 회귀 모델 with TF-IDF

#### 라이브러리 불러오기

```
1 import os
2 import pandas as pd
3 import numpy as np
4
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.model_selection import train_test_split
7 from sklearn.linear_model import LogisticRegression
```

```
1 # 경고메세지 무시
2 import warnings
3 warnings.filterwarnings(action='ignore')
```

#### 데이터 불러오기 - 텍스트 형태

```
1 DATA_IN_PATH = './data_in/'
2 DATA_OUT_PATH = './data_out/'
3 TRAIN_CLEAN_DATA = 'nsmc_train_clean.csv'
4 TEST_CLEAN_DATA = 'nsmc_test_clean.csv'
5
6 RANDOM_SEED = 42
7 TEST_SPLIT = 0.2
```

```
1 train_df = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA )
2 test_df = pd.read_csv(DATA_IN_PATH + TEST_CLEAN_DATA )
```

```
1 # 결측치 제거
2 train_df = train_df.dropna()
3 test_df = test_df.dropna()
```

## 한글 텍스트 분류

### ■ 분류 모델링

#### TF-IDF 벡터화

- min\_df : 설정한 값보다 특정 토큰의 df 값이 적게 나오면 벡터화 과정에서 제거한다는 의미
- max\_df : 설정한 값보다 특정 토큰의 df 값이 크게 나오면 벡터화 과정에서 제거한다는 의미
- ngram\_range : 빈도의 기본 단위를 어느 범위의 n-gram으로 설정할 것인지를 지정하는 인자

```
1 from konlpy.tag import Okt
2
3 okt = Okt()
4 def okt_tokenizer(text):
5     # 입력 인자로 들어온 text 를 형태소 단어로 토큰화 하여 list 객체 반환
6     tokens_ko = okt.morphs(text)
7     return tokens_ko
```

```
1 okt_tokenizer("꼭 대박나세요")
```

['꼭', '대박나세요']

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.model_selection import GridSearchCV
4
5 # Okt 객체의 morphs( ) 객체를 이용한 tokenizer를 사용. ngram_range는 (1,2)
6 tfidf_vect = TfidfVectorizer(tokenizer=okt_tokenizer, ngram_range=(1,2), min_df=3, max_df=0.9)
7 tfidf_vect.fit(train_df['document'])
8 tfidf_matrix_train = tfidf_vect.transform(train_df['document'])
```

## 한글 텍스트 분류

### ■ 분류 모델링

#### 로지스틱 회귀 모델 분류 & GridSearchCV

```
1 # Logistic Regression 을 이용하여 감성 분석 Classification 수행.
2 lg_clf = LogisticRegression(random_state=0)
3
4 # Parameter C 최적화를 위해 GridSearchCV 를 이용.
5 params = { 'C': [1 ,3.5, 4.5, 5.5, 10 ] }
6 grid_cv = GridSearchCV(lg_clf , param_grid=params , cv=3 ,scoring='accuracy', verbose=1 )
7 grid_cv.fit(tfidf_matrix_train , train_df['label'] )
8 print(grid_cv.best_params_ , round(grid_cv.best_score_,4))
```

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

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 15 out of 15 | elapsed: 48.9s finished
```

```
{'C': 4.5} 0.8612
```

#### 테스트 데이터에 TF-IDF 벡터 변환 후 정확도 평가

```
1 from sklearn.metrics import accuracy_score
2
3 # 학습 데이터를 적용한 TfidfVectorizer를 이용하여 테스트 데이터를 TF-IDF 값으로 Feature 변환함.
4 tfidf_matrix_test = tfidf_vect.transform(test_df['document'])
5
6 # classifier 는 GridSearchCV에서 최적 파라미터로 학습된 classifier를 그대로 이용
7 best_estimator = grid_cv.best_estimator_
8 preds = best_estimator.predict(tfidf_matrix_test)
9
10 print('Logistic Regression 정확도: ',accuracy_score(test_df['label'],preds))
```

Logistic Regression 정확도: 0.8650835283673635

---

# [ 영어 텍스트 분류 ]

## 텍스트 분류

### ■ 감성 분류

- 영화 리뷰 데이터를 분석 한 후 감정을 긍정 혹은 부정으로 예측하는 모델 생성 후 새로운 리뷰가 긍정인지 부정인지 예측
- 영어로 작성된 영화 리뷰 데이터 감성 분류, 한글로 작성된 영화 리뷰 데이터 감성 분류

### ■ 영어 텍스트 분류

- 데이터 이름 : Bag of Words Meets Bags of popcorn('워드 팝콘')
- 데이터 용도 : 텍스트 분류 학습을 목적으로 사용, 데이터 권한: MIT 권한
- 데이터 출처 : <https://www.Kaggle.com/c/word2vec-nlp-tutorial/data>

### ■ 워드 팝콘

- 인터넷 영화 데이터베이스(IMDB)에서 나온 영화 평점 데이터를 활용한 캐글 문제
- 각 데이터는 영화 리뷰 텍스트, 평점에 따른 감정 값(긍정 혹은 부정)으로 구성되어 있다.

### ■ 목표

- 데이터 전처리
- 데이터 탐색적 분석
- 분류 모델링

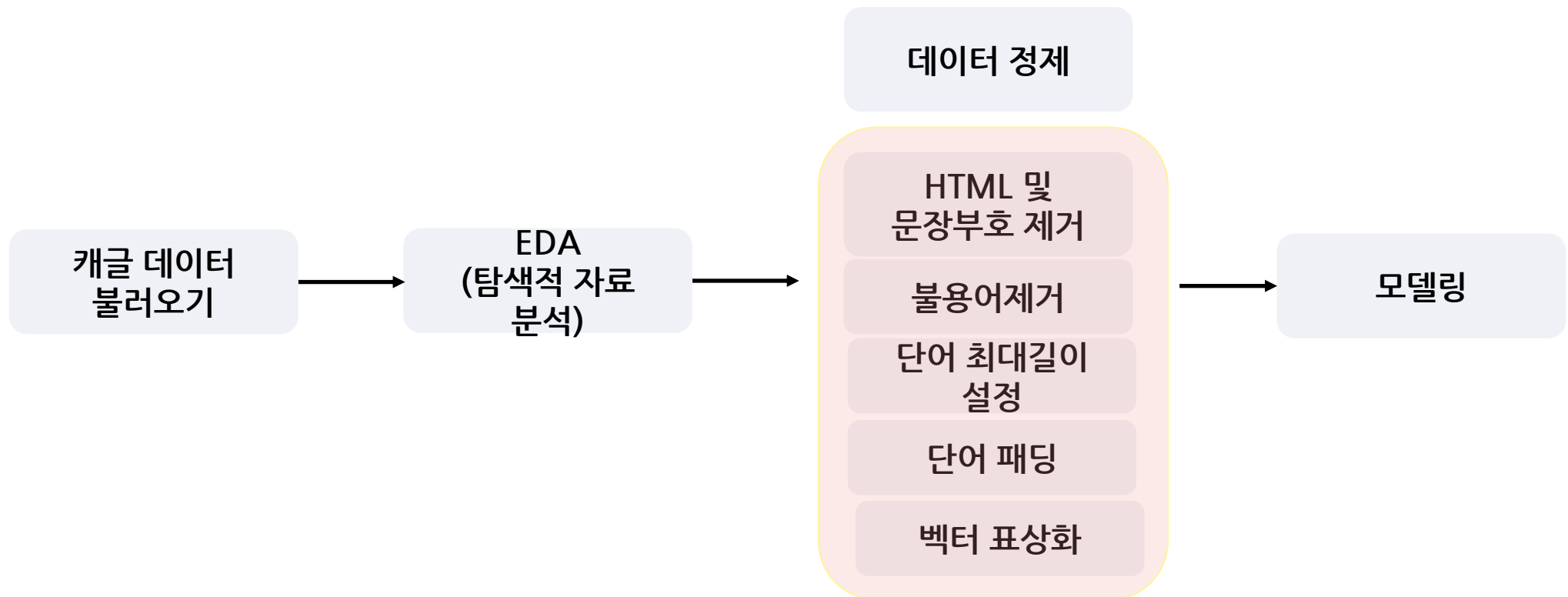
## 텍스트 분류

### ■ 데이터 분석 및 전처리

### • 데이터 처리 과정


#### • 탐색적 데이터 분석(EDA; Exploratory Data Analysis)

- 데이터에 대해 최대한 많은 정보를 뽑아내는 것
- 데이터의 평균값, 중앙값, 최소값, 최대값, 범위, 분포, 이상치(Outlier) 등
- 히스토그램, 그래프 등의 다양한 방법으로 시각화 하면서 데이터에 대한 직관을 얻어야 한다.




## 텍스트 분류

- 데이터 분석 및 전처리
  - 데이터 내려받기

 Getting Started Prediction Competition

### Bag of Words Meets Bags of Popcorn





Use Google's Word2Vec for movie reviews

 Kaggle · 577 teams · 6 years ago



[Overview](#) [Data](#) [Code](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#) [My Submissions](#) [Late Submission](#)

#### Data Explorer

51.85 MB

-  [labeledTrainData.tsv.zip](#)
-  [sampleSubmission.csv](#)
-  [testData.tsv.zip](#)
-  [unlabeledTrainData.tsv.zip](#)


#### Summary

- ▶  4 files
- ▶  2 columns

[Download All](#)

#### < labeledTrainData.tsv.zip (12.96 MB)

Unable to show preview



Previews for binary data are not supported

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 1) 데이터 불러오기 및 분석하기 ¶

영화 리뷰 데이터를 불러온 후 탐색적 데이터 분석 과정을 진행한다.  
데이터 분석은 다음의 순서로 진행한다.

- 데이터의 개수
- 각 리뷰의 문자 길이 분포
- 많이 사용된 단어
- 긍정, 부정 데이터의 분포
- 각 리뷰의 단어 개수 분포
- 특수문자 및 대, 소문자 비율

```
1 import numpy as np
2 import pandas as pd
3 import os
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 %matplotlib inline
```



## 텍스트 분류

### 데이터 분석 및 전처리

#### 데이터 불러오기

```
1 train_data = pd.read_csv('./data_in/labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
2 train_data.head()
```

	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...
3	"3630_4"	0	"It must be assumed that those who praised thi...
4	"9495_8"	1	"Superbly trashy and wondrously unpretentious ...

- id, sentiment, review 로 구분, 각 review에 대한 감정(sentiment)이 긍정(1), 부정(0)로 표시

#### 데이터 개수

```
1 print('전체 학습데이터의 개수: {}'.format(len(train_data)))
```

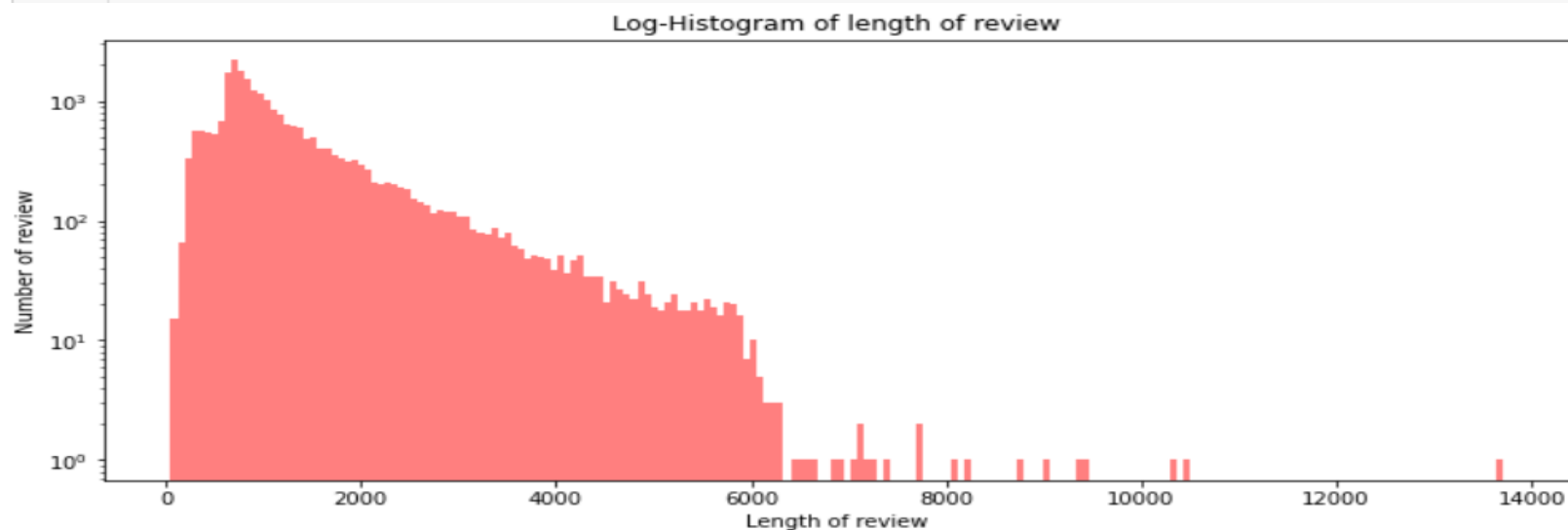
전체 학습데이터의 개수: 25000

## 텍스트 분류

### 데이터 분석 및 전처리

#### 리뷰 문자 길이에 대한 히스토그램

```
1 plt.figure(figsize=(12, 5))
2 plt.hist(train_length, bins=200, alpha=0.5, color='r', label='word')
3 plt.yscale('log', nonposy='clip')
4 plt.title('Log-Histogram of length of review') # 그래프 제목
5 plt.xlabel('Length of review') # 그래프 x 축 라벨
6 plt.ylabel('Number of review') # 그래프 y 축 라벨
```



- 그래프 해석)
- 리뷰의 문자 길이가 대부분 6000 이하이고 대부분 2000 이하에 분포되어 있음을 할 수 있음
- 일부 데이터의 경우 10000 이상의 값을 가지고 있어 이상치로 보인다.

## 텍스트 분류

### 데이터 분석 및 전처리

#### 리뷰 문자 길이에 대한 기술 통계

```
1 print('리뷰 길이 최대 값: {}'.format(np.max(train_length)))
2 print('리뷰 길이 최소 값: {}'.format(np.min(train_length)))
3 print('리뷰 길이 평균 값: {:.2f}'.format(np.mean(train_length)))
4 print('리뷰 길이 표준편차: {:.2f}'.format(np.std(train_length)))
5 print('리뷰 길이 중간 값: {}'.format(np.median(train_length)))
6 # 사분위의 대한 경우는 0~100 스케일로 되어있음
7 print('리뷰 길이 제 1 사분위: {}'.format(np.percentile(train_length, 25)))
8 print('리뷰 길이 제 3 사분위: {}'.format(np.percentile(train_length, 75)))
```

리뷰 길이 최대 값: 13710  
리뷰 길이 최소 값: 54  
리뷰 길이 평균 값: 1329.71  
리뷰 길이 표준편차: 1005.22  
리뷰 길이 중간 값: 983.0  
리뷰 길이 제 1 사분위: 705.0  
리뷰 길이 제 3 사분위: 1619.0

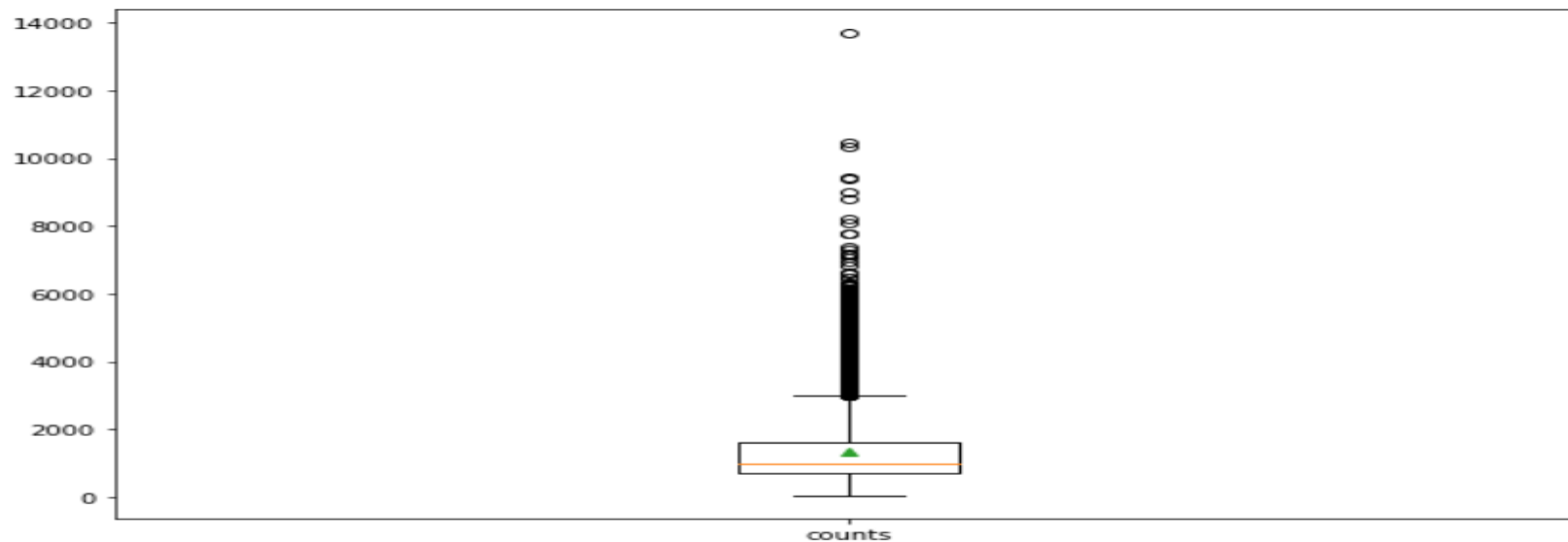
- 해석)
- 리뷰의 길이 평균이 1300정도, 최대값이 13000

## 텍스트 분류

### 데이터 분석 및 전처리

#### 리뷰 문자 길이에 대한 박스 플롯

```
1 plt.figure(figsize=(10, 7))
2 # 박스플롯 생성
3 # 첫번째 파라미터: 여러 분포에 대한 데이터 리스트를 입력
4 # labels: 입력한 데이터에 대한 라벨
5 # showmeans: 평균값을 마크함
6
7 plt.boxplot(train_length, labels=['counts'], showmeans=True)
```



- 그래프 해석)
- 리뷰 길이가 대부분 2000 이하, 평균이 1500 이하, 길이가 4000 이상인 이상치도 많이 있음

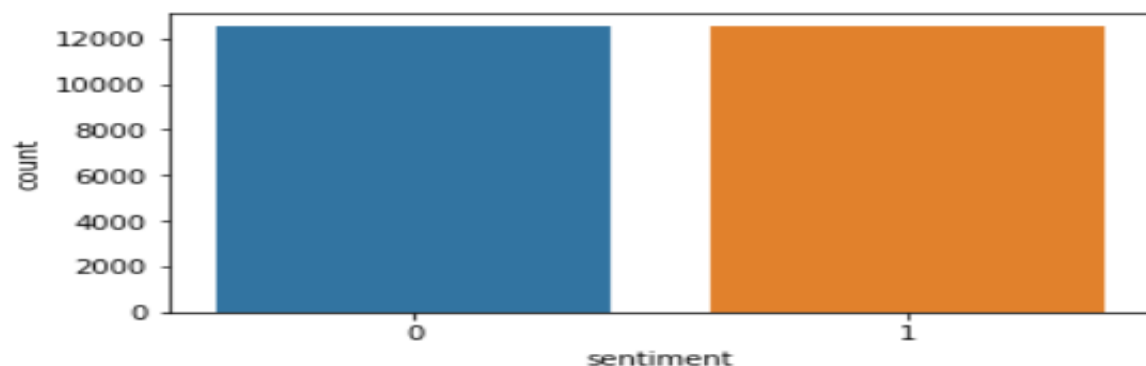


## 텍스트 분류

- 데이터 분석 및 전처리

### 라벨의 분포 - 긍정, 부정 데이터의 분포

```
1 fig, axe = plt.subplots(ncols=1)
2 fig.set_size_inches(6, 3)
3 sns.countplot(train_data['sentiment'])
```



- 그래프 해석)
- 라벨의 분포가 거의 동일

### 각 라벨에 대한 정확한 개수 확인

```
1 print("긍정 리뷰 개수: {}".format(train_data['sentiment'].value_counts()[1]))
2 print("부정 리뷰 개수: {}".format(train_data['sentiment'].value_counts()[0]))
```

긍정 리뷰 개수: 12500

부정 리뷰 개수: 12500

## 텍스트 분류

### 데이터 분석 및 전처리

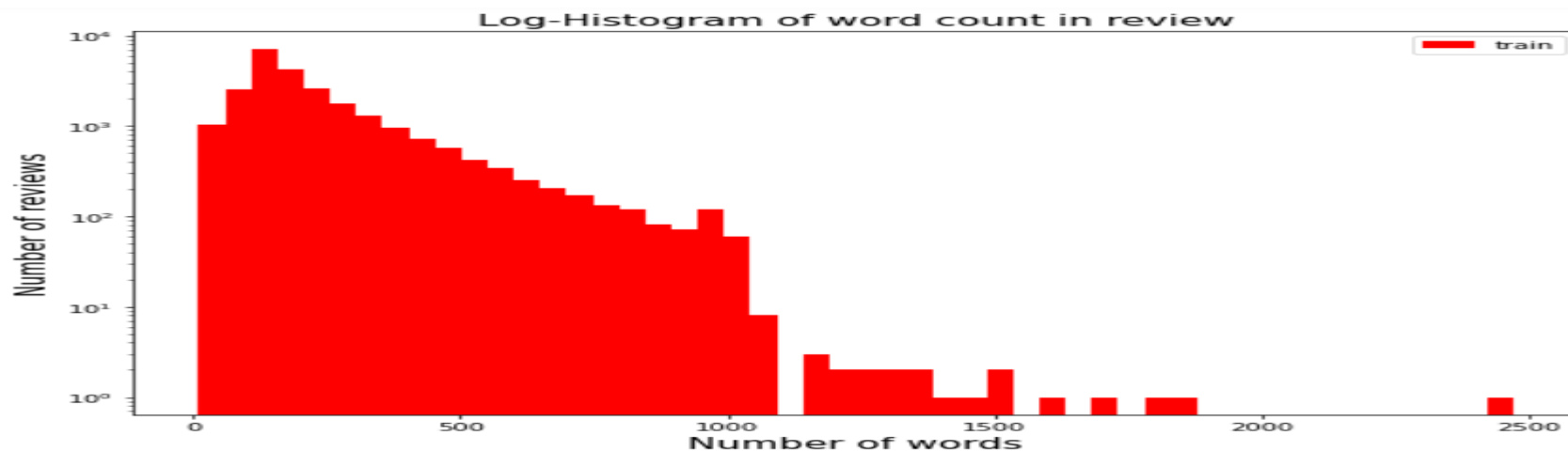
#### 각 리뷰의 단어 개수 분포 확인

- 띄어쓰기 기준

```
1 train_word_counts = train_data['review'].apply(lambda x: len(x.split(' ')))
```

- 리뷰 단어 개수 히스트그램

```
1 plt.figure(figsize=(10, 7))
2 plt.hist(train_word_counts, bins=50, facecolor='r', label='train')
3 plt.title('Log-Histogram of word count in review', fontsize=15)
4 plt.yscale('log', nonposy='clip')
5 plt.legend()
6 plt.xlabel('Number of words', fontsize=15)
7 plt.ylabel('Number of reviews', fontsize=15)
```



- 그래프 해석)
- 대부분의 리뷰의 단어 수가 100개 미만, 대부분 200개 정도의 단어를 가지고 있음을 확인

## 텍스트 분류

### 데이터 분석 및 전처리

#### 리뷰 단어 개수 기술 통계

```
1 print('리뷰 단어 개수 최대 값: {}'.format(np.max(train_word_counts)))
2 print('리뷰 단어 개수 최소 값: {}'.format(np.min(train_word_counts)))
3 print('리뷰 단어 개수 평균 값: {:.2f}'.format(np.mean(train_word_counts)))
4 print('리뷰 단어 개수 표준편차: {:.2f}'.format(np.std(train_word_counts)))
5 print('리뷰 단어 개수 중간 값: {}'.format(np.median(train_word_counts)))
6 # 사분위의 대한 경우는 0~100 스케일로 되어있음
7 print('리뷰 단어 개수 제 1 사분위: {}'.format(np.percentile(train_word_counts, 25)))
8 print('리뷰 단어 개수 제 3 사분위: {}'.format(np.percentile(train_word_counts, 75)))
```

리뷰 단어 개수 최대 값: 2470  
리뷰 단어 개수 최소 값: 10  
리뷰 단어 개수 평균 값: 233.79  
리뷰 단어 개수 표준편차: 173.74  
리뷰 단어 개수 중간 값: 174.0  
리뷰 단어 개수 제 1 사분위: 127.0  
리뷰 단어 개수 제 3 사분위: 284.0

- 해석)
- 리뷰 단어 개수 평균이 233개, 최대값이 2,470개의 단어
- 3사분위값이 284개로 리뷰의 75%가 300개 이하의 단어로 구성되어 있음을 확인



## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 특수문자 및 대, 소문자 비율

```
1 qmarks = np.mean(train_data['review'].apply(lambda x: '?' in x)) # 물음표가 구두점으로 쓰임
2 fullstop = np.mean(train_data['review'].apply(lambda x: '.' in x)) # 마침표
3 capital_first = np.mean(train_data['review'].apply(lambda x: x[0].isupper())) # 첫번째 대문자
4 capitals = np.mean(train_data['review'].apply(lambda x: max([y.isupper() for y in x]))) # 대문자가 몇개
5 numbers = np.mean(train_data['review'].apply(lambda x: max([y.isdigit() for y in x]))) # 숫자가 몇개
6
7 print('물음표가있는 질문: {:.2f}%'.format(qmarks * 100))
8 print('마침표가 있는 질문: {:.2f}%'.format(fullstop * 100))
9 print('첫 글자가 대문자 인 질문: {:.2f}%'.format(capital_first * 100))
10 print('대문자가있는 질문: {:.2f}%'.format(capitals * 100))
11 print('숫자가있는 질문: {:.2f}%'.format(numbers * 100))
```

물음표가있는 질문: 29.55%

마침표가 있는 질문: 99.69%

첫 글자가 대문자 인 질문: 0.00%

대문자가있는 질문: 99.59%

숫자가있는 질문: 56.66%

- 해석)
- 대부분 마침표를 포함하고 있고, 대문자도 대부분 사용하고 있음
- 전처리 과정에서 대문자의 경우 모두 소문자로 바꾸고 특수문자의 경우 제거한다.

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 2) 데이터 전처리

데이터 분석과정을 바탕으로 데이터를 모델에 적용시키기 위해 전처리 과정을 진행

- 전처리에 필요한 라이브러리 불러오기

```
1 import re
2 import json
3 import pandas as pd
4 import numpy as np
5 from bs4 import BeautifulSoup
6 from nltk.corpus import stopwords
7 from tensorflow.python.keras.preprocessing.sequence import pad_sequences
8 from tensorflow.python.keras.preprocessing.text import Tokenizer
```

- 데이터 불러오기 : pandas
- 데이터 정제 : re, BeautifulSoup
- 불용어 제거 : NLTK의 stopwords 모듈
- 토큰나이징 : tensorflow의 pad\_sequences, Tokenizer
- 정제된 데이터 저장 : numpy

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 학습 데이터 불러오기

```
1 DATA_IN_PATH = './data_in/'
2
3 train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
4 print(train_data['review'][0])
```

"With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary, watched The Wiz and watched Moonwalker again. Maybe i just want to get a certain insight into this guy who i think i remember going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's press and also the obvious message of drugs are bad m'kay.<br /><br />Visually impressive but of course this is all son so unless you remotely like MJ in anyway then you are going to hate this and find it boring. Some may call MJ's contribution 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

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### HTML 태그와 특수문자 제거

- BeautifulSoup을 이용한 HTML 태그 제거
- re.sub를 이용한 특수문자 제거

```
1 review = train_data['review'][0] # 리뷰 중 하나를 가져온다.  
2 review_text = BeautifulSoup(review, "html.parser").get_text() # HTML 태그 제거  
3 review_text = re.sub("[^a-zA-Z]", " ", review_text) # 영어 문자를 제외한 나머지는 모두 공백으로 바꾼다.
```

```
1 print(review_text)
```

With all this stuff going down at the moment with MJ i ve started listening to his music watching the odd do  
watched The Wiz and watched Moonwalker again Maybe i just want to get a certain insight into this guy who i t  
n the eighties just to maybe make up my mind whether he is guilty or innocent Moonwalker is part biography p  
remember going to see at the cinema when it was originally released Some of it has subtle messages about MJ s  
ss and also the obvious message of drugs are bad m kay Visually impressive but of course this is all about Mic  
ou remotely like MJ in anyway then you are going to hate this and find it boring Some may call MJ an egotist  
king of this movie BUT MJ and most of his fans would say that he made it for the fans which if true is really

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 불용어 삭제

- NLTK의 불용어 사전 이용
- NLTK의 불용어 사전은 소문자 단어로 구성되어 있으므로,
- 불용어를 제거하기 위해서는 먼저 모든 문자를 소문자로 바꾼 후 불용어 제거해야 한다.

```
1 stop_words = set(stopwords.words('english')) # 영어 불용어들의 set을 만든다.
2
3 review_text = review_text.lower() # 소문자 변환 후
4 words = review_text.split() # 띄어쓰기 기준으로 단어를 나눠서 단어 리스트로 만든다.
5 print('불용어 제거 전 단어 수: ', len(words))
6 words = [w for w in words if not w in stop_words] # 불용어 제거한 리스트를 만든다
7 print('불용어 제거 후 단어 수: ', len(words))
```

불용어 제거 전 단어 수: 437

불용어 제거 후 단어 수: 219

```
1 print(words)
```

['stuff', 'going', 'moment', 'mj', 'started', 'listening', 'music', 'watching', 'odd', 'documentary',  
onwalker', 'maybe', 'want', 'get', 'certain', 'insight', 'guy', 'thought', 'really', 'cool', 'eightie  
ther', 'guilty', 'innocent', 'moonwalker', 'part', 'biography', 'part', 'feature', 'film', 'remember'

### 텍스트 분류

#### ■ 데이터 분석 및 전처리

모델에 적용하기 위해서 단어 리스트 --> 문자열로 합쳐야 한다.

- 파이썬의 join() 함수 이용

```
1 clean_review = ' '.join(words) # 단어 리스트들을 다시 하나의 문자열로 합친다.  
2 print(clean_review)
```

stuff going moment mj started listening music watching odd documentary watched wiz watched moonwalk  
uy thought really cool eighties maybe make mind whether guilty innocent moonwalker part biography p  
cinema originally released subtle messages mj feeling towards press also obvious message drugs bad  
ael jackson unless remotely like mj anyway going hate find boring may call mj egotist consenting m  
ans true really nice actual feature film bit finally starts minutes excluding smooth criminal sequ  
ic powerful drug lord wants mj dead bad beyond mj overheard plans nah joe pesci character ranted w  
c dunno maybe hates mj music lots cool things like mj turning car robot whole speed demon sequence  
ame filming kiddy bad sequence usually directors hate working one kid let alone whole bunch perform  
movie people like mj one level another think people stay away try give wholesome message ironically  
l jackson truly one talented people ever grace planet guilty well attention gave subject hmmm well  
doors know fact either extremely nice stupid guy one sickest liars hope latter

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 전처리 과정 함수화

```
1 def preprocessing(review, remove_stopwords = False):
2     # 불용어 제거는 옵션으로 선택 가능하다.
3
4     # 1. HTML 태그 제거
5     review_text = BeautifulSoup(review, "html.parser").get_text()
6
7     # 2. 영어가 아닌 특수문자들을 공백(" ")으로 바꾸기
8     review_text = re.sub("[^a-zA-Z]", " ", review_text)
9
10    # 3. 대문자들을 소문자로 바꾸고 공백단위로 텍스트를 나뉘어서 리스트로 만든다.
11    words = review_text.lower().split()
12
13    if remove_stopwords:
14        # 4. 불용어 제거
15
16        # 영어에 관련된 불용어 불러오기
17        stops = set(stopwords.words("english"))
18        # 불용어가 아닌 단어들로 이루어진 새로운 리스트 생성
19        words = [w for w in words if not w in stops]
20        # 5. 단어 리스트를 공백을 넣어서 하나의 글로 합친다.
21        clean_review = ' '.join(words)
22
23    else: # 불용어 제거하지 않을 때
24        clean_review = ' '.join(words)
25
26    return clean_review
```

### 텍스트 분류

- 데이터 분석 및 전처리

#### 전체 데이터(25000건)에 전처리 함수 수행

```
1 clean_train_reviews = []
2 for review in train_data['review']:
3     clean_train_reviews.append(preprocessing(review, remove_stopwords = True))
4
5 # 전처리한 데이터 출력
6 clean_train_reviews[0]
```

'stuff going moment mj started listening music watching odd documentary watched wiz watched moonwalke  
guy thought really cool eighties maybe make mind whether guilty innocent moonwalker part biography pa  
e cinema originally released subtle messages mj feeling towards press also obvious message drugs bad  
chael jackson unless remotely like mj anyway going hate find boring may call mj egotist consenting ma  
fans true really nice actual feature film bit finally starts minutes excluding smooth criminal sequen  
hic powerful drug lord wants mj dead bad beyond mj overheard plans nah joe pesci character ranted wan  
tc dunno maybe hates mj music lots cool things like mj turning car robot whole speed demon sequence a  
came filming kiddy bad sequence usually directors hate working one kid let alone whole bunch performi  
e movie people like mj one level another think people stay away try give wholesome message ironically  
ael jackson truly one talented people ever grace planet guilty well attention gave subject hmmm well  
ed doors know fact either extremely nice stupid guy one sickest liars hope latter'

```
1 print(len(clean_train_reviews))
```

25000



## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 3) 각 단어를 인덱스로 벡터화

- 모델에 따라 단어들의 인덱스로 구성된 벡터 형태가 필요하기도 하고
- 모델에 따라 벡터가 아닌 텍스트로 구성되어야 하는 경우도 있으므로
- 지금까지 전처리한 데이터는 판다스의 데이터프레임으로 일단 저장해 둔다.

#### (1) 전처리한 텍스트 데이터를 데이터프레임으로 저장

```
1 # 전처리 과정이 끝난 전체 데이터 -> 데이터프레임으로 저장
2 clean_train_df = pd.DataFrame({'review': clean_train_reviews, 'sentiment': train_data['sentiment']})
```

```
1 clean_train_df.head()
```

	review	sentiment
0	stuff going moment mj started listening music ...	1
1	classic war worlds timothy hines entertaining ...	1
2	film starts manager nicholas bell giving welco...	0
3	must assumed praised film greatest filmed oper...	0
4	superbly trashy wondrously unpretentious explo...	1

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### (2) 각 단어의 인덱스로 벡터화

- 각 단어의 인덱스로 벡터화: 텐서플로의 전처리 모듈 Tokenizer 사용
  - Tokenizer 모듈 생성 -> 정제된 데이터에 적용 -> 인덱스로 구성된 벡터로 변환

```
1 from tensorflow.python.keras.preprocessing.sequence import pad_sequences
2 from tensorflow.python.keras.preprocessing.text import Tokenizer
3
4 tokenizer = Tokenizer()
5 tokenizer.fit_on_texts(clean_train_reviews)
6 text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
```

```
1 # 인덱스로 구성된 벡터 첫번째 값 : 각 단어의 인덱스로 바뀜
2 print(text_sequences[0])
```

```
[404, 70, 419, 8815, 506, 2456, 115, 54, 873, 516, 178, 18686, 178, 11242, 165, 78, 14, 662, 2457,
0, 581, 2333, 1194, 11242, 71, 4826, 71, 635, 2, 253, 70, 11, 302, 1663, 486, 1144, 3265, 8815, 41
5, 4424, 1851, 998, 146, 342, 1442, 743, 2424, 4, 8815, 418, 70, 637, 69, 237, 94, 541, 8815, 26057
20, 323, 167, 10, 207, 633, 635, 2, 116, 291, 382, 121, 15535, 3315, 1501, 574, 734, 10013, 923, 1
1, 15, 576, 8815, 22224, 2274, 13426, 734, 10013, 27, 28606, 340, 16, 41, 18687, 1500, 388, 11243,
4, 8815, 1430, 380, 2163, 114, 1919, 2503, 574, 17, 60, 100, 4875, 5100, 260, 1268, 26057, 15, 574,
46, 114, 615, 3266, 1160, 684, 48, 1175, 224, 1, 16, 4, 8815, 3, 507, 62, 25, 16, 640, 133, 231, 95
4, 1, 128, 342, 1442, 247, 3, 865, 16, 42, 1487, 997, 2333, 12, 549, 386, 717, 6920, 12, 41, 16, 15
8, 207, 254, 117, 3, 18688, 18689, 316, 1356]
```

## 텍스트 분류

### ■ 데이터 분석 및 전처리

```
1 # 각 인덱스가 어떤 단어를 의미하는 지 확인: 단어사전 필요
2 word_vocab = tokenizer.word_index
3 word_vocab["<PAD>"] = 0 # word_index에 <PAD>값이 정의되어 있지 않으므로 <PAD>의 인덱스 값을 0으로 입력
4 print(word_vocab)
```

```
{'movie': 1, 'film': 2, 'one': 3, 'like': 4, 'good': 5, 'time': 6, 'even': 7, 'would': 8, 'story': 9, 'real': 10, 'l': 12, 'much': 13, 'get': 14, 'bad': 15, 'people': 16, 'also': 17, 'first': 18, 'great': 19, 'made': 20, 'would': 23, 'movies': 24, 'think': 25, 'characters': 26, 'character': 27, 'watch': 28, 'two': 29, 'films': 30, 'life': 33, 'plot': 34, 'acting': 35, 'never': 36, 'love': 37, 'little': 38, 'best': 39, 'show': 40, 'know': 43, 'better': 44, 'end': 45, 'still': 46, 'say': 47, 'scene': 48, 'scenes': 49, 'go': 50, 'something': 51, 'watching': 54, 'though': 55, 'thing': 56, 'old': 57, 'years': 58, 'actors': 59, 'director': 60, 'work': 61, '3': 63, 'nothing': 64, 'funny': 65, 'actually': 66, 'makes': 67, 'look': 68, 'find': 69, 'going': 70, 'part': 71, 'world': 74, 'cast': 75, 'us': 76, 'quite': 77, 'want': 78, 'things': 79, 'pretty': 80, 'young': 81, 'seems': 82, 'ror': 84, 'got': 85, 'however': 86, 'fact': 87, 'take': 88, 'big': 89, 'enough': 90, 'long': 91, 'thought': 94, 'give': 95, 'original': 96, 'action': 97, 'right': 98, 'without': 99, 'must': 100, 'comedy': 101, 'always': 102, 'point': 104, 'gets': 105, 'family': 106, 'come': 107, 'role': 108, 'saw': 109, 'almost': 110, 'interesting': 111, 'ne': 113, 'whole': 114, 'music': 115, 'bit': 116, 'guy': 117, 'script': 118, 'far': 119, 'making': 120, 'min': 121, 'anything': 123, 'last': 124, 'might': 125, 'since': 126, 'performance': 127, 'girl': 128, 'probably': 129, '1': 130, 'tv': 132, 'away': 133, 'yet': 134, 'day': 135, 'rather': 136, 'worst': 137, 'fun': 138, 'sure': 139, 'has': 140, 'played': 142, 'found': 143, 'although': 144, 'especially': 145, 'course': 146, 'believe': 147, 'screen': 148, 'g': 150, 'trying': 151, 'set': 152, 'goes': 153, 'book': 154, 'looks': 155, 'place': 156, 'actor': 157, 'diff': 159, 'year': 160, 'money': 161, 'ending': 162, 'dvd': 163, 'let': 164, 'maybe': 165, 'someone': 166, 'true': 168, 'n': 169, 'everything': 170, 'shows': 171, 'three': 172, 'worth': 173, 'job': 174, 'main': 175, 'together': 176, 'd': 178, 'american': 179, 'everyone': 180, 'plays': 181, 'john': 182, 'effects': 183, 'later': 184, 'audience': 185, 'akes': 187, 'instead': 188, 'house': 189, 'beautiful': 190, 'seem': 191, 'night': 192, 'high': 193, 'version': 194}
```

```
1 # 전체 데이터에서 사용된 단어의 총 개수
2 print("전체 단어 개수: ", len(word_vocab))
```

전체 단어 개수: 74066

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 단어 사전, 전체 단어 개수 딕셔너리에 저장

- 추후 모델에서 사용

```
1 data_configs = {}  
2  
3 data_configs['vocab'] = word_vocab  
4 data_configs['vocab_size'] = len(word_vocab)
```

#### 4) 패딩 과정 - 데이터의 길이 맞추기

- 각 데이터의 길이가 서로 다른데 이 길이를 하나로 통일해야 모델에 바로 적용 가능하기 때문에
- 특정 길이를 최대 길이로 정하고 더 긴 데이터의 경우는 뒷부분을 자르고
- 더 짧은 데이터의 경우에는 0 값으로 패딩하는 작업 진행
- 텐서플로우의 전처리 모듈 이용

```
1 MAX_SEQUENCE_LENGTH = 174 # 문장 최대 길이  
2  
3 train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')  
4  
5 print('Shape of train data: ', train_inputs.shape)
```

Shape of train data: (25000, 174)

- 최대 길이 174 : 단어 갯의 통계에서 중간값
  - 25,000개의 데이터가 174라는 길이를 동일하게 가지게 되었음을 확인
- 텐서플로우의 pad\_sequences() 함수 사용
  - pad\_sequences(패딩을 적용할 데이터, 최대길이값, 0값을 데이터 앞에 넣을지 뒤에 넣을지 설정)

### 텍스트 분류

#### ■ 데이터 분석 및 전처리

##### 5) 학습 시 라벨(정답)을 나타내는 값을 넘파이 배열로 저장

- 전처리한 데이터를 저장할 때 넘파이 형태로 저장하기 때문에 정답도 넘파이 형태로 저장

```
1 train_labels = np.array(train_data['sentiment'])
2 print('Shape of label tensor:', train_labels.shape)
```

Shape of label tensor: (25000,)

```
1 train_labels[:10]
```

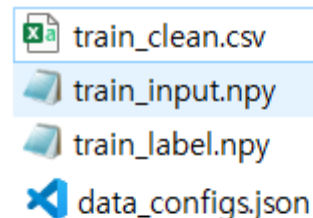
array([1, 1, 0, 0, 1, 1, 0, 0, 0, 1], dtype=int64)

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 6) 전처리 과정이 끝난 데이터 저장

- 정제된 텍스트 데이터 : csv 파일로 저장, 'train\_clean.csv'
- 벡터화된 데이터 : 넘파이 파일로 저장, 'train\_input.npy'
- 정답 라벨 : 넘파이 파일로 저장, 'train\_label.npy'
- 데이터 정보(단어사전, 전체 단어 개수): 딕셔너리 형태이므로 JSON 파일로 저장, 'data\_configs.json'



```
1 TRAIN_INPUT_DATA = 'train_input.npy'
2 TRAIN_LABEL_DATA = 'train_label.npy'
3 TRAIN_CLEAN_DATA = 'train_clean.csv'
4 DATA_CONFIGS = 'data_configs.json'
5
6 import os
7
8 DATA_IN_PATH = './data_in/'
9
10 # 저장하는 디렉토리가 존재하지 않으면 생성
11 if not os.path.exists(DATA_IN_PATH):
12     os.makedirs(DATA_IN_PATH)
```

```
1 # 전처리 된 데이터를 넘파이 형태로 저장
2 np.save(open(DATA_IN_PATH + TRAIN_INPUT_DATA, 'wb'), train_inputs)
3 np.save(open(DATA_IN_PATH + TRAIN_LABEL_DATA, 'wb'), train_labels)
4
5 # 정제된 텍스트를 csv 형태로 저장
6 clean_train_df.to_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA, index = False)
7
8 # 데이터 사전을 json 형태로 저장
9 json.dump(data_configs, open(DATA_IN_PATH + DATA_CONFIGS, 'w'), ensure_ascii=False)
```

## 텍스트 분류

### ■ 데이터 분석 및 전처리

#### 7) 테스트 데이터도 학습 데이터와 동일한 과정 진행

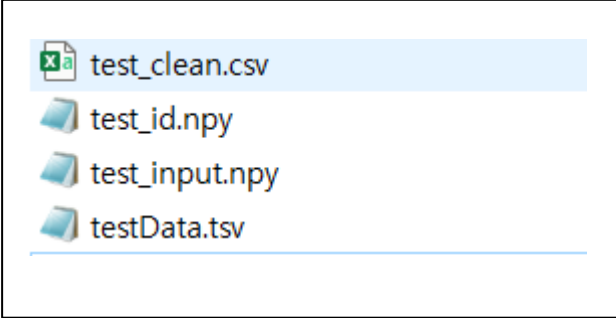
- 테스트 데이터는 라벨(정답)이 없으므로 따로 저장하지 않아도 된다.
  - 단어사전과 단어 개수에 대한 정보도 학습 데이터의 것을 사용해야 하므로 저장하지 않아도 된다.
    - 학습 데이터와 동일한 인덱스 정보를 사용해야만 함
  - 테스트 데이터는 각 리뷰에 대한 'id'값은 저장해야 한다.
- 
- 정제된 텍스트 데이터(id와 리뷰만 저장) : csv 파일로 저장, 'test\_clean.csv'
  - 벡터화된 데이터 : 넘파이 파일로 저장, 'test\_input.npy'
  - 테스트 데이터는 각 리뷰에 대한 'id'값 저장, 'test\_id.npy'

```
1 test_data = pd.read_csv(DATA_IN_PATH + "testData.tsv", header=0, delimiter="\t", quoting=3)
2
3 clean_test_reviews = []
4
5 for review in test_data['review']:
6     clean_test_reviews.append(preprocessing(review, remove_stopwords = True))
7
8 clean_test_df = pd.DataFrame({'review': clean_test_reviews, 'id': test_data['id']})
9 test_id = np.array(test_data['id'])
10
11 text_sequences = tokenizer.texts_to_sequences(clean_test_reviews)
12 test_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
```

## 텍스트 분류

### ■ 데이터 분석 및 전처리

```
1 TEST_INPUT_DATA = 'test_input.npy'|
2 TEST_CLEAN_DATA = 'test_clean.csv'
3 TEST_ID_DATA = 'test_id.npy'
4
5 np.save(open(DATA_IN_PATH + TEST_INPUT_DATA, 'wb'), test_inputs)
6 np.save(open(DATA_IN_PATH + TEST_ID_DATA, 'wb'), test_id)
7 clean_test_df.to_csv(DATA_IN_PATH + TEST_CLEAN_DATA, index = False)
```



test\_clean.csv

test\_id.npy

test\_input.npy

testData.tsv



## 텍스트 분류

### ■ 분류 모델링

- 머신러닝 모델 – 선형회귀모델, 랜덤포레스트모델
- 딥러닝 모델 – 합성곱 신경망(CNN), 순환 신경망(RNN)

### ■ 회귀 모델

#### • 선형 회귀 모델(Linear Regression)

- 종속변수(Y)와 한 개 이상의 독립변수(X)와의 선형 상관 관계를 모델링하는 회귀 분석 기법

1) 단순 선형 회귀:  $y = wx + b$   $w$ : 계수(가중치),  $b$ : 절편(편향)

2) 다중 선형 회귀:  $y = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n + b$

#### • 선형 회귀의 비용 함수

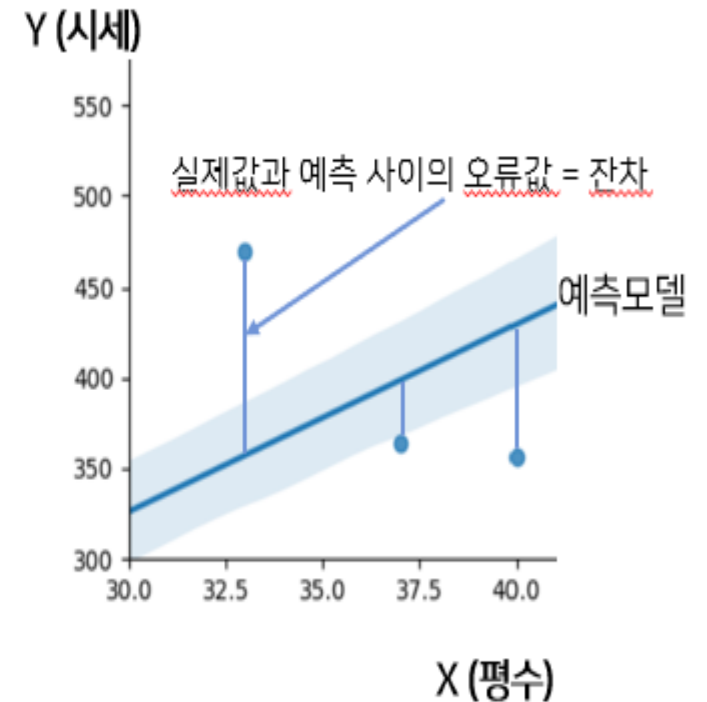
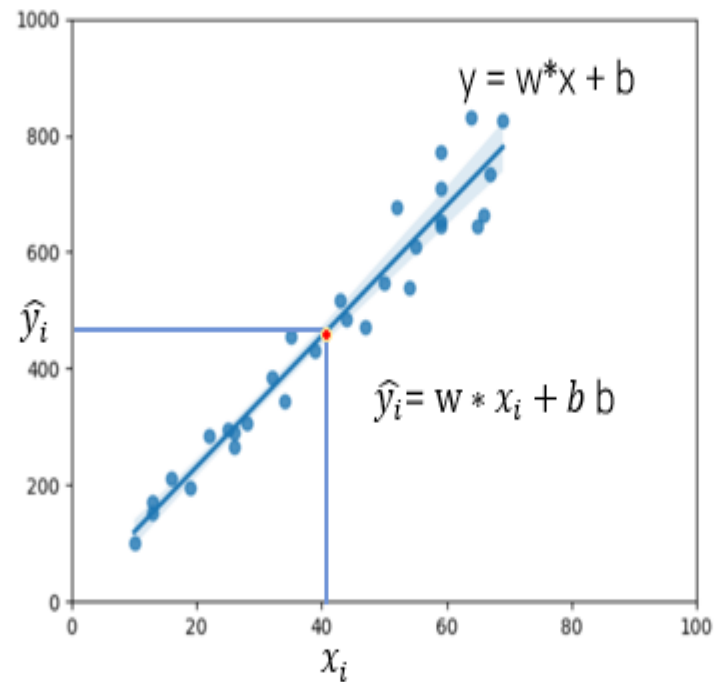
$$\text{Cost}_{\text{tr}} = \sum_i (y_i - \hat{y}_i)^2$$
$$\hat{y}_i = b + wx_i$$

- 실제 참값과 회귀 모델이 출력한 예측값 사이의 잔차의 제곱의 합을 최소화하는  $w$ (계수)를 구하는 것이 목적  
-> Least Square, 최소 제곱법

## 텍스트 분류

- 선형 회귀 모델
  - 주택 가격 예측: 주택 시세가 평수로만 결정된다고 가정

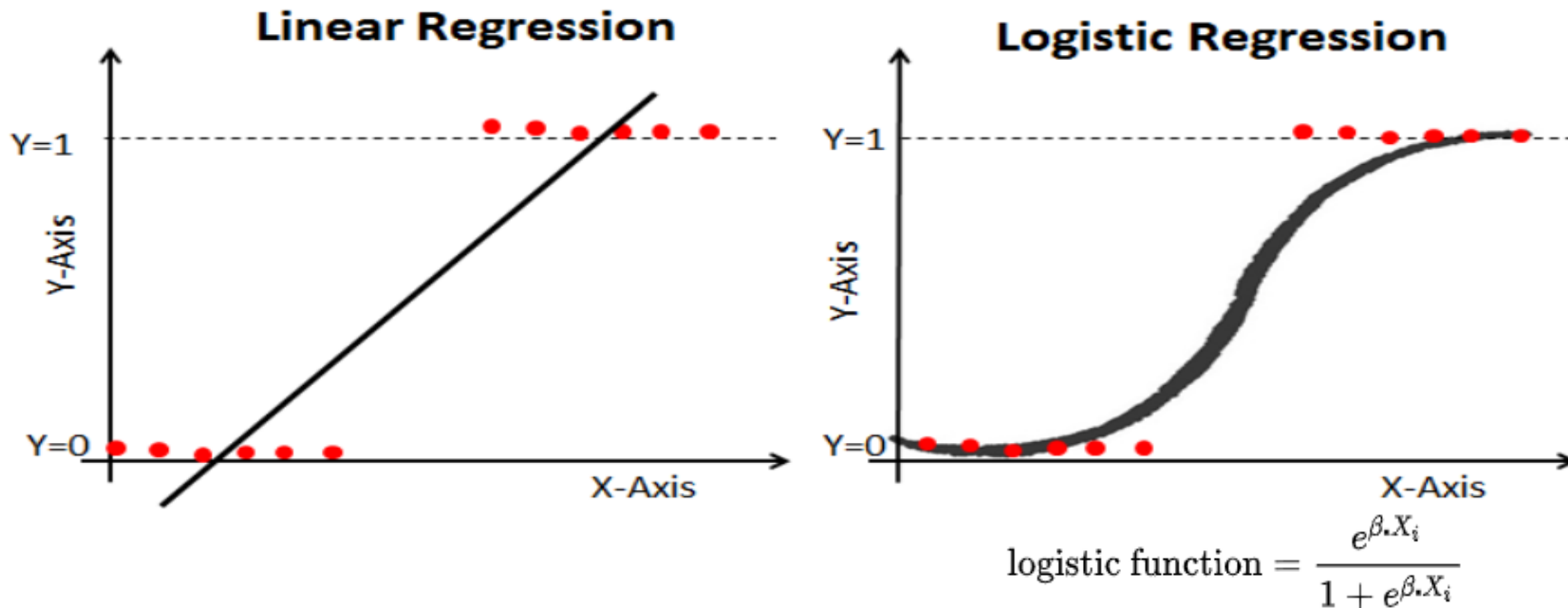
	Xi(평수)	Yi(시세)
0	10	102
1	22	284
2	32	384
3	64	832
4	50	547
<u>NewData</u>	<u>Xi</u>	<u>Yi</u>



## 텍스트 분류

### 로지스틱 회귀 모델

- 선형 모델의 결과값에 로지스틱 함수를 적용하여 0 ~ 1 사이의 값을 갖게 해서 확률로 표현한 모델
- 이렇게 나온 결과를 통해 1에 가까우면 정답이 1이라고 예측하고, 0에 가까울 경우 0으로 예측
- 로지스틱 회귀 모델에 가지고 텍스트 분류 진행
- 선형회귀 분석과 유사하지만 해당 데이터 결과가 특정 분류(Classification)로 나눔
- 로지스틱 회귀에는 종속변수가 이항적 문제에 사용



### 단어 임베딩 방법

- Word2Vec을 통한 단어 임베딩 벡터로 만드는 방법
- tf-idf를 통해 임베딩 벡터로 만드는 방법

## 텍스트 분류

### ▪ TF-IDF를 활용한 로지스틱 회귀 모델 구현

- TF-IDF를 활용해 문장 벡터를 생성
- 사이킷런의 TfidfVectorizer 사용
- TfidfVectorizer를 사용하기 위해서는 입력값이 반드시 텍스트로 이루어진 데이터 형태이어야 함
- csv 형태로 저장해 두었던 학습 데이터와 테스트 데이터 사용

### 라이브러리 불러오기

```
1 import os
2 import pandas as pd
3 import numpy as np
4
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.model_selection import train_test_split
7 from sklearn.linear_model import LogisticRegression
```

### 데이터 불러오기 - 텍스트 형태

```
1 DATA_IN_PATH = './data_in/'
2 DATA_OUT_PATH = './data_out/'
3 TRAIN_CLEAN_DATA = 'train_clean.csv'
4
5 RANDOM_SEED = 42
6 TEST_SPLIT = 0.2
```

## 텍스트 분류

### ▪ TF-IDF를 활용한 로지스틱 회귀 모델 구현

```
1 train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA )
```

```
1 reviews = list(train_data['review'])  
2 sentiments = list(train_data['sentiment'])
```

```
1 reviews[:1]
```

```
['stuff going moment mj started listening music watching odd documentary watched wi  
guy thought really cool eighties maybe make mind whether guilty innocent moonwalker  
e cinema originally released subtle messages mj feeling towards press also obvious  
chael jackson unless remotely like mj anyway going hate find boring may call mj egc  
fans true really nice actual feature film bit finally starts minutes excluding smoc  
hic powerful drug lord wants mj dead bad beyond mj overheard plans nah joe pesci ch  
tc dunno maybe hates mj music lots cool things like mj turning car robot whole spec  
came filming kiddy bad sequence usually directors hate working one kid let alone wh  
e movie people like mj one level another think people stay away try give wholesome  
ael jackson truly one talented people ever grace planet guilty well attention gave  
ed doors know fact either extremely nice stupid guy one sickest liars hope latter']
```

```
1 sentiments[:1]
```

```
[1]
```

## 텍스트 분류

### ▪ TF-IDF를 활용한 로지스틱 회귀 모델 구현

#### TF-IDF 벡터화

- `min_df` : 설정한 값보다 특정 토큰의 `df` 값이 적게 나오면 벡터화 과정에서 제거한다는 의미
  - `analyzer` : 분석하기 위한 기준, 'char':문자 하나를 단위로 지정, 'word':단어 하나를 단위로 지정
  - `sublinear_tf` : 문서의 단어 빈도수(`term frequency`)에 대한 스무딩(`smoothing`) 여부를 설정하는 값
  - `gram_range` : 빈도의 기본 단위를 어느 범위의 `n-gram`으로 설정할 것인지를 지정하는 인자
  - `max_features` : 각 벡터의 최대 길이, 특징의 길이를 설정
- 
- `TfidfVectorizer` 객체를 생성 --> `fit_transform()` 함수를 사용해 전체 문장에 대한 특징 벡터 데이터 `X`를 생성
  - `y` - 라벨(정답) 데이터

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="char", sublinear_tf=True, ngram_range=(1,3), max_features=5000)
4
5 X = vectorizer.fit_transform(reviews)
6 y = np.array(sentiments)
```

```
1 X
```

```
<25000x5000 sparse matrix of type '<class 'numpy.float64'>'
  with 17862871 stored elements in Compressed Sparse Row format>
```

```
1 features = vectorizer.get_feature_names()
2 features[:10]
```

```
[' ', ' a', ' aa', ' ab', ' ac', ' ad', ' ae', ' af', ' ag', ' ah']
```

## 텍스트 분류

### ▪ TF-IDF를 활용한 로지스틱 회귀 모델 구현

#### 학습 데이터와 테스트 데이터로 분리

- 사이킷런의 `train_test_split` 함수 이용
- 학습데이터:테스트데이터 = 80:20

```
1 from sklearn.model_selection import train_test_split
2 # RANDOM_SEED = 42
3 # TEST_SPLIT = 0.2
4 X_train, X_test, y_train, y_test = train_test_split(X, y, #
5                                                    test_size=TEST_SPLIT, random_state=RANDOM_SEED)
```

#### 로지스틱 회귀모델 생성

- `class_weight='balanced'`: 각 레벨에 대해 균형있게 학습할 수 있게 설정

```
1 from sklearn.linear_model import LogisticRegression
2
3 lgs = LogisticRegression(class_weight='balanced')
```

### 텍스트 분류

- TF-IDF를 활용한 로지스틱 회귀 모델 구현

#### 로지스틱 회귀모델 학습: fit()

```
1 lgs.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,  
                    fit_intercept=True, intercept_scaling=1, l1_ratio=None,  
                    max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',  
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,  
                    warm_start=False)
```

#### 모델 예측 및 모델 평가

- 테스트 데이터로 모델 예측 및 평가

```
1 predicted = lgs.predict(X_test)
```

```
1 print("Accuracy: %f" % lgs.score(X_test, y_test))
```

Accuracy: 0.859800

- 해석)
- 정확도가 85% 정도 나옴



## 텍스트 분류

### ▪ TF-IDF를 활용한 로지스틱 회귀 모델 구현

생성된 로지스틱 회귀모델을 이용하여 평가 데이터 결과를 예측

- 평가 데이터 파일 불러오기: 텍스트 형태 파일

```
1 TEST_CLEAN_DATA = 'test_clean.csv'
2
3 test_data = pd.read_csv(DATA_IN_PATH + TEST_CLEAN_DATA)
```

- 평가 데이터도 학습 데이터와 동일한 TF-IDF 벡터화 진행
- 주의) fit을 호출하지 않고 transform만 호출
  - fit은 학습 데이터에 맞게 설정, 그 설정에 맞게 평가 데이터도 변환해야 한다.

```
1 testDataVecs = vectorizer.transform(test_data['review'])
```

- 생성된 로지스틱회귀모델을 이용하여 예측

```
1 test_predicted = lgs.predict(testDataVecs)
2 print(test_predicted)
```

```
[1 0 1 ... 0 1 0]
```


- 예측한 결과값을 데이터프레임형태로 만들어 csv 파일로 저장
- 각 데이터의 고유값 id와 결과값으로 구성
- 캐글에 csv 파일 제출

## 텍스트 분류

### TF-IDF를 활용한 로지스틱 회귀 모델 구현


- 예측한 결과값을 데이터프레임형태로 만들어 csv 파일로 저장
- 각 데이터의 고유값 id와 결과값으로 구성
- 캐글에 csv 파일 제출

```
1 if not os.path.exists(DATA_OUT_PATH):  
2     os.makedirs(DATA_OUT_PATH)  
3  
4 answer_dataset = pd.DataFrame({'id': test_data['id'], 'sentiment': test_predicted})  
5 answer_dataset.to_csv(DATA_OUT_PATH + 'lgs_tfidf_answer.csv', index=False, quoting=3)
```

 Getting Started Prediction Competition

### Bag of Words Meets Bags of Popcorn

Use Google's Word2Vec for movie reviews

 Kaggle · 577 teams · 6 years ago

[Overview](#) [Data](#) [Code](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#) [My Submissions](#) [Late Submission](#)

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
lgs_tfidf_answer.csv	just now	1 seconds	1 seconds	0.85384

Complete

[Jump to your position on the leaderboard](#)

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### 라이브러리 불러오기

```
1 import os
2 import re
3 import pandas as pd
4 import numpy as np
5
6 from bs4 import BeautifulSoup
7 from nltk.corpus import stopwords
8 from nltk.tokenize import word_tokenize
```

#### 학습 데이터 불러오기: 전처리가 된 텍스트 파일

- word2vec은 단어로 표현된 리스트를 입력값으로 넣어야 하기 때문에
- 전처리된 텍스트 csv 파일을 불러온 후 각 단어들의 리스트로 나눠야 함

```
1 DATA_IN_PATH = './data_in/'
2 TRAIN_CLEAN_DATA = 'train_clean.csv'
3
4 RANDOM_SEED = 42
5 TEST_SPLIT = 0.2
```

```
1 train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
```

```
1 reviews = list(train_data['review'])
2 sentiments = list(train_data['sentiment'])
```

### 텍스트 분류

#### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

**word2vec을 사용하기 위해 입력값을 단어로 구분된 리스트로 만들어야 한다.**

- 읽어온 학습 데이터 전체 리뷰를 단어 리스트로 바꾼다.
- 각 리뷰를 split 함수를 이용해서 띄어쓰기 기준으로 구분한 후 리스트에 하나씩 추가해서 단어리스트를 만든다.

```
1 sentences = []  
2 for review in reviews:  
3     sentences.append(review.split())
```

```
1 sentences[:1]
```

```
'supplying',  
'drugs',  
'etc',  
'dunno',  
'maybe',
```

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### Word2Vec 벡터화

##### Word2Vec 모델의 하이퍼파라미터 설정

- num\_features : 각 단어에 대한 임베딩된 벡터의 차원 설정
- min\_word\_count : 모델에 의미 있는 단어를 가지고 학습하기 위해 적은 빈도수의 단어들은 학습하지 않는다.
- num\_workers : 모델 학습 시 학습을 위한 프로세스의 개수를 지정
- context : word2vec을 수행하기 위한 컨텍스트 윈도우 크기를 지정
- downsampling : word2vec 학습을 수행할 때 빠른 학습을 위해 정답 단어 라벨에 대한 다운 샘플링 비율을 지정
  - 보통 0.001이 좋은 성능을 낸다고 하난.

```
1 num_features = 300      # 워드 벡터 특징값 수
2 min_word_count = 40     # 단어에 대한 최소 빈도수
3 num_workers = 4         # 프로세스 개수
4 context = 10            # 컨텍스트 윈도우 크기
5 downsampling = 1e-3     # 다운 샘플링 비율
```

##### 설정된 하이퍼파라미터를 가지고 word2vec 학습

- gensim 라이브러리 사용, gensim.models에 있는 word2vec 모듈 사용

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

설정된 하이퍼파라미터를 가지고 word2vec 학습

- gensim 라이브러리 사용, gensim.models에 있는 word2vec 모듈 사용

```
1 #!pip install gensim
```

- word2vec 학습하는 과정에서 진행 상황을 확인하기 위해 logging 사용
- level(로그 수준)을 INFO에 맞추면 word2vec 학습과정에서 로그메세지를 양식에 맞게 INFO 수준으로 보여줌

```
1 import logging
2 logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
3                     level=logging.INFO)
```

### Word2Vec 학습

- word2vec 모듈에 있는 Word2Vec 객체를 생성하여 실행
- 학습하고 생성된 객체는 model 변수에 할당
- 입력데이터와 하이퍼파라미터를 순서대로 입력하여 학습 실행

```
1 from gensim.models import word2vec
2
3 model = word2vec.Word2Vec(sentences, workers=num_workers,
4                           size=num_features, min_count = min_word_count,
5                           window = context, sample = downsampling)
```

```
2021-03-04 11:20:15,495 : INFO : collecting all words and their counts
```

### 텍스트 분류

- Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### word2vec으로 학습된 모델 저장

- 모델 저장 `model.save(모델이름)`
- 저장한 모델은 `Word2Vec.load()`로 불러올 수 있다.

```
1 model_name = "./data_out/300features_40minwords_10context"  
2 model.save(model_name)
```

```
2021-03-04 11:23:48,353 : INFO : saving Word2Vec object under ./data_out/300features_40minwords  
2021-03-04 11:23:48,354 : INFO : not storing attribute vectors_norm  
2021-03-04 11:23:48,354 : INFO : not storing attribute cum_table  
2021-03-04 11:23:48,780 : INFO : saved ./data_out/300features_40minwords_10context
```

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### 로지스틱 회귀 모델 적용하기 위해 전처리

- 하나의 리뷰를 같은 형태의 입력값으로 만들어야 한다.
- word2vec 모델에서 각 단어가 벡터로 표현되어 있고, 리뷰마다 단어의 개수가 모두 다르기 때문에 입력값을 하나의 형태로 만들어야 한다.
- 가장 단순한 방법 : 문장에 있는 모든 단어의 벡터값에 대해 평균을 내서 리뷰 하나당 하나의 벡터를 만드는 것

#### 하나의 리뷰에 대해 전체 단어의 평균값을 계산하는 함수

- 문장에 특징값을 만들 수 있는 함수
- `get_features`(단어의 모음인 하나의 리뷰, word2vec모델, word2vec으로 임베딩할 때 정했던 벡터의 차원수)

```
1 def get_features(words, model, num_features):
2
3     # np.zeros 함수를 사용해 모두 0의 값을 가지는 벡터 생성
4     feature_vector = np.zeros((num_features), dtype=np.float32)
5
6     # 문장이 단어가 해당 모델 단어 사전에 속하는지 확인하기 위해 set 객체로 생성
7     num_words = 0
8     index2word_set = set(model.wv.index2word)
9
10    # 리뷰를 구성하는 단어에 대해 임베딩된 벡터가 있는 단어 벡터의 합을 구한다.
11    for w in words:
12        if w in index2word_set:
13            num_words += 1
14            feature_vector = np.add(feature_vector, model[w])
15
16    # 사용한 단어의 전체 개수로 나눠서 평균 벡터의 값을 구한다.
17    feature_vector = np.divide(feature_vector, num_words)
18    return feature_vector
```



## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### 전체 리뷰에 대해 각 리뷰의 평균 벡터를 구하는 함수

- `get_dataset`(학습데이터의 전체 리뷰데이터, `word2vec`모델, `word2vec`으로 임베딩할때 정했던 벡터의 차

```
1 def get_dataset(reviews, model, num_features):
2     dataset = list()
3
4     for s in reviews:
5         dataset.append(get_features(s, model, num_features))
6
7     reviewFeatureVecs = np.stack(dataset)
8
9     return reviewFeatureVecs
```

#### 전체 리뷰에 대해 평균 벡터 구하기

```
1 test_data_vecs = get_dataset(sentences, model, num_features)
```

```
c:\users\jinsuk.kim\anaconda3\envs\mlp\lib\site-packages\ipykernel_launcher.py:14: Deprecat
` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
```

```
1 test_data_vecs[:1]
```

```
array([[ -1.00970184e-02,  -6.70368820e-02,   5.29728085e-02,
        -2.06597373e-02,   1.43107489e-01,   2.62437314e-01,
```

### 텍스트 분류

- Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### 학습 데이터와 테스트 데이터 분리

```
1 from sklearn.model_selection import train_test_split
2 import numpy as np
3
4 # RANDOM_SEED = 42
5 # TEST_SPLIT = 0.2
6
7 X = test_data_vecs
8 y = np.array(sentiments)
9
10 X_train, X_test, y_train, y_test = train_test_split(X, y, #
11                                                    test_size=TEST_SPLIT, random_state=RANDOM_SEED)
```

#### 로지스틱 회귀 모델 생성

```
1 from sklearn.linear_model import LogisticRegression
2
3 lgs = LogisticRegression(class_weight='balanced')
```

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### 모델 학습

```
1 lgs.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,  
                    fit_intercept=True, intercept_scaling=1, l1_ratio=None,  
                    max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',  
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,  
                    warm_start=False)
```

#### 모델 예측 및 평가

```
1 # 예측  
2 predicted = lgs.predict(X_test)  
3 predicted
```

```
array([0, 1, 0, ..., 0, 0, 0])
```

```
1 # 실제 정답  
2 y_train
```

```
array([0, 0, 0, ..., 1, 1, 0])
```

```
1 # 정확도  
2 print("Accuracy: %f" % lgs.score(X_test, y_test))
```

```
Accuracy: 0.862400
```

- 해석)
- 분류 정확도가 86% 정도
- TF-IDF는 85% 정도이므로 TF-IDF에 비해 Word2Vec이 성능이 다소 좋다고 볼 수 있음

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

#### 생성된 로지스틱 회귀모델을 이용하여 평가 데이터 결과를 예측

- 평가 데이터 파일 불러오기: 텍스트 형태 파일

```
1 TEST_CLEAN_DATA = 'test_clean.csv'
2
3 test_data = pd.read_csv(DATA_IN_PATH + TEST_CLEAN_DATA)
4
5 test_review = list(test_data['review'])
```

```
1 test_data.head(5)
```

	review	id
0	naturally film main themes mortality nostalgia...	"12311_10"
1	movie disaster within disaster film full great...	"8348_2"
2	movie kids saw tonight child loved one point k...	"5828_4"
3	afraid dark left impression several different ...	"7186_2"
4	accurate depiction small time mob life filmed ...	"12128_7"

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

- 평가 데이터도 학습 데이터와 동일하게 각 리뷰가 하나의 문자열로 이루어져 있으므로
- 평가 데이터도 각 단어의 리스트로 만들어야 한다.

```
1 test_sentences = list()
2 for review in test_review:
3     test_sentences.append(review.split())
```

- 평가 데이터도 word2vec으로 임베딩된 벡터값으로 만들어야 한다.
- 평가 데이터에는 새롭게 word2vec 모델을 만들면 안되고, 학습 데이터 모델에 학습시킨 모델을 이용하여
- 평가 데이터에 각 단어들을 벡터로 만들어 각 리뷰에 대한 특징값을 만든다.
- 그런 후 이전에 정의했던 함수를 동일하게 적용한다.

```
1 test_data_vecs = get_dataset(test_sentences, model, num_features)
```

- 생성된 로지스틱회귀모델을 이용하여 예측

```
1 test_predicted = lgs.predict(test_data_vecs)
2 print(test_predicted)
```

```
[1 0 1 ... 0 1 1]
```

## 텍스트 분류

### ▪ Word2Vec을 활용한 로지스틱 회귀 모델 구현

- 예측한 결과값을 데이터프레임형태로 만들어 csv 파일로 저장
- 각 데이터의 고유값 id와 결과값으로 구성
- 캐글에 csv 파일 제출한 후 정확도 확인

```
1 DATA_OUT_PATH = './data_out/'
2
3 if not os.path.exists(DATA_OUT_PATH):
4     os.makedirs(DATA_OUT_PATH)
5
6 ids = list(test_data['id'])
7 answer_dataset = pd.DataFrame({'id': ids, 'sentiment': test_predicted})
8 answer_dataset.to_csv(DATA_OUT_PATH + 'lgs_w2v_answer.csv', index=False, quoting=3)
```

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
lgs_w2v_answer.csv	just now	1 seconds	1 seconds	0.85840

Complete

[Jump to your position on the leaderboard](#) ▼

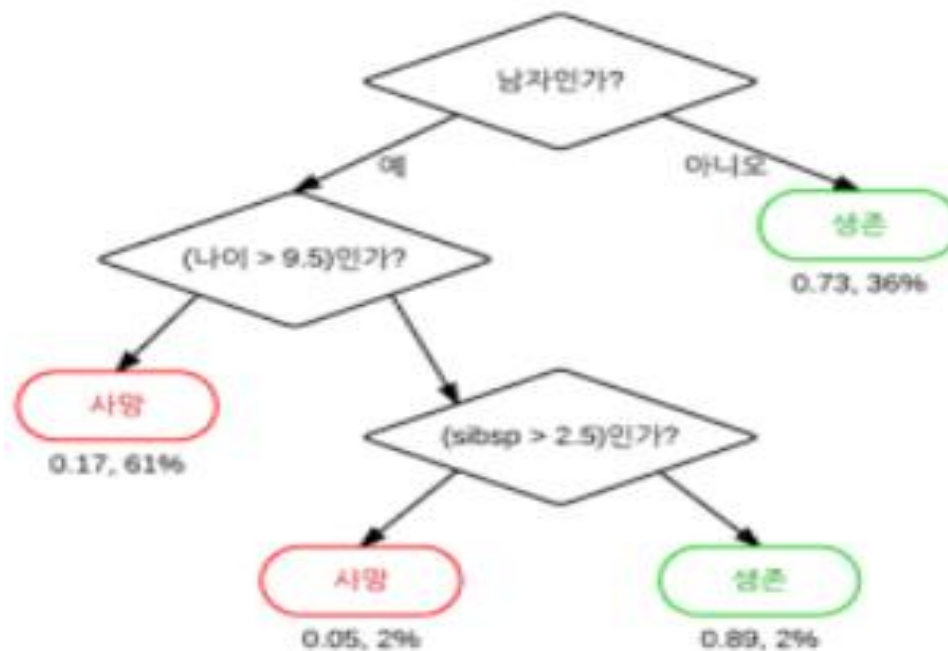
## 텍스트 분류

### ■ 랜덤포레스트 분류 모델

- 여러 개의 의사결정트리의 결과값을 평균낸 것으로 결과를 사용하는 분류 모델
- 분류 혹은 회귀에 사용 → 분류나무, 회귀나무

### ■ 의사결정트리(Decision Tree)

- 트리구조와 같은 형태로 이루어진 알고리즘
- 트리 구조 형태에서 각 노드는 하나의 질문이 된다. 질문에 따라 다음 노드가 달라지며, 몇 개의 질문이 끝난 후에 결과가 나오는 형태

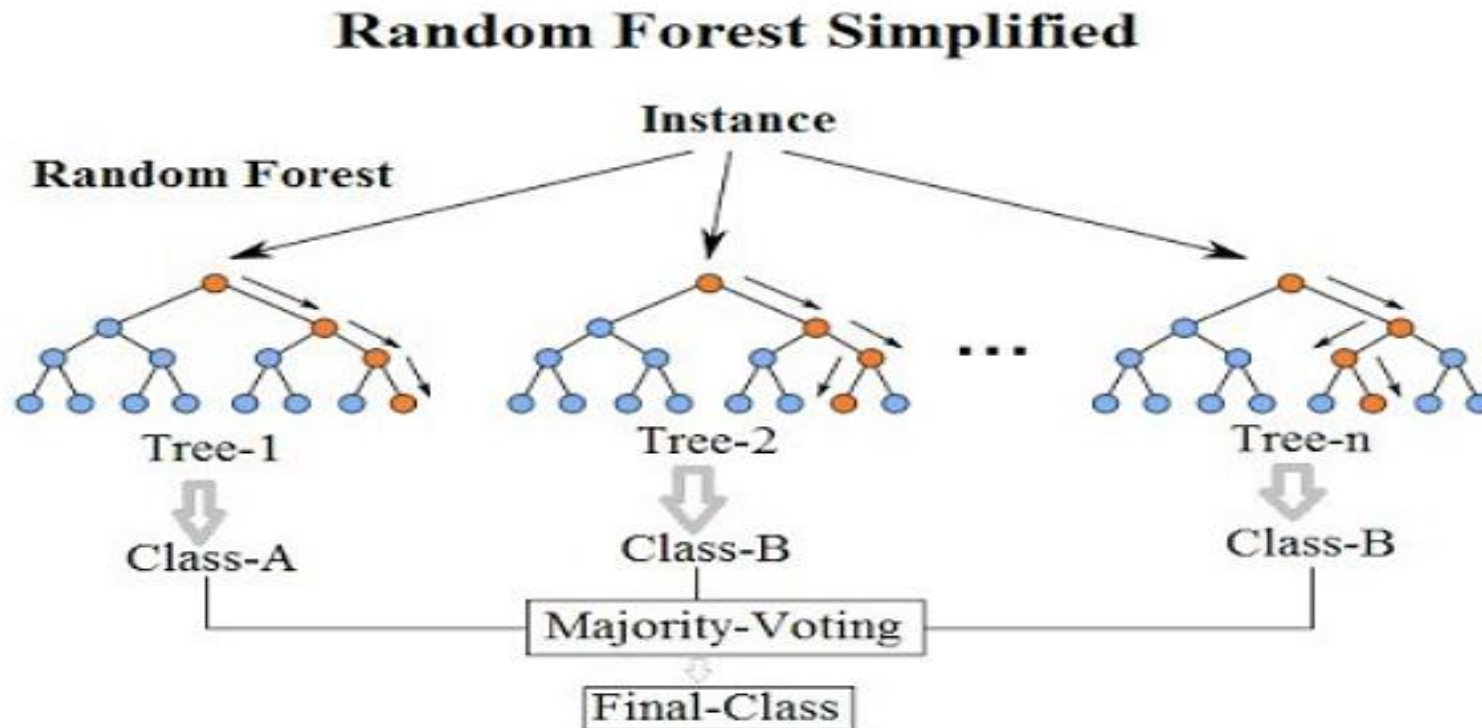


타이타닉 탑승객의 생존여부 의사결정트리

## 텍스트 분류

### ■ 랜덤포레스트 분류 모델

- 앙상블 알고리즘 중 비교적 빠른 수행 속도
- 다양한 영역에서 높은 정확도, 결정 트리 기반의 알고리즘
- 여러 개의 의사결정트리의 결과값을 평균낸 것으로 결과를 사용하는 분류 모델





## 텍스트 분류

### ▪ 랜덤포레스트 분류 모델

#### 라이브러리 불러오기

```
1 import pandas as pd
2 import numpy as np
3 import os
4 from sklearn.feature_extraction.text import CountVectorizer
```

#### 데이터 불러오기

- 전처리된 텍스트 파일 불러오기

```
1 DATA_IN_PATH = './data_in/'
2 DATA_OUT_PATH = './data_out/'
3 TRAIN_CLEAN_DATA = 'train_clean.csv'
4 TEST_SIZE = 0.2
5 RANDOM_SEED = 42
```

```
1 train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
```

```
1 reviews = list(train_data['review'])
2 y = np.array(train_data['sentiment'])
```

## 텍스트 분류

### 랜덤포레스트 분류 모델

#### CountVectorizer를 활용한 벡터화

- 사이킷런의 CountVectorizer를 이용하여 모델의 입력값을 생성
- CountVectorizer 객체를 불러온 후 리뷰 텍스트 데이터에 적용시켜 벡터화된 값을 변수에 할당
- 분석 단위를 하나의 단어로 지정: analyzer = "word", 각 벡터의 최대 길이 5000으로 지정

```
1 vectorizer = CountVectorizer(analyzer = "word", max_features = 5000)
2
3 train_data_features = vectorizer.fit_transform(reviews)
```

```
1 train_data_features
```

```
<25000x5000 sparse matrix of type '<class 'numpy.int64'>'
  with 1975048 stored elements in Compressed Sparse Row format>
```

- (25000, 5000)크기의 행렬로 구성
- 25000개의 리뷰 데이터가 각각 5000개의 특징값을 갖는 벡터로 표현되어 있음을 의미

```
1 type(train_data_features)
```

```
scipy.sparse.csr.csr_matrix
```

## 텍스트 분류

### ■ 랜덤포레스트 분류 모델

#### 학습 데이터와 테스트로 분리

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(train_data_features, y, #
4                                                    test_size=TEST_SIZE, random_state=RANDOM_SEED)
```

#### 랜덤포레스트 모델 생성

```
1 from sklearn.ensemble import RandomForestClassifier
2
3 # 랜덤 포레스트 분류기에 100개 의사 결정 트리를 사용한다.
4 forest = RandomForestClassifier(n_estimators = 100)
```

#### 모델 학습

```
1 # 단어 묶음을 벡터화한 데이터와 정답 데이터를 가지고 학습
2 forest.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                        criterion='gini', max_depth=None, max_features='auto',
                        max_leaf_nodes=None, max_samples=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=100,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
```

### 텍스트 분류

- 랜덤포레스트 분류 모델

#### 모델 예측과 평가

```
1 predict = forest.predict(X_test)
2 predict
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

```
1 y_test
```

```
array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

```
1 # 정확도
2 print("Accuracy: %f" % forest.score(X_test, y_test))
```

```
Accuracy: 0.843000
```

## 텍스트 분류

### ▪ 랜덤포레스트 분류 모델

#### 평가 데이터셋으로 성능 평가

##### 평가 데이터 불러오기

- 평가 데이터도 각 리뷰를 단어들의 리스트로 만든다.
- id와 결과값을 csv에 담기 위해 id 분리

```
1 TEST_CLEAN_DATA = 'test_clean.csv'
2
3 test_data = pd.read_csv(DATA_IN_PATH + TEST_CLEAN_DATA)
4
5 test_reviews = list(test_data['review'])
6 ids = list(test_data['id'])
```

- 모델을 적용하기 위해 학습 데이터에 적용한 것처럼 벡터화
- 학습 데이터에 적용시킨 vectorizer를 사용

```
1 test_data_features = vectorizer.transform(test_reviews)
```

## 텍스트 분류

### 랜덤포레스트 분류 모델

#### 데이터 제출하기

- 벡터화된 평가 데이터를 학습시킨 랜덤포레스트 모델에 적용하고
- id와 예측값을 판다스 데이터프레임 형식으로 만든 후
- csv 파일로 저장한다.
- 캐글에 csv 파일 제출한 후 정확도 확인

```
1  if not os.path.exists(DATA_OUT_PATH):
2      os.makedirs(DATA_OUT_PATH)
3
4  # 위에서 만든 랜덤 포레스트 분류기를 통해 예측값을 가져온다.
5  result = forest.predict(test_data_features)
6
7  # 판다스 데이터 프레임에 데이터를 구성해서 output에 넣는다.
8  output = pd.DataFrame( data={"id": ids, "sentiment": result} )
9
10 # csv파일로 만든다.
11 output.to_csv( DATA_OUT_PATH + "Bag_of_Words_model.csv", index=False, quoting=3 )
```

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Bag_of_Words_model.csv	just now	1 seconds	1 seconds	0.84256

Complete

[Jump to your position on the leaderboard](#) ▼

### 텍스트 분류 실습 - 20 뉴스그룹 분류

- 텍스트 분류 실습
  - 사이킷런의 예제 데이터 20 뉴스 그룹 데이터 세트를 이용하여 텍스트 분류 적용
- 텍스트 분류
  - 특정 문서의 분류를 학습 데이터를 통해 학습해 모델을 생성한 뒤 이 학습 모델을 이용하여 다른 문서의 분류를 예측하는 것
  - 로지스틱 회귀, 선형 서포트 벡터 머신, 나이브 베이즈 등의 모델 이용
- 로지스틱 회귀를 이용한 텍스트 분류 실습 프로세스
  - 1) 텍스트 정규화
  - 2) 피처 벡터화
  - 3) 머신러닝 알고리즘을 적용하여 분류를 학습/예측/평가

## 텍스트 분류 실습 - 20 뉴스그룹 분류

### 1. 데이터 로딩과 데이터 구성 확인

```
1 from sklearn.datasets import fetch_20newsgroups
2
3 news_data = fetch_20newsgroups(subset='all', random_state=156)
```

```
1 print(news_data.keys())
```

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

```
1 import pandas as pd
2
3 print('target 클래스의 값과 분포도 \n', pd.Series(news_data.target).value_counts().sort_index())
4 print('target 클래스의 이름들 \n', news_data.target_names)
5 len(news_data.target_names), pd.Series(news_data.target).shape
```

```
target 클래스의 값과 분포도
```

```
0    799
1    973
2    985
3    982
4    963
5    988
```

```
dtype: int64
```

```
target 클래스의 이름들
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'com  
sc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.c  
space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politi
```

```
(20, (18846,))
```



### 텍스트 분류 실습 - 20 뉴스그룹 분류

```
1 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...
```

## 텍스트 분류 실습 - 20 뉴스그룹 분류

### 2. 학습과 테스트용 데이터 생성

```
1 from sklearn.datasets import fetch_20newsgroups
2
3 # subset='train'으로 학습용(Train) 데이터만 추출
4 # remove=('headers', 'footers', 'quotes')로 내용만 추출
5 train_news= fetch_20newsgroups(subset='train',
6                                remove=('headers', 'footers', 'quotes'),
7                                random_state=156)
8 X_train = train_news.data
9 y_train = train_news.target
10 print(type(X_train))
11
12 # subset='test'으로 테스트(Test) 데이터만 추출
13 # remove=('headers', 'footers', 'quotes')로 내용만 추출
14 test_news= fetch_20newsgroups(subset='test',
15                               remove=('headers', 'footers', 'quotes'),
16                               random_state=156)
17 X_test = test_news.data
18 y_test = test_news.target
19 print('학습 데이터 크기 {0} , 테스트 데이터 크기 {1}'.format(len(train_news.data),
20                                                                len(test_news.data)))
```

<class 'list'>

학습 데이터 크기 11314 , 테스트 데이터 크기 7532

## 텍스트 분류 실습 - 20 뉴스그룹 분류

### 3. Count 피쳐 벡터화 변환과 머신러닝 모델 학습/예측/평가

- 주의: 학습 데이터에 대해 fit( )된 CountVectorizer를 이용해서 테스트 데이터를 피쳐 벡터화 해야함.
  - 테스트 데이터에서 다시 CountVectorizer의 fit\_transform()을 수행하거나 fit()을 수행 하면 안됨.
  - 이는 이렇게 테스트 데이터에서 fit()을 수행하게 되면 기존 학습된 모델에서 가지는 feature의 갯수가 달라지기 때문임.

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 # Count Vectorization으로 feature extraction 변환 수행.
4 cnt_vect = CountVectorizer()
5 cnt_vect.fit(X_train)
6 X_train_cnt_vect = cnt_vect.transform(X_train)
7
8 # 학습 데이터로 fit( )된 CountVectorizer를 이용하여 테스트 데이터를 feature extraction 변환 수행.
9 X_test_cnt_vect = cnt_vect.transform(X_test)
10
11 print('학습 데이터 Text의 CountVectorizer Shape:', X_train_cnt_vect.shape, X_test_cnt_vect.shape)
```

학습 데이터 Text의 CountVectorizer Shape: (11314, 101631) (7532, 101631)

### 4. LogisticRegression을 이용하여 학습/예측/평가 수행

```
1 from sklearn.linear_model import LogisticRegression
2 from sklearn.metrics import accuracy_score
3
4 # LogisticRegression을 이용하여 학습/예측/평가 수행.
5 lr_clf = LogisticRegression()
6 lr_clf.fit(X_train_cnt_vect, y_train)
7 pred = lr_clf.predict(X_test_cnt_vect)
8 print('CountVectorized Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test, pred)))
```

### 텍스트 분류 실습 - 20 뉴스그룹 분류

#### 5. TF-IDF 피쳐 변환과 머신러닝 학습/예측/평가

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 # TF-IDF Vectorization 적용하여 학습 데이터셋과 테스트 데이터 셋 변환.
4 tfidf_vect = TfidfVectorizer()
5 tfidf_vect.fit(X_train)
6 X_train_tfidf_vect = tfidf_vect.transform(X_train)
7 X_test_tfidf_vect = tfidf_vect.transform(X_test)
8
9 # LogisticRegression을 이용하여 학습/예측/평가 수행.
10 lr_clf = LogisticRegression()
11 lr_clf.fit(X_train_tfidf_vect , y_train)
12 pred = lr_clf.predict(X_test_tfidf_vect)
13 print('TF-IDF Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test , pred)))
14
```

TF-IDF Logistic Regression 의 예측 정확도는 0.674

### 텍스트 분류 실습 - 20 뉴스그룹 분류

#### 6. stop words 필터링을 추가하고 ngram을 기본(1,1)에서 (1,2)로 변경하여 피처 벡터화

```
1 # stop words 필터링을 추가하고 ngram을 기본(1,1)에서 (1,2)로 변경하여 Feature Vectorization 적용.
2 tfidf_vect = TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_df=300 )
3 tfidf_vect.fit(X_train)
4 X_train_tfidf_vect = tfidf_vect.transform(X_train)
5 X_test_tfidf_vect = tfidf_vect.transform(X_test)
6
7 lr_clf = LogisticRegression()
8 lr_clf.fit(X_train_tfidf_vect , y_train)
9 pred = lr_clf.predict(X_test_tfidf_vect)
10 print('TF-IDF Vectorized Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred)))
11
```

TF-IDF Vectorized Logistic Regression 의 예측 정확도는 0.690

---

# [ 토픽 모델링 ]

## 토픽 모델링

- 토픽 모델링(Topic Medeling)
  - 문서 집합에 숨어 있는 주제를 찾아내는 것
  - 머신러닝 기반의 토픽 모델링
    - 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출
  - LSA(Latent Semantic Analysis) 방식
  - LDA(Latent Dirichlet Allocation) 방식
- LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습
  - 네이버 뉴스 기사 크롤링 데이터 사용

- LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

### gensim으로 네이버 뉴스를 이용한 토픽 모델링

- 1. 토픽 모델링을 적용하기 위해 텍스트를 전처리
- 2. gensim을 사용한 토픽 모델링

#### 1. 토픽 모델링을 위한 라이브러리 불러오기

```
: 1 from tqdm import tqdm_notebook # progress bar
  2 from konlpy.tag import Okt #Okt 등 형태소 분석기 불러오기
  3 import numpy as np
  4 import string # 특수문자
  5 import re
  6 import warnings # 경고 알림 제거
  7 import pickle
  8 from gensim import corpora
  9 from gensim import models
 10 import matplotlib.pyplot as plt
 11 %matplotlib inline
 12
 13 warnings.filterwarnings("ignore", category=DeprecationWarning)
```



## ▪ LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

### 2. 텍스트 전처리 함수 만들기

```
1 def read_documents(input_file_name):
2     corpus = []
3
4     with open(input_file_name, 'rb') as f:
5         temp_corpus = pickle.load(f)
6
7     for page in temp_corpus:
8         corpus += page
9
10    return corpus
11
12 def text_cleaning(docs):
13     # 한국어를 제외한 글자를 제거하는 함수.
14     for doc in docs:
15         doc = re.sub("[^ㄱ-ㅎㅌ-ㅣ가-힣 ]", "", str(doc))
16
17     return docs
18
19 def define_stopwords(path):
20     SW = set()
21     # 불용어를 추가하는 방법 1.
22     for i in string.punctuation:
23         SW.add(i)
24     # 불용어를 추가하는 방법 2.
25     # stopwords-ko.txt에 직접 추가
26     with open(path) as f:
27         for word in f:
28             SW.add(word)
29     return SW
```

## ▪ LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

```
31 def text_tokenizing(corpus, tokenizer):
32     okt = Okt()
33     token_corpus = []
34
35     if tokenizer == "noun":
36         for n in tqdm_notebook(range(len(corpus)), desc="Preprocessing"):
37             token_text = okt.nouns(corpus[n])
38             token_text = [word for word in token_text if word not in SW and len(word) > 1]
39             token_corpus.append(token_text)
40
41     elif tokenized == "morph":
42         for n in tqdm_notebook(range(len(corpus)), desc="Preprocessing"):
43             token_text = okt.morphs(corpus[n])
44             token_text = [word for word in token_text if word not in SW and len(word) > 1]
45             token_corpus.append(token_text)
46
47     elif tokenizer == "word":
48         for n in tqdm_notebook(range(len(corpus)), desc="Preprocessing"):
49             token_text = corpus[n].split()
50             token_text = [word for word in token_text if word not in SW and len(word) > 1]
51             token_corpus.append(token_text)
52
53     return token_corpus
```

```
1 input_file_name = "data/naver_news_content.pk"
2 documents = read_documents(input_file_name)
3 SW = define_stopwords("data/stopwords-ko.txt")
4 cleaned_text = text_cleaning(documents)
5 tokenized_text = text_tokenizing(cleaned_text, tokenizer="noun") #tokenizer= "noun" or "word"
```

Preprocessing: 100%  10/10 [00:08<00:00, 1.22it/s]

## ▪ LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

```
1 print(tokenized_text[0])
```

['본문', '내용', '플레이어', '플레이어', '오류', '우회', '함수', '추가', '성큼', '몸철', '날씨', '밀집', '공간', '활동', '해외여행', '야외', '소규모', '인원', '골프', '급증', '때문', '한국', '골프', '의류', '매출', '기점', '일본', '추월', '추정', '국내외', '골프', '웨어', '브랜드', '추위', '시즌', '위해', '감각', '트렌디', '라인업', '무장', '소비자', '눈길', '골퍼', '격식', '골프', '웨어', '감각', '도', '과연', '한국', '레저', '산업', '연구소', '국내', '골프', '인구', '가운데', '올해', '비중', '렌디', '스포츠', '탈바꿈', '때문', '골프', '웨어', '변화', '바람', '거세', '감각', '개성', '골프', '통', '골프', '웨어', '모델', '사진', '제공', '코오롱', '인더스', '트리', '부문', '코오롱', '전개',

### 3. 토픽 모델링에 사용할 행렬 생성

#### 1) 문서-단어 행렬(DTM, document-term matrix) 생성

```
1 # 어휘(vocabulary) 학습
2 dictionary = corpora.Dictionary(tokenized_text)
3
4 # 문서-단어 행렬(document-term matrix) 생성
5 corpus = [dictionary.doc2bow(text) for text in tokenized_text]
```

```
1 print(dictionary)
```

Dictionary(914 unique tokens: ['가운데', '가지', '각인', '감각', '강국']...)

```
1 corpus[0][:5]
```

[(0, 1), (1, 1), (2, 1), (3, 5), (4, 1)]

## ▪ LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

### 2) TFIDF 문서-단어 행렬 생성

```
1 # TFIDF 문서-단어 행렬 생성
2 tfidf = models.TfidfModel(corpus)
3 corpus_tfidf = tfidf[corpus]
4 corpus_tfidf[0][:5]
```

```
[(0, 0.009126737974895004),
 (1, 0.017454788624084866),
 (2, 0.017454788624084866),
 (3, 0.08727394312042433),
 (4, 0.017454788624084866)]
```

### 4. LDA 모델 생성

```
1 model = models.ldamodel.LdaModel(corpus, num_topics=4, id2word=dictionary)
```

```
1 model.show_topic(0, 10)
```

```
[('골프', 0.023091981),
 ('웨어', 0.01513274),
 ('논문', 0.01212985),
 ('브랜드', 0.007971047),
 ('램지', 0.007547554),
 ('관련', 0.007149657),
 ('라이브', 0.007006109),
 ('코로나', 0.006440769),
 ('본문', 0.0059930324),
 ('내용', 0.0058897086)]
```

### ▪ LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

#### 5. 토픽 모델링을 추가하여 코드 완성하기

```
1  # 토픽 개수, 키워드 개수를 정해주는 변수를 추가.
2  NUM_TOPICS = 3 # 토픽의 수
3
4  NUM_TOPIC_WORDS = 30 # 키워드 수
5
6  def build_doc_term_mat(documents):
7      # 문서-단어 행렬 만들어주는 함수.
8      print("Building document-term matrix.")
9      dictionary = corpora.Dictionary(documents)
10     corpus = [dictionary.doc2bow(document) for document in documents]
11     return corpus, dictionary
12
13 def print_topic_words(model):
14     # 토픽 모델링 결과를 출력해 주는 함수.
15     print("\nPrinting topic words.\n")
16
17     for topic_id in range(model.num_topics):
18         topic_word_probs = model.show_topic(topic_id, NUM_TOPIC_WORDS)
19         print("Topic ID: {}".format(topic_id))
20
21         for topic_word, prob in topic_word_probs:
22             print("\t{}\t{}".format(topic_word, prob))
23     print("\n")
```

## ▪ LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

```

1 # 1) document-term matrix 생성
2 corpus, dictionary = build_doc_term_mat(tokenized_text)
3
4 # 2) LDA 모델 생성
5 model = models.ldamodel.LdaModel(corpus, num_topics=NUM_TOPICS, id2word=dictionary, alpha="auto", eta="auto")
6
7 # 3) 결과를 출력.
8 print_topic_words(model)

```

Building document-term matrix.

Printing topic words.

Topic ID: 0

골프	0.01274965237826109
코로나	0.009363619610667229
플레이어	0.0070604910142719746
내용	0.007013153750449419
서점	0.006636959500610828
벚꽃	0.006296992301940918
본문	0.005727563984692097
주사기	0.005516388453543186
웨어	0.005388683173805475
라이브	0.0052011688239872456
관련	0.004913897719234228
취소	0.004853131715208292
논문	0.004716092720627785
지역	0.004702571779489517
지난해	0.004388177767395973
백신	0.004209159407764673
브랜드	0.004187149461358786

Topic ID: 1

골프	0.026873502880334854
웨어	0.01712164096534252
주사기	0.01045741606503725
브랜드	0.007263353560119867
논문	0.0069460803642869
내용	0.006890041287988424
컬러	0.0059076217003166676
코로나	0.0058203102089464664
접종	0.0051562064327299595
라이브	0.005126995965838432
본문	0.005107234697788954
백신	0.005088075064122677
플레이어	0.0047659003175795
패턴	0.004599078558385372
뉴스	0.004551183432340622
램지	0.0044305408373475075
시장	0.004260058514773846
기존	0.0040842099115252495
바늘	0.0038976070936769247
지난	0.0038826600648462772
기능	0.003812120296061039
고객	0.00381070957519114

Topic ID: 2

주사기	0.01324341632425785
접종	0.009618542157113552
바늘	0.008401429280638695
본문	0.008258428424596786
백신	0.007837258279323578
코로나	0.007665525656193495
플레이어	0.0071406858041882515
내용	0.006957292556762695
일본	0.0066609689965844154
뉴스	0.006354865152388811
지역	0.0060176062397658825
조립	0.005748099181801081
데루모	0.005582146346569061
지난	0.005466986447572708
개발	0.005034672562032938
추가	0.004664421547204256
길이	0.0046592759899795055
관련	0.004524745047092438
후생노동성	0.00409969968586254

## ▪ LDA(Latent Dirichlet Allocation)를 이용한 토픽 모델링 실습

### 6. pyLDAvis를 통한 토픽 모델링 결과 시각화하기

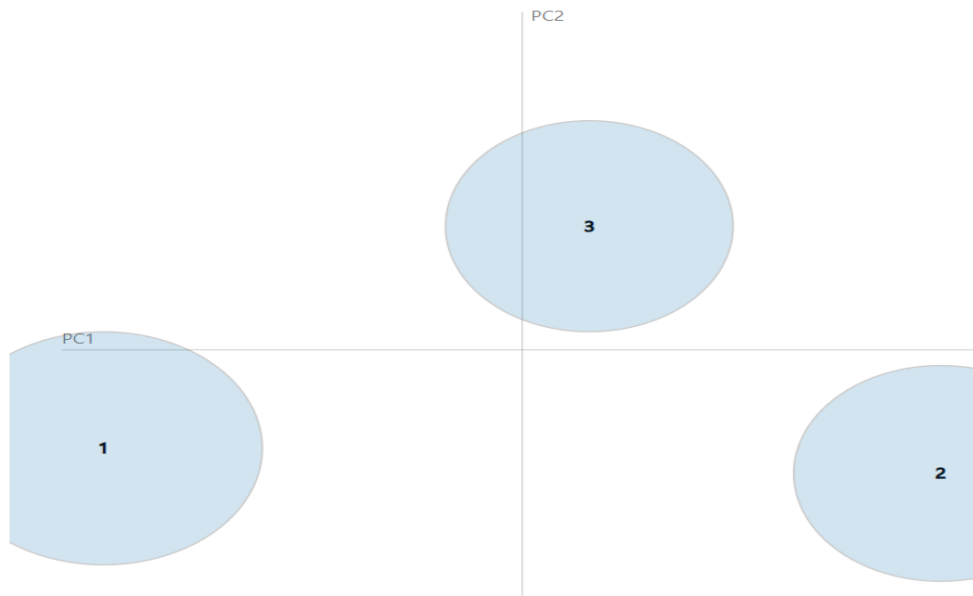
```

1 #!pip install pyLDAvis

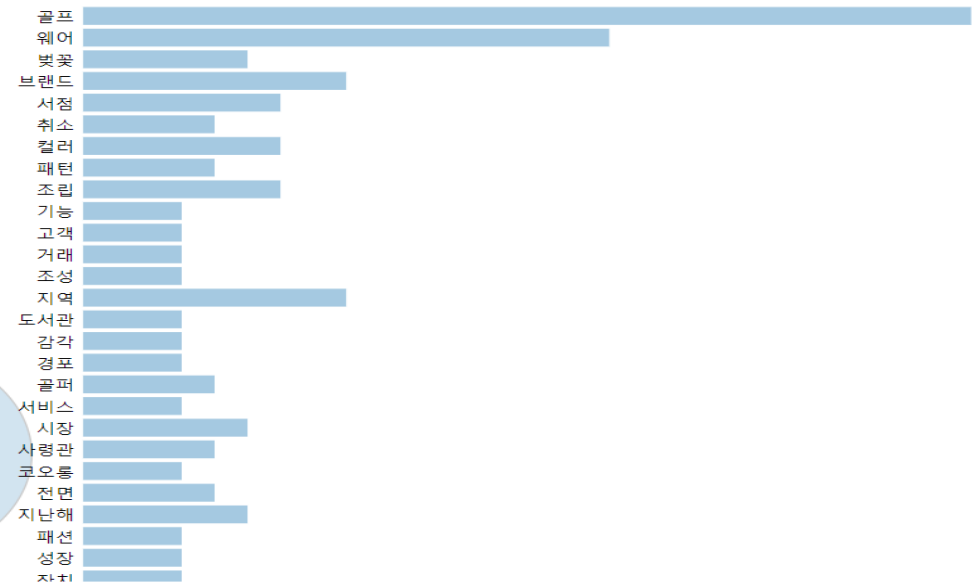
1 # pyLDAvis 불러오기
2 import pyLDAvis
3 import pyLDAvis.gensim
4
5 # pyLDAvis를 jupyter notebook에서 실행할 수 있게 활성화.
6 pyLDAvis.enable_notebook()
7
8 # pyLDAvis 실행.
9 data = pyLDAvis.gensim.prepare(model, corpus, dictionary)
10 data
    
```

Selected Topic:

Intertopic Distance Map (via multidimensional scaling)



Top-30 Most Salient Terms<sup>1</sup>



---

# [ 텍스트 유사도 ]



## 텍스트 유사도

### ■ 텍스트 유사도

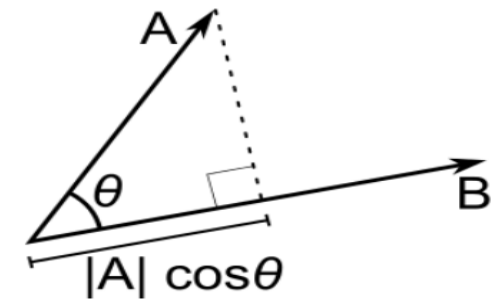
- 텍스트가 얼마나 유사한지를 표현하는 방식 중 하나
- 두 문장 간의 유사도를 계산하기 위해서는 문장 내에 존재하는 단어들을 수치화해야 함
- 텍스트 유사도 측정 방법: 통계를 이용하는 방법, 인공신경망을 이용하는 방법
- 텍스트를 벡터화 한 후 벡터화 된 각 문장 간의 유사도를 측정하는 방식
  - 자카드 유사도
  - 유클리디언 유사도
  - 맨하탄 유사도
  - 코사인 유사도

## 텍스트 유사도

### ■ 코사인 유사도(Cosine Similarity)

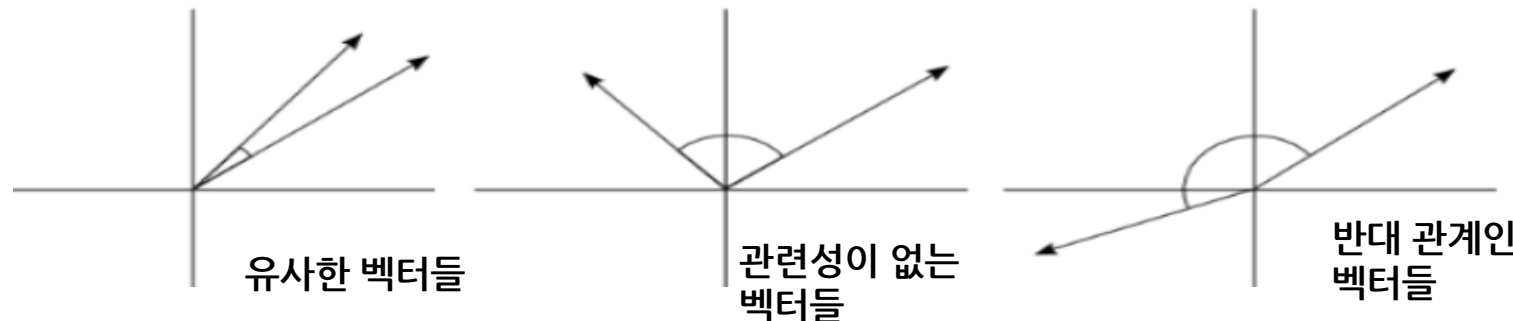
- 두 개의 벡터값에서 코사인 각도를 이용하여 유사도를 측정하는 방법
- 벡터의 크기가 중요하지 않을 때 그 거리를 측정하기 위해 사용
- 두 벡터 사이의 사잇각을 구해서 얼마나 유사한지 수치로 적용한 것
- 코사인 유사도 값은 -1에서 1사이의 값
- 두 벡터가 완전히 동일한 경우에는 1, 반대방향인 경우에는 -1, 두 벡터가 서로 직각을 이루면 0
- 두 벡터의 방향이 같아질 수록 유사, 1에 가까울수록 유사하다는 것을 의미
- 공간 벡터의 내적과 크기를 이용하여 코사인 각도를 계산

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$



### ■ 두 벡터의 사잇각

- 두 벡터의 사잇각에 따라서 벡터 간의 상호관계를 알 수 있음



$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

## 텍스트 유사도

### 2) 코사인 유사도(Cosine Similarity)

- A : 6월에 뉴턴은 선생님의 제안으로 트리니티에 입학했다.
- B : 6월에 뉴턴은 선생님의 제안으로 대학교에 입학했다.
- 단어 문서 행렬 표현

	6월	뉴턴	선생님	제안	트리니티	입학	대학
A	1	1	1	1	1	1	0
B	1	1	1	1	0	1	1

- A = [1,1,1,1,1,1,0]
- B = [1,1,1,1,0,1,1]

$$\begin{aligned} & \sum_{i=1}^n A_i \times B_i \\ &= (1*1) + (1*1) + (1*1) + (1*1) + (1*1) + (1*0) + (1*1) + (0*1) = 5 \\ & \sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2} \\ &= \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2} \times \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2} \\ &= \sqrt{6} \times \sqrt{6} = \sqrt{36} = 6 \end{aligned}$$

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{5}{6} = 0.83333$$

## 텍스트 유사도

- 두 문장(글) 간의 유사도를 측정하는 모델을 만드는 것

### 코사인 유사도 반환 함수 생성

```
1 import numpy as np
2
3 def cos_similarity(v1, v2):
4     dot_product = np.dot(v1, v2)
5     l2_norm = (np.sqrt(sum(np.square(v1)))) * np.sqrt(sum(np.square(v2)))
6     similarity = dot_product / l2_norm
7
8     return similarity
```

### TF-IDF 벡터화 후 코사인 유사도 비교

- 1) TF-IDF 벡터화된 행렬로 변환 후
- 2) doc\_list로 정의된 3개의 간단한 문서의 유사도 비교

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 doc_list = ['if you take the blue pill, the story ends' ,
4             'if you take the red pill, you stay in Wonderland',
5             'if you take the red pill, I show you how deep the rabbit hole goes']
6
7 tfidf_vect_simple = TfidfVectorizer()
8 feature_vect_simple = tfidf_vect_simple.fit_transform(doc_list)
9 print(feature_vect_simple.shape)
```

(3, 18)

## 텍스트 유사도

### 코사인 유사도 계산 과정

- 반환된 행렬이 희소 행렬이므로, `cos_similarity()` 함수의 인자인 `array`로 만들기 위해
- 밀집행렬로 변환한 뒤 다시 각각을 배열로 반환이 필요
- `feature_vect_dense[0]` : `doc_list` 첫번째 문서의 피처 벡터화
- `feature_vect_dense[1]` : `doc_list` 두번째 문서의 피처 벡터화
- 두 문서의 피처 벡터화 후 `np.array()`로 변환
- `cos_similarity()` 함수 사용하여 코사인 유사도 계산

```
1  # TfidfVectorizer로 transform()한 결과는 Sparse Matrix이므로 Dense Matrix로 변환.
2  feature_vect_dense = feature_vect_simple.todense()
3
4  #첫번째 문장과 두번째 문장의 feature vector 추출
5  vect1 = np.array(feature_vect_dense[0]).reshape(-1,)
6  vect2 = np.array(feature_vect_dense[1]).reshape(-1,)
7
8  #첫번째 문장과 두번째 문장의 feature vector로 두개 문장의 Cosine 유사도 추출
9  similarity_simple = cos_similarity(vect1, vect2 )
10 print('문장 1, 문장 2 Cosine 유사도: {0:.3f}'.format(similarity_simple))
11
```

문장 1, 문장 2 Cosine 유사도: 0.402

### 텍스트 유사도

```
: 1  #첫번째 문장과 세번째 문장의 feature vector로 두개 문장의 Cosine 유사도 추출
2  vect1 = np.array(feature_vect_dense[0]).reshape(-1,)
3  vect3 = np.array(feature_vect_dense[2]).reshape(-1,)
4  similarity_simple = cos_similarity(vect1, vect3 )
5  print('문장 1, 문장 3 Cosine 유사도: {0:.3f}'.format(similarity_simple))
6
7  #두번째 문장과 세번째 문장의 feature vector로 두개 문장의 Cosine 유사도 추출
8  vect2 = np.array(feature_vect_dense[1]).reshape(-1,)
9  vect3 = np.array(feature_vect_dense[2]).reshape(-1,)
10 similarity_simple = cos_similarity(vect2, vect3 )
11 print('문장 2, 문장 3 Cosine 유사도: {0:.3f}'.format(similarity_simple))
```

문장 1, 문장 3 Cosine 유사도: 0.404

문장 2, 문장 3 Cosine 유사도: 0.456

## 텍스트 유사도

### 2) 사이킷런의 `cosine_similarity()` 함수를 이용하여 비교

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 doc_list = ['if you take the blue pill, the story ends' ,
4             'if you take the red pill, you stay in Wonderland',
5             'if you take the red pill, I show you how deep the rabbit hole goes']
6
7 tfidf_vect_simple = TfidfVectorizer()
8 feature_vect_simple = tfidf_vect_simple.fit_transform(doc_list)
```

```
1 feature_vect_simple
```

```
<3x18 sparse matrix of type '<class 'numpy.float64'>'
  with 29 stored elements in Compressed Sparse Row format>
```

### `cosine_similarity` 함수

- 첫번째 파라미터 : 비교 기준이 되는 문서의 피처 행렬
- 두번째 파라미터 : 비교되는 문서의 피처 행렬
- 파라미터는 희소행렬, 밀집 행렬, 배열 모두 가능

```
1 from sklearn.metrics.pairwise import cosine_similarity
2
3 similarity_simple_pair = cosine_similarity(feature_vect_simple[0] , feature_vect_simple)
4 print(similarity_simple_pair)
```

```
[[1.          0.40207758  0.40425045]]
```

- 해석)
- 첫번째 유사도 1 : 비교 기준인 첫번째 문서 자신에 대한 유사도 측정
- 두번째 유사도 0.40207758 : 첫번째 문서와 두번째 문서의 유사도
- 세번째 유사도 0.40425045 : 첫번째 문서와 세번째 문서의 유사도

## 텍스트 유사도

- 첫번째 유사도 1 제외

```
1 from sklearn.metrics.pairwise import cosine_similarity
2
3 similarity_simple_pair = cosine_similarity(feature_vect_simple[0] , feature_vect_simple[1:])
4 print(similarity_simple_pair)
```

```
[[0.40207758 0.40425045]]
```

### cosine\_similarity() - 쌍(pair)으로 코사인 유사도 값 제공

- 모든 개별 문서에 쌍으로 코사인 유사도 값을 계산
- 1번째 문서와 2, 3번째 문서의 코사인 유사도
- 2번째 문서와 1, 3번째 문서의 코사인 유사도
- 3번째 문서와 1, 2번째 문서의 코사인 유사도

```
1 similarity_simple_pair = cosine_similarity(feature_vect_simple, feature_vect_simple)
2 print(similarity_simple_pair)
3 print('shape:', similarity_simple_pair.shape)
```

```
[[1.          0.40207758 0.40425045]
 [0.40207758 1.          0.45647296]
 [0.40425045 0.45647296 1.          ]]
shape: (3, 3)
```



## 텍스트 유사도

- Opinion Review 데이터 셋을 이용한 문서 유사도 측정

### Opinion Review 데이터 셋

- UCI 머신러닝 리포지토리
- 51개의 텍스트 파일로 구성
- 각 파일은 Tripadvisor(호텔), Edmunds.com(자동차), Amazon.com(전자제품) 사이트에서 가져온 리뷰 문서
- 각 문서는 100개 정도의 문장으로 구성

### 데이터 전처리를 위한 함수 생성

```
1 from nltk.stem import WordNetLemmatizer
2 import nltk
3 import string
4
5 remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
6 lemmar = WordNetLemmatizer()
7
8 def LemTokens(tokens):
9     return [lemmar.lemmatize(token) for token in tokens]
10
11 def LemNormalize(text):
12     return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

## 텍스트 유사도

- Opinion Review 데이터 셋을 이용한 문서 유사도 측정

### Opinion Review 데이터 셋 불러오기

```
1 import pandas as pd
2 import glob, os
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 from sklearn.cluster import KMeans
5
6 path = './OpinionDataset1.0/OpinionDataset1.0/topics'
7 all_files = glob.glob(os.path.join(path, "*.data"))
8 filename_list = []
9 opinion_text = []
10
11 for file_ in all_files:
12     df = pd.read_table(file_, index_col=None, header=0, encoding='latin1')
13     filename_ = file_.split('###')[-1]
14     filename = filename_.split('.')[0]
15     filename_list.append(filename)
16     opinion_text.append(df.to_string())
17
18 document_df = pd.DataFrame({'filename':filename_list, 'opinion_text':opinion_text})
```

```
1 document_df.head()
```

	filename	opinion_text
0	accuracy_garmin_nuvi_255W_gps	...
1	bathroom_bestwestern_hotel_sfo	...
2	battery-life_amazon_kindle	...
3	battery-life_ipod_nano_8gb	...
4	battery-life_netbook_1005ha	...

## 텍스트 유사도

- Opinion Review 데이터 셋을 이용한 문서 유사도 측정

### 사이킷런의 TfidfVectorizer로 피터 벡터화

```
1 tfidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english', #
2                             ngram_range=(1,2), min_df=0.05, max_df=0.85 )
3 feature_vect = tfidf_vect.fit_transform(document_df['opinion_text'])
```

### 비지도학습 - k-평균군집화를 이용하여 군집화

- 전자제품, 호텔, 자동차 3가지 주제로 군집화

```
1 km_cluster = KMeans(n_clusters=3, max_iter=10000, random_state=0)
2 km_cluster.fit(feature_vect)
3 cluster_label = km_cluster.labels_
4 cluster_centers = km_cluster.cluster_centers_
5 document_df['cluster_label'] = cluster_label
```

```
1 cluster_label
```

```
array([0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 0, 1, 1,
       2, 0, 1, 2, 0, 0, 1, 2, 1, 1, 1, 0, 0, 0, 0, 2, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 2, 0, 0, 0])
```

```
1 document_df.head()
```

	filename	opinion_text	cluster_label
0	accuracy_garmin_nuvi_255W_gps	...	0
1	bathroom_bestwestern_hotel_sfo	...	1
2	battery-life_amazon_kindle	...	0
3	battery-life_ipod_nano_8gb	...	0
4	battery-life_netbook_1005ha	...	0

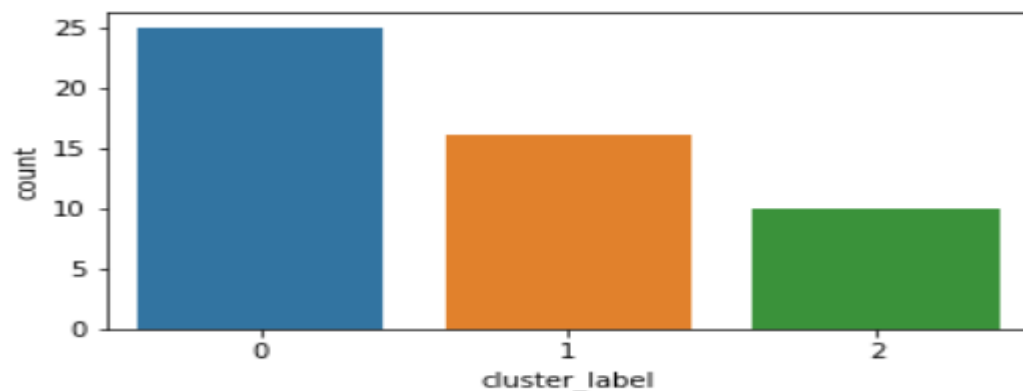
## 텍스트 유사도

- Opinion Review 데이터 셋을 이용한 문서 유사도 측정

```
1 # 클러스터별 개수 확인
2 document_df['cluster_label'].value_counts()

0    25
1    16
2    10
Name: cluster_label, dtype: int64
```

```
1 # 클러스터별 개수 시각화
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 fig, ax = plt.subplots(ncols=1)
6 fig.set_size_inches(6, 3)
7 sns.countplot(document_df['cluster_label'])
```



## 텍스트 유사도

### Opinion Review 데이터 셋을 이용한 문서 유사도 측정

#### 호텔을 주제로 클러스터링 된 문서를 이용하여 특정 문서와 다른 문서간의 유사도 측정

- 1) 호텔을 주제로 군집화된 문서의 인덱스 추출
  - 추출한 인덱스를 이용해 TfidfVectorizer 객체 변수인 feature\_vect에서 호텔로 군집화된 문서의 피처 벡처벡터를 추출
- 2) 호텔로 클러스터링된 데이터 중 첫번째 문서를 추출
- 3) 첫번째 문서와 다른 문서간의 코사인 유사도 측정

```
1 from sklearn.metrics.pairwise import cosine_similarity
2
3 # 1) cluster_label=1인 데이터는 호텔로 클러스터링된 데이터임. DataFrame에서 해당 Index를 추출
4 hotel_indexes = document_df[document_df['cluster_label']==1].index
5 print('호텔로 클러스터링 된 문서들의 DataFrame Index:', hotel_indexes)
6
7 # 2) 호텔로 클러스터링된 데이터 중 첫번째 문서를 추출하여 파일명 표시.
8 comparison_docname = document_df.iloc[hotel_indexes[0]]['filename']
9 print('##### 비교 기준 문서명(첫번째 문서):', comparison_docname, '와 타 문서 유사도#####')
10
11 ''' document_df에서 추출한 Index 객체를 feature_vect로 입력하여 호텔 클러스터링된 feature_vect 추출
12 이를 이용하여 호텔로 클러스터링된 문서 중 첫번째 문서와 다른 문서간의 코사인 유사도 측정. '''
13
14 # 3) 첫번째 문서와 다른 문서간의 코사인 유사도 측정
15 similarity_pair = cosine_similarity(feature_vect[hotel_indexes[0]] , feature_vect[hotel_indexes])
16 print(similarity_pair)
```

호텔로 클러스터링 된 문서들의 DataFrame Index: Int64Index([1, 13, 14, 15, 20, 21, 24, 28, 30, 31, 32, 38, 39, 40, 45, 46], dtype='int64')

##### 비교 기준 문서명(첫번째 문서): bathroom\_bestwestern\_hotel\_sfo 와 타 문서 유사도#####

```
[[1.          0.0430688  0.05221059  0.06189595  0.05846178  0.06193118
  0.03638665  0.11742762  0.38038865  0.32619948  0.51442299  0.11282857
  0.13989623  0.1386783  0.09518068  0.07049362]]
```

## 텍스트 유사도

- Opinion Review 데이터 셋을 이용한 문서 유사도 측정

### 코사인 유사도 시각화

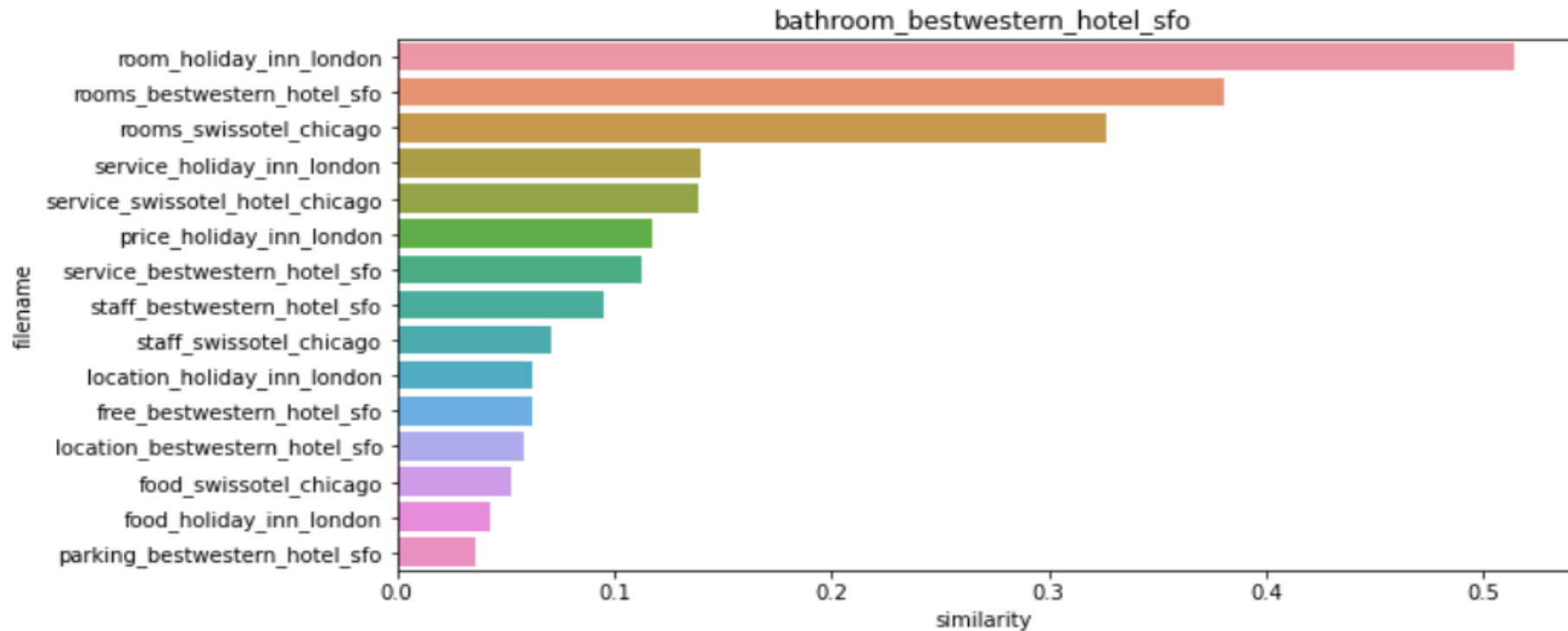
```
1 import seaborn as sns
2 import numpy as np
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5
6 # argsort()를 이용하여 앞예제의 첫번째 문서와 타 문서간 유사도가 큰 순으로 정렬한 인덱스 반환하되 자기 자신은 제외.
7 sorted_index = similarity_pair.argsort()[::-1]
8 sorted_index = sorted_index[:, 1:]
9 print(sorted_index)
10
11 # 유사도가 큰 순으로 hotel_indexes를 추출하여 재 정렬.
12 print(hotel_indexes)
13 hotel_sorted_indexes = hotel_indexes[sorted_index.reshape(-1,)]
14
15 # 유사도가 큰 순으로 유사도 값을 재정렬하되 자기 자신은 제외
16 hotel_1_sim_value = np.sort(similarity_pair.reshape(-1,))[::-1]
17 hotel_1_sim_value = hotel_1_sim_value[1:]
18
19 # 유사도가 큰 순으로 정렬된 Index와 유사도값을 이용하여 파일명과 유사도값을 Seaborn 막대 그래프로 시각화
20 hotel_1_sim_df = pd.DataFrame()
21 hotel_1_sim_df['filename'] = document_df.iloc[hotel_sorted_indexes]['filename']
22 hotel_1_sim_df['similarity'] = hotel_1_sim_value
23
24 plt.figure(figsize=(10,5))
25 sns.barplot(x='similarity', y='filename', data=hotel_1_sim_df)
26 plt.title(comparison_docname)
```

```
[[10  8  9 12 13  7 11 14 15  5  3  4  2  1  6]]
```

```
Int64Index([1, 13, 14, 15, 20, 21, 24, 28, 30, 31, 32, 38, 39, 40, 45, 46], dtype='int64')
```

## 텍스트 유사도

- Opinion Review 데이터 셋을 이용한 문서 유사도 측정



- 해석)
- 샌프란시스코의 베스트 웨스턴 호텔 화장실 리뷰인 bathroom\_bestwestern\_hotel\_sfo 문서와
- 가장 비슷한 문서는 room\_holiday\_inn\_london 문서로
- 약 0.572의 코사인 유사도 값을 가진다.

THANK YOU