

# 텍스트분석

8 - 8,9

Kuggle\_서민

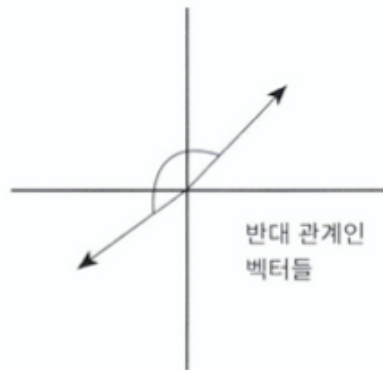
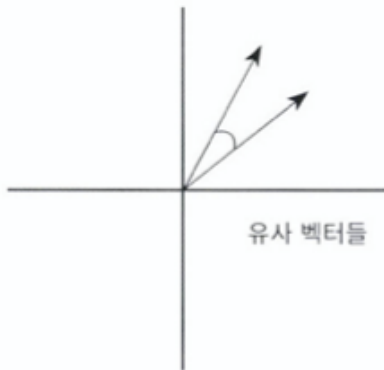
## 8.8 \_ 문서 유사도

문서 유사도 측정 방법 - 코사인 유사도 / 두 벡터의 사잇각

코사인 유사도(Cosine Similarity) - 벡터들의 크기보다는 상호 방향성이 얼마나 유사한지

➡ 두 벡터 사이의 사잇각

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



## 8.8 \_ 문서 유사도

### cos\_similarity 함수

```
import numpy as np

def cos_similarity(v1, v2):
    dot_product = np.dot(v1, v2)
    l2_norm = (np.sqrt(sum(np.square(v1))) * np.sqrt(sum(np.square(v2))))
    similarity = dot_product / l2_norm

    return similarity
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
doc_list = ['if you take the blue pill, the story ends',
            'if you take the red pill, you stay in Wonderland',
            'if you take the red pill, I show you how deep the rabbit hole goes']
```

```
tfidf_vect_simple = TfidfVectorizer()
feature_vect_simple = tfidf_vect_simple.fit_transform(doc_list)
print(feature_vect_simple.shape)
```

TF-IDF로 벡터화된 행렬로 변환  
For 유사도 비교

## 8.8 \_ 문서 유사도

cos\_similarity 함수: 1번째 & 2번째 문장 비교

**[Output]**

(3, 18)

반환된 행렬이 희소 행렬이므로  
배열로 만들어주기 위해서 밀집  
행렬로 변환한 뒤 다시 각각의  
배열로 변환



Feature\_vect\_dense[0]: doc\_list의 첫 번째 문서의 피처 벡터화

Feature\_vect\_dense[1]: doc\_list의 첫 번째 문서의 피처 벡터화

```
# TFidfVectorizer로 transform()한 결과는 희소 행렬이므로 밀집 행렬로 변환.  
feature_vect_dense = feature_vect_simple.todense()
```

```
#첫 번째 문장과 두 번째 문장의 피처 벡터 추출  
vect1 = np.array(feature_vect_dense[0]).reshape(-1, )  
vect2 = np.array(feature_vect_dense[1]).reshape(-1, )
```

```
#첫 번째 문장과 두 번째 문장의 피처 벡터로 두 개 문장의 코사인 유사도 추출  
similarity_simple = cos_similarity(vect1, vect2 )  
print('문장 1, 문장 2 Cosine 유사도: {0:.3f}'.format(similarity_simple))
```

**[Output]**

문장 1, 문장 2 Cosine 유사도: 0.402

## 8.8 \_ 문서 유사도

cos\_similarity 함수: 2번째 & 3번째 문장 비교

```
vect1 = np.array(feature_vect_dense[0]).reshape(-1, )  
vect3 = np.array(feature_vect_dense[2]).reshape(-1, )  
similarity_simple = cos_similarity(vect1, vect3 )  
print('문장 1, 문장 3 Cosine 유사도: {0:.3f}'.format(similarity_simple))  
  
vect2 = np.array(feature_vect_dense[1]).reshape(-1, )  
vect3 = np.array(feature_vect_dense[2]).reshape(-1, )  
similarity_simple = cos_similarity(vect2, vect3 )  
print('문장 2, 문장 3 Cosine 유사도: {0:.3f}'.format(similarity_simple))
```

### 【Output】

문장 1, 문장 3 Cosine 유사도: 0.404

문장 2, 문장 3 Cosine 유사도: 0.456

## 8.8 \_ 문서 유사도

Sklearn.metrics.pairwise.cosine\_similarity API

Cosine\_similarity() 함수

- 1. 첫번째 파라미터: 비교 기준이 되는 문서의 피처행렬
- 2. 두번째 파라미터: 비교되는 문서의 피처행렬

Cos_similarity()	Cosiner_similarity
변환 작업 필요함! (아까 밀집행렬로 변환후 다시 배열로 한것)	희소행렬, 밀집행렬 가능/ 행렬, 배열 가능 따라서 변환작업 필요 없음

## 8.8 \_ 문서 유사도

cosine\_similarity()

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_simple_pair = cosine_similarity(feature_vect_simple[0], feature_vect_simple)
print(similarity_simple_pair)
```

**[Output]**

2번째 문서와 3번째 문서의 유사도

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

1번째와 2번째 문서의 유사도

## 8.8 \_ 문서 유사도

cosine\_similarity(): 쌍으로 나타내기

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

【Output】

```
[[1.          0.40207758 0.40425045]
 [0.40207758 1.          0.45647296]
 [0.40425045 0.45647296 1.          ]]
```

```
shape: (3, 3)
```

1번째 문서와 2,3번째 문서의 유사도

2번째 문서와 1,3번째 문서의 유사도

3번째 문서와 1,2번째 문서의 유사도



## 8.8 \_ 문서 유사도

### Opinion Review 데이터 세트를 이용한 문서 유사도 측정

```
import pandas as pd
import glob, os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

path = r'C:\Users\chkwon\Text\OpinosisDataset1.0\OpinosisDataset1.0\topics'
all_files = glob.glob(os.path.join(path, "*.data"))
filename_list = []
opinion_text = []

for file_ in all_files:
    df = pd.read_table(file_, index_col=None, header=0, encoding='latin1')
    filename_ = file_.split('\\')[-1]
    filename = filename_.split('.')[0]
    filename_list.append(filename)
    opinion_text.append(df.to_string())

document_df = pd.DataFrame({'filename':filename_list, 'opinion_text':opinion_text})

tfidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english',
                             ngram_range=(1, 2), min_df=0.05, max_df=0.85 )
```

```
feature_vect = tfidf_vect.fit_transform(document_df['opinion_text'])

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

- 호텔 군집화 데이터를 기반으로 별도의 TF-IDF 벡터화를 수행하지 않고, 바로 위에 TfidfVectorizer로 만들어진 데이터를 추출
- Feature\_vect에서 호텔로 군집화된 문서의 피처 벡터 추출

## 8.8 \_ 문서 유사도

### Opinion review 데이터 세트

```
from sklearn.metrics.pairwise import cosine_similarity

# cluster_label=1인 데이터는 호텔로 군집화된 데이터임. DataFrame에서 해당 인덱스
hotel_indexes = document_df[document_df['cluster_label']==1].index
print('호텔로 군집화 된 문서들의 DataFrame Index:', hotel_indexes)

# 호텔로 군집화된 데이터 중 첫 번째 문서를 추출해 파일명 표시.
comparison_docname = document_df.iloc[hotel_indexes[0]]['filename']
print('##### 비교 기준 문서명 ', comparison_docname, ' 와 타 문서 유사도#####')

''' document_df에서 추출한 Index 객체를 feature_vect로 입력해 호텔 군집화된 feature_vect 추출
이를 이용해 호텔로 군집화된 문서 중 첫 번째 문서와 다른 문서 간의 코사인 유사도 측정.'''
similarity_pair = cosine_similarity(feature_vect[hotel_indexes[0]], feature_vect[hotel_indexes])
print(similarity_pair)
```

#### [Output]

호텔로 군집화 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 bathroom\_bestwestern\_hotel\_sfo 와 타 문서 유사도#####

```
[[1.          0.05907195 0.05404862 0.03739629 0.06629355 0.06734556
  0.04017338 0.13113702 0.41011101 0.3871916  0.57253197 0.10600704
  0.13058128 0.1602411  0.05539602 0.05839754]]
```

## 8.8 \_ 문서 유사도

### Opinion review 데이터 세트: 시각화

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# 첫 번째 문서와 타 문서 간 유사도가 큰 순으로 정렬한 인덱스를 추출하되 자기 자신은 제외.
sorted_index = similarity_pair.argsort()[1:, :-1]
sorted_index = sorted_index[:, 1:]

# 유사도가 큰 순으로 hotel_indexes를 추출해 재정렬
hotel_sorted_indexes = hotel_indexes[sorted_index.reshape(-1)]

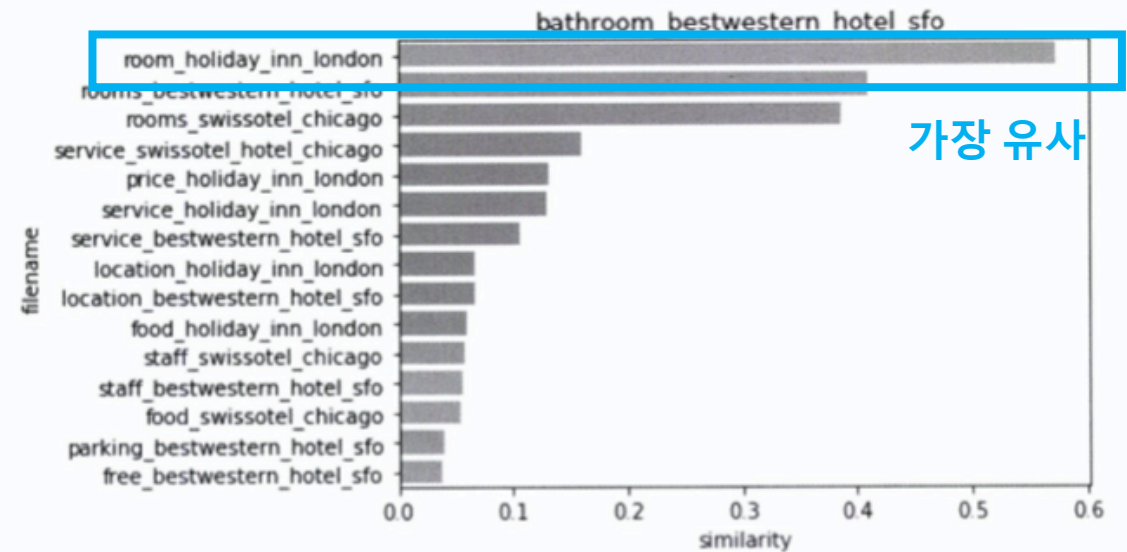
# 유사도가 큰 순으로 유사도 값을 재정렬하되 자기 자신은 제외
hotel_1_sim_value = np.sort(similarity_pair.reshape(-1))[1:]
hotel_1_sim_value = hotel_1_sim_value[1:]

# 유사도가 큰 순으로 정렬된 인덱스와 유사도 값을 이용해 파일명과 유사도 값을 막대 그래프로 시각화
hotel_1_sim_df = pd.DataFrame()
hotel_1_sim_df['filename'] = document_df.iloc[hotel_sorted_indexes]['filename']
hotel_1_sim_df['similarity'] = hotel_1_sim_value

sns.barplot(x='similarity', y='filename', data=hotel_1_sim_df)
plt.title(comparison_docname)
```

Cosine\_similarity()는 쌍 형태의 ndarray를 반환하므로  
판단스 인덱스로 이용하기 위해 reshape(-1)로 차원 변경

【Output】



## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### 한글 NLP 처리의 어려움

1. 띄어쓰기	2. 다양한 조사
<p>띄어쓰기로 인한 의미 변경</p> <ol style="list-style-type: none"><li>1) 아버지가 방에 들어가신다</li><li>2) 아버지 가방에 들어가신다</li></ol>	<p>조사는 경우의 수가 많기 때문에 어근 추출등의 전처리 시 제거하기 까다롭다</p> <ol style="list-style-type: none"><li>1) 집+이</li><li>2) 집+으로</li><li>3) 집+에서</li><li>4) 집+에</li></ol>

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### KoNLPy 소개

KoNLPy : 파이썬의 대표적인 한글 형태소 패키지

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### KoNLPy 소개

KoNLPy : 파이썬의 대표적인 한글 형태소 패키지

형태소: 단어의 의미로써 가지는 최소 단위



형태소 분석(Morphological Analysis) :

말뭉치 → 형태소 어근 단위로 쪼갬 → 형태소에 품사 태깅(POS tagging)



## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### KoNLPy 설치

설치는 사이트와 책 참고해주세요!! 운영체제와 컴터환경마다 조금씩 다르니닷!

<http://konlpy-ko.readthedocs.io/ko/v.0.4.3/install/> Java가 설치되어 있어야함

### 윈도우

1. Java 1.7+이 설치되어 있나요?

2. JAVA\_HOME 설정하기

3. JPytype1 (>=0.5.7)을 다운로드 받고 설치. 다운 받은 .whl 파일을 설치하기 위해서는 pip  
을 업그레이드 해야할 수 있습니다.

```
> pip install --upgrade pip  
> pip install JPytype1-0.5.7-cp27-none-win_amd64.whl
```

```
C:\Users\chkwon>conda install -c conda-forge jpytype1
```

```
C:\Users\chkwon>pip install --upgrade pip
```

```
C:\Users\chkwon>pip install JPytype1-0.6.3-cp36-cp36m-win_amd64.whl
```

JPytype allows full access to Java class libraries.

JPytype1-0.6.3-cp27-cp27m-win32.whl

JPytype1-0.6.3-cp27-cp27m-win\_amd64.whl

JPytype1-0.6.3-cp34-cp34m-win32.whl

JPytype1-0.6.3-cp34-cp34m-win\_amd64.whl

JPytype1-0.6.3-cp35-cp35m-win32.whl

JPytype1-0.6.3-cp35-cp35m-win\_amd64.whl

JPytype1-0.6.3-cp36-cp36m-win32.whl

JPytype1-0.6.3-cp36-cp36m-win\_amd64.whl

JPytype1-0.6.3-cp37-cp37m-win32.whl

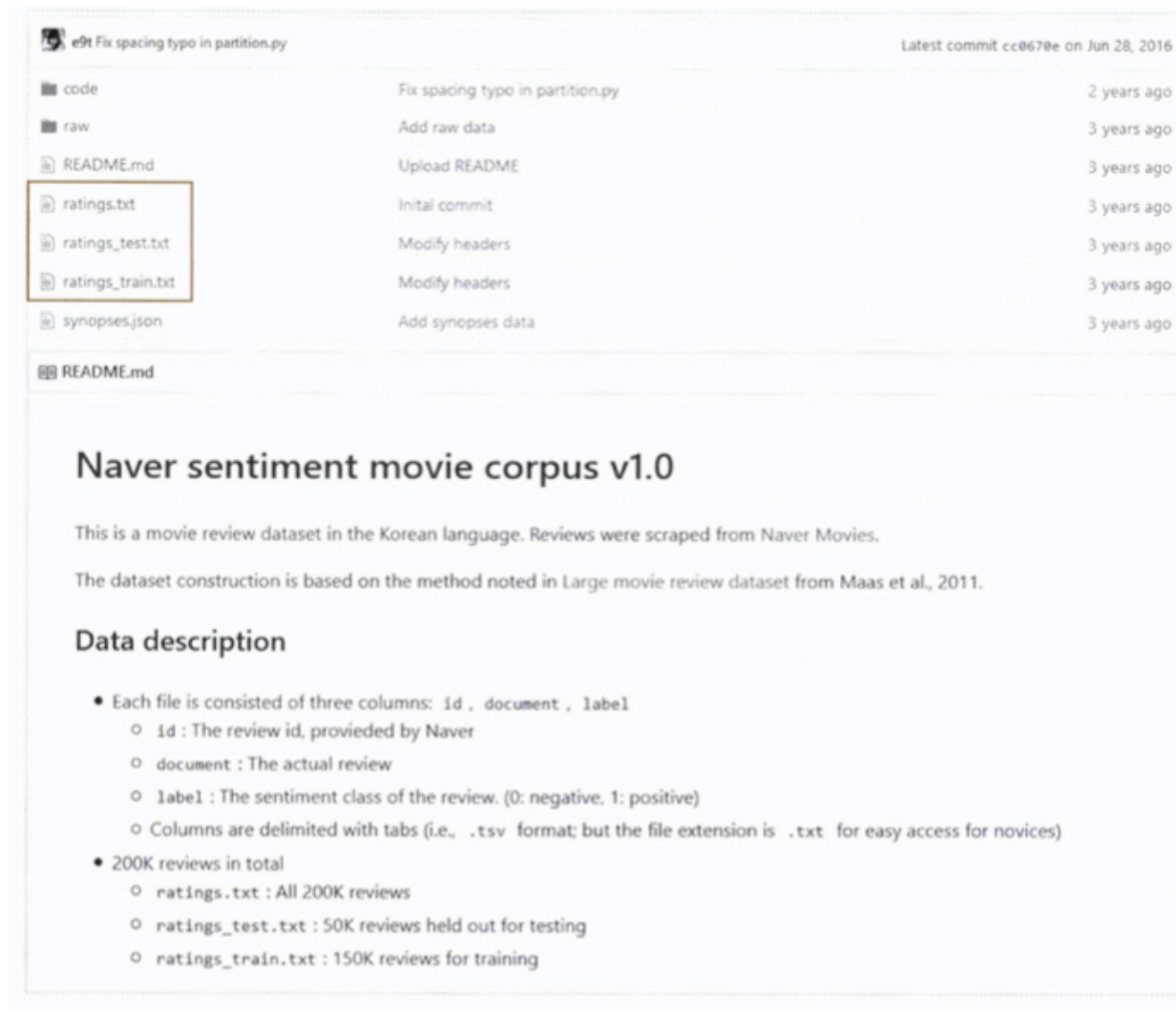
JPytype1-0.6.3-cp37-cp37m-win\_amd64.whl

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

데이터 로딩

네이버 영화 평점 데이터

<http://github.com/e9t/nsmc>



The screenshot displays the GitHub repository for 'e9t/nsmc'. The top section shows the repository name and the latest commit. Below this, a table lists the files and their commit history:

File	Commit Message	Time
code	Fix spacing typo in partition.py	2 years ago
raw	Add raw data	3 years ago
README.md	Upload README	3 years ago
ratings.txt	Initial commit	3 years ago
ratings_test.txt	Modify headers	3 years ago
ratings_train.txt	Modify headers	3 years ago
synopses.json	Add synopses data	3 years ago

Below the file list, the README.md content is visible. It is titled 'Naver sentiment movie corpus v1.0' and describes the dataset as a Korean movie review dataset. It mentions that the dataset construction is based on the method from Maas et al., 2011. The 'Data description' section lists the following details:

- Each file is consisted of three columns: id, document, label
  - id: The review id, provided by Naver
  - document: The actual review
  - label: The sentiment class of the review. (0: negative, 1: positive)
  - Columns are delimited with tabs (i.e., .tsv format; but the file extension is .txt for easy access for novices)
- 200K reviews in total
  - ratings.txt: All 200K reviews
  - ratings\_test.txt: 50K reviews held out for testing
  - ratings\_train.txt: 150K reviews for training



## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### 데이터 로딩

#### 네이버 영화 평점 데이터

```
import pandas as pd

train_df = pd.read_csv('ratings_train.txt', sep='\t')
train_df.head(3)
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄...오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0

0: 부정

1: 긍정



```
train_df['label'].value_counts( )
```

**[Output]**

```
0    75173
1    74827
```

균등분포

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### 데이터 로딩

#### 네이버 영화 평점 데이터

‘document’ 칼럼에 Null과 숫자는 공백으로 변환 – re이용

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0

```
import re

train_df = train_df.fillna(' ')
# 정규 표현식을 이용해 숫자를 공백으로 변경(정규 표현식으로 \d는 숫자를 의미함.)
train_df['document'] = train_df['document'].apply( lambda x : re.sub(r"\d+", " ", x) )

# 테스트 데이터 세트를 로딩하고 동일하게 Null 및 숫자를 공백으로 변환
test_df = pd.read_csv('ratings_test.txt', sep='\t')
test_df = test_df.fillna(' ')
test_df['document'] = test_df['document'].apply( lambda x : re.sub(r"\d+", " ", x) )
```

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### 데이터 로딩

#### 네이버 영화 평점 데이터

```
from konlpy.tag import Twitter

twitter = Twitter()

def tw_tokenizer(text):
    # 입력 인자로 들어온 텍스트를 형태소 단어로 토큰화해 리스트 형태로 반환
    tokens_ko = twitter.morphs(text)
    return tokens_ko
```

형태소 단어로 토큰화

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### 데이터 로딩

#### 네이버 영화 평점 데이터

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Twitter 객체의 morphs( ) 객체를 이용한 tokenizer를 사용. ngram_range는 (1, 2)
tfidf_vect = TfidfVectorizer(tokenizer=tw_tokenizer, ngram_range=(1, 2), min_df=3, max_df=0.9)
tfidf_vect.fit(train_df['document'])
tfidf_matrix_train = tfidf_vect.transform(train_df['document'])
```

TfidfVectorizer를 이용해 TF\_IDF  
피쳐 모델 생성

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### 데이터 로딩

#### 네이버 영화 평점 데이터

```
# 로지스틱 회귀를 이용해 감성 분석 분류 수행.  
lg_clf = LogisticRegression(random_state=0)  
  
# 파라미터 C 최적화를 위해 GridSearchCV를 이용.  
params = { 'C': [1, 3.5, 4.5, 5.5, 10] }  
grid_cv = GridSearchCV(lg_clf, param_grid=params, cv=3, scoring='accuracy', verbose=1)  
grid_cv.fit(tfidf_matrix_train, train_df['label'])  
print(grid_cv.best_params_, round(grid_cv.best_score_, 4))
```

로지스틱 회귀를 이용하여 분류 기반의 감성 분석 수행  
- 그리드서치cv 이용

#### **[Output]**

```
{'C': 3.5} 0.8593
```

C가 3.5일 때 최고 0.8593의 정확도

## 8.9 \_ 한글 텍스트 처리 – 네이버 영화 평점 감성분석

### 데이터 로딩

#### 네이버 영화 평점 데이터

```
from sklearn.metrics import accuracy_score

# 학습 데이터를 적용한 TfidfVectorizer를 이용해 테스트 데이터를 TF-IDF 값으로 피쳐 변환함.
tfidf_matrix_test = tfidf_vect.transform(test_df['document'])

# classifier는 GridSearchCV에서 최적 파라미터로 학습된 classifier를 그대로 이용
best_estimator = grid_cv.best_estimator_
preds = best_estimator.predict(tfidf_matrix_test)

print('Logistic Regression 정확도: ', accuracy_score(test_df['label'], preds))
```

#### **[Output]**

Logistic Regression 정확도: 0.86172

TfidfVectorizer를 그대로 사용

: 학습 시 설정된 TfidfVectorizer와 피쳐 개수와 테스트 데이터를 TfidfVectorizer로 변환할 피쳐 개수가 같아짐