# Chapter 08. 텍스트 분석

## 8.6 토픽 모델링(Topic Modeling) - 20 뉴스그룹

토픽 모델링(Topic Modeling)

- 문서 집합에 숨어 있는 주제를 찾아내는 것
- 사람: 더 함축적인 의미로 문장을 요약
- 머신러닝: 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출
- LSA(Latent Semantic Analysis)와 LDA(Latent Dirichlet Allocation)

20 뉴스그룹 데이터 세트 적용

- fetch_20newsgroups()의 categories 파라미터로 필터링
- Count 기반으로 벡터화 변환
- 피처 벡터화된 데이터 세트에 LDA기반의 토픽 모델링 적용

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

# 모터사이클, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 전자공학, 의학 8개
cats = ['rec.motorcycles', 'rec.sport.baseball', 'comp.graphi
        'talk.politics.mideast', 'soc.religion.christian', 's

# 위에서 cats 변수로 기재된 카테고리만 추출. featch_20newsgroups( )으
news_df= fetch_20newsgroups(subset='all', remove=('headers',
                            categories=cats, random_state=0)

# LDA는 Count 기반의 벡터화만 적용합니다.
count_vect = CountVectorizer(max_df=0.95, max_features=1000,
                                                        ngra
```

```
feat_vect = count_vect.fit_transform(news_df.data)
print('CountVectorizer Shape:', feat_vect.shape)
```

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

→ 7862개의 문서가 1000개의 피처로 구성된 행렬 데이터

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

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

→ LatentDirichletAllocation 클래스의 n_components 파라미터를 이용해 토픽 개수 8개로 조정

→ LatentDirichletAllocation.fit(데이터 세트)을 수행하면 객체는 components_ 속성 값을 가짐
→ components_: 개별 토픽별로 각 word 피처가 얼마나 많은 그 토픽에 할당됐는지에 대한 수치
→ 높은 값일수록 해당 word 피처는 그 토픽의 중심 word가 됨

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

```
### output ###
(8, 1000)
array([[3.60992018e+01, 1.35626798e+02, 2.15751867e+01, ...,
        3.02911688e+01, 8.66830093e+01, 6.79285199e+01],
       [1.25199920e-01, 1.44401815e+01, 1.25045596e-01, ...,
        1.81506995e+02, 1.25097844e-01, 9.39593286e+01],
       [3.34762663e+02, 1.25176265e-01, 1.46743299e+02, ...,
```

```
         1.25105772e-01, 3.63689741e+01, 1.25025218e-01],
        ...,
        [3.60204965e+01, 2.08640688e+01, 4.29606813e+00, ...,
         1.45056650e+01, 8.33854413e+00, 1.55690009e+01],
        [1.25128711e-01, 1.25247756e-01, 1.25005143e-01, ...,
         9.17278769e+01, 1.25177668e-01, 3.74575887e+01],
        [5.49258690e+01, 4.47009532e+00, 9.88524814e+00, ...,
         4.87048440e+01, 1.25034678e-01, 1.25074632e-01]])
```

→ array[8, 4000]으로 구성. 8개의 토픽별로 1000개의 word 피처가 해당 토픽별로 연관도 값을 가짐

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

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

        # top_indexes 대상인 인덱스별로 feature_names에 해당하는 wo
        feature_concat = ' '.join([feature_names[i] for i in
        print(feature_concat)

# CountVectorizer 객체 내의 전체 word의 명칭을 get_features_names(
feature_names = count_vect.get_feature_names_out()
  ### renamed function: get_features_names( ) -> get_features

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

```
### output ###
Topic # 0
year 10 game medical health team 12 20 disease cancer 1993 ga
Topic # 1
don just like know people said think time ve didn right going
```

```
Topic # 2
image file jpeg program gif images output format files color 
Topic # 3
like know don think use does just good time book read informa
Topic # 4
armenian israel armenians jews turkish people israeli jewish 
Topic # 5
edu com available graphics ftp data pub motif mail widget sof
Topic # 6
god people jesus church believe christ does christian say thi
Topic # 7
use dos thanks windows using window does display help like pr
```

→ display_topics() 함수를 만들어서 각 토픽별로 연관도가 높은 순으로 word 나열

→ Topic #1, #3, #5가 주로 애매한 주제어가 추출
→ 모터사이클, 야구 주제의 경우 명확한 주제어가 추출되지 않았음.

# 8.7 문서 군집화 소개와 실습(Opinion Review 데이터 세트)

### 문서 군집화(Document Clustering) 개념

- 비슷한 텍스트 구성의 문서를 군집화(Clustering)하는 것
- 동일한 군집에 속하는 문서를 같은 카테고리 소속으로 분류
- 학습 데이터 세트가 필요 없는 비지도학습 기반으로 동작

### Opinion Review 데이터 세트를 이용한 문서 군집화 수행하기

Opinion Review 데이터 세트

- 51개의 텍스트 파일. 각 파일은 Tripadvisor(호텔), Edmunds.com(자동차), Amazon.com(전자제품) 사이트에서 가져온 리뷰 문서이고, 각 문서는 약 100개 정도의 문장을 가지고 있음

여러 개의 파일 DataFrame으로 로딩

```
import pandas as pd
import glob, os
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_colwidth', 700)

# 다음은 저자의 컴퓨터에서 압축 파일을 풀어놓은 디렉터리이니, 각자 디렉터리
#path = r'C:\Users\chkwon\Text\OpinosisDataset1.0\OpinosisDat
# path로 지정한 디렉터리 밑에 있는 모든 .data 파일들의 파일명을 리스트로
all_files = glob.glob(os.path.join(path, "*.data"))
filename_list = []
opinion_text = []

# 개별 파일들의 파일명은 filename_list 리스트로 취합,
# 개별 파일들의 파일 내용은 DataFrame 로딩 후 다시 string으로 변환하여
for file_ in all_files:
    # 개별 파일을 읽어서 DataFrame으로 생성
    df = pd.read_table(file_,index_col=None, header=0,encodin

    # 절대경로로 주어진 file 명을 가공. 만일 Linux에서 수행시에는 아래
    # 맨 마지막 .data 확장자도 제거
    filename_ = file_.split('/')[-1]
    filename = filename_.split('.')[0]

    # 파일명 리스트와 파일 내용 리스트에 파일명과 파일 내용을 추가.
    filename_list.append(filename)
    opinion_text.append(df.to_string())

# 파일명 리스트와 파일 내용 리스트를  DataFrame으로 생성
document_df = pd.DataFrame({'filename':filename_list, 'opinio
document_df.head()
```

→ 파일 이름(filename) 자체만으로 의견(opinion)의 텍스트(text)가 어떠한 제품/서비스에 대한 리뷰인지 잘 알 수 있음.

TfidfVectorizer의 tokenizer 인자로 사용될 어근 변환 함수 LemNormalize( ) 생성

```
from nltk.stem import WordNetLemmatizer
import nltk
import string

remove_punct_dict = dict((ord(punct), None) for punct in stri
lemmar = WordNetLemmatizer()

# 입력으로 들어온 token 단어들에 대해서 lemmatization 어근 변환.
def LemTokens(tokens):
    return [lemmar.lemmatize(token) for token in tokens]

# TfidfVectorizer 객체 생성 시 tokenizer인자로 해당 함수를 설정하여 T
# 입력으로 문장을 받아서 stop words 제거-> 소문자 변환 -> 단어 토큰화 -
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translat
```

문서를 TF-IDF 형태로 피처 벡터화하기

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_wor
                             ngram_range=(1,2), min_df=0.05, m

# opinion_text 칼럼값으로 feature vectorization 수행
feature_vect = tfidf_vect.fit_transform(document_df['opinion_
```

TF-IDF 변환된 피처 벡터화 행렬 데이터에 대해 K-평균 군집화 수행

```
from sklearn.cluster import KMeans

# 5개 집합으로 군집화 수행. 예제를 위해 동일한 클러스터링 결과 도출용 rand
km_cluster = KMeans(n_clusters=5, max_iter=10000, random_stat
km_cluster.fit(feature_vect)
```

```
cluster_label = km_cluster.labels_
cluster_centers = km_cluster.cluster_centers_
```

```
document_df['cluster_label'] = cluster_label
document_df.head()
```

군집화 결과 확인하기

```
document_df[document_df['cluster_label']==0].sort_values(by='
```

```
document_df[document_df['cluster_label']==1].sort_values(by='
```

```
document_df[document_df['cluster_label']==2].sort_values(by='
```

```
document_df[document_df['cluster_label']==3].sort_values(by='
```

```
document_df[document_df['cluster_label']==4].sort_values(by='
```

→ 전반적으로 군집 개수가 약간 많게 설정돼 있어서 세분화되어 군집화된 경향이 있음

중심 개수 5개 → 3개로 낮춰서 군집화한 후 결과 확인하기

```
from sklearn.cluster import KMeans

# 3개의 집합으로 군집화
km_cluster = KMeans(n_clusters=3, max_iter=10000, random_stat
km_cluster.fit(feature_vect)
cluster_label = km_cluster.labels_

# 소속 클러스터를 cluster_label 칼럼으로 할당하고 cluster_label 값으로
document_df['cluster_label'] = cluster_label
document_df.sort_values(by='cluster_label')
```

### 군집별 핵심 단어 추출하기

KMeans 객체 clusters_centers_ 속성

- 배열 값으로 제공, 행은 개별 군집, 열은 개별 피처 의미
- 각 배열 내의 값은 개별 군집 내의 상대 위치를 숫자 값으로 표현한 일종의 좌표 값

```
cluster_centers = km_cluster.cluster_centers_
print('cluster_centers shape :',cluster_centers.shape)
print(cluster_centers)
```

```
### output ###
cluster_centers shape : (3, 4611)
[[0.         0.00099499 0.00174637 ... 0.         0.00183397 (
 [0.01005322 0.         0.         ... 0.00706287 0.         (
 [0.         0.00092551 0.         ... 0.         0.         (
```

→ (3, 4611) 배열: 군집이 3개, word 피처가 4611개로 구성
→ 각 행의 배열 값은 각 군집 내의 4611개 피처의 위치가 개별 중심과 얼마나 가까운가를 상대 값으로 나타낸 것. 0~1. 1에 가까울수록 중심과 가까운 값

get_cluster_details() 함수

```
# 군집별 top n 핵심단어, 그 단어의 중심 위치 상대값, 대상 파일명들을 반환
def get_cluster_details(cluster_model, cluster_data, feature_
    cluster_details = {}

    # cluster_centers array의 값이 큰 순으로 정렬된 인덱스 값을 반환
    # 군집 중심점(centroid)별 할당된 word 피처들의 거리값이 큰 순으로 
    centroid_feature_ordered_ind = cluster_model.cluster_cent

    # 개별 군집별로 반복하면서 핵심 단어, 그 단어의 중심 위치 상대값, 대상
    for cluster_num in range(clusters_num):
        # 개별 군집별 정보를 담을 데이터 초기화.
```

```
        cluster_details[cluster_num] = {}
        cluster_details[cluster_num]['cluster'] = cluster_num

        # cluster_centers_.argsort()[:,::-1] 로 구한 인덱스를 이용
        top_feature_indexes = centroid_feature_ordered_ind[cl
        top_features = [ feature_names[ind] for ind in top_fe

        # top_feature_indexes를 이용해 해당 피처 단어의 중심 위치 상
        top_feature_values = cluster_model.cluster_centers_[c

        # cluster_details 딕셔너리 객체에 개별 군집별 핵심 단어와 중심
        cluster_details[cluster_num]['top_features'] = top_fe
        cluster_details[cluster_num]['top_features_value'] =
        filenames = cluster_data[cluster_data['cluster_label'
        filenames = filenames.values.tolist()

        cluster_details[cluster_num]['filenames'] = filenames

    return cluster_details
```

print_cluster_details() 함수

```
def print_cluster_details(cluster_details):
    for cluster_num, cluster_detail in cluster_details.items(
        print('####### Cluster {0}'.format(cluster_num))
        print('Top features:', cluster_detail['top_features']
        print('Reviews 파일명 :',cluster_detail['filenames'][:7
        print('==================================================
```

각 군집별 핵심 단어 찾기

```
feature_names = tfidf_vect.get_feature_names_out()

cluster_details = get_cluster_details(cluster_model=km_cluste
```

```
                                        feature_names=feature_names
print_cluster_details(cluster_details)
```

```
### output ###
####### Cluster 0
Top features: ['room', 'hotel', 'service', 'staff', 'food', '
Reviews 파일명 : ['service_holiday_inn_london', 'service_bestwe
==================================================
####### Cluster 1
Top features: ['screen', 'battery', 'keyboard', 'battery life
Reviews 파일명 : ['speed_garmin_nuvi_255W_gps', 'screen_garmin_
==================================================
####### Cluster 2
Top features: ['interior', 'seat', 'mileage', 'comfortable',
Reviews 파일명 : ['seats_honda_accord_2008', 'quality_toyota_ca
==================================================
```

# 8.8 문서 유사도

### 문서 유사도 측정 방법 - 코사인 유사도

코사인 유사도(Cosine Similarity)

- 문서와 문서 간의 유사도 비교 시 일반적으로 사용
- 벡터와 벡터 간의 유사도를 비교할 때 벡터의 크기보다는 벡터의 상호 방향성이 얼마나 유사한지에 기반
- 두 벡터 사이의 사잇각을 구해서 얼마나 유사한지 수치로 적용한 것

### 두 벡터 사잇각

두 벡터의 사잇각에 따른 상호관계

- 유사 벡터들
- 관련성이 없는 벡터들
- 반대 관계인 벡터들

두 벡터 A와 B의 코사인 값, 유사도 $\cos\theta$

- 두 벡터의 내적을 총 벡터 크기의 합으로 나눈 것
- 내적 결과를 총 벡터 크기로 정규화(L2 Norm)한 것

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

간단한 문서에 대해서 서로 간의 문서 유사도 구하기

cos_similarity() 함수 생성
: 두 개의 넘파이 배열에 대한 코사인 유사도 구하는 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.s
    similarity = dot_product / l2_norm

    return similarity
```

doc_list로 정의된 문서 TF-IDF로 벡터화된 행렬로 변환

```
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 th

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

```
### output ###
(3, 18)
```

두 개 문서의 유사도 측정

```
# TFidfVectorizer로 transform()한 결과는 희소 행렬이므로 밀집 행렬로
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 추
similarity_simple = cos_similarity(vect1, vect2)
print('문장 1, 문장 2 Cosine 유사도: {0:.3f}'.format(similarity_
```

```
### output ###
문장 1, 문장 2 Cosine 유사도: 0.402
```

```
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_

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_
```

```
### output ###
문장 1, 문장 3 Cosine 유사도: 0.404
문장 2, 문장 3 Cosine 유사도: 0.456
```

사이킷런 sklearn.metrics.pairwise.cosine_similarity API로 유사도 측정

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_simple_pair = cosine_similarity(feature_vect_simpl
print(similarity_simple_pair)
```

```
### output ###
[[1.         0.40207758 0.40425045]]
```

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_simple_pair = cosine_similarity(feature_vect_simpl
print(similarity_simple_pair)
```

```
### output ###
[[0.40207758 0.40425045]]
```

```
similarity_simple_pair = cosine_similarity(feature_vect_simpl
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)
```

### Opinion Review 데이터 세트를 이용한 문서 유사도 측정

데이터 세트 새롭게 DataFrame으로 로드 후 문서 군집화 적용

```
import pandas as pd
import glob, os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings('ignore')

path = r'/content/drive/MyDrive/EURON/파이썬 머신러닝 완벽 가이드/t
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, encod
    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, 'opinio

tfidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_wor
                             ngram_range=(1,2), min_df=0.05, m
feature_vect = tfidf_vect.fit_transform(document_df['opinion_

km_cluster = KMeans(n_clusters=3, max_iter=10000, random_stat
```

```
km_cluster.fit(feature_vect)
cluster_label = km_cluster.labels_
cluster_centers = km_cluster.cluster_centers_
document_df['cluster_label'] = cluster_label
```

호텔을 주제로 군집화된 문서를 이용해 특정 문서와 다른 문서 간의 유사도 알아보기

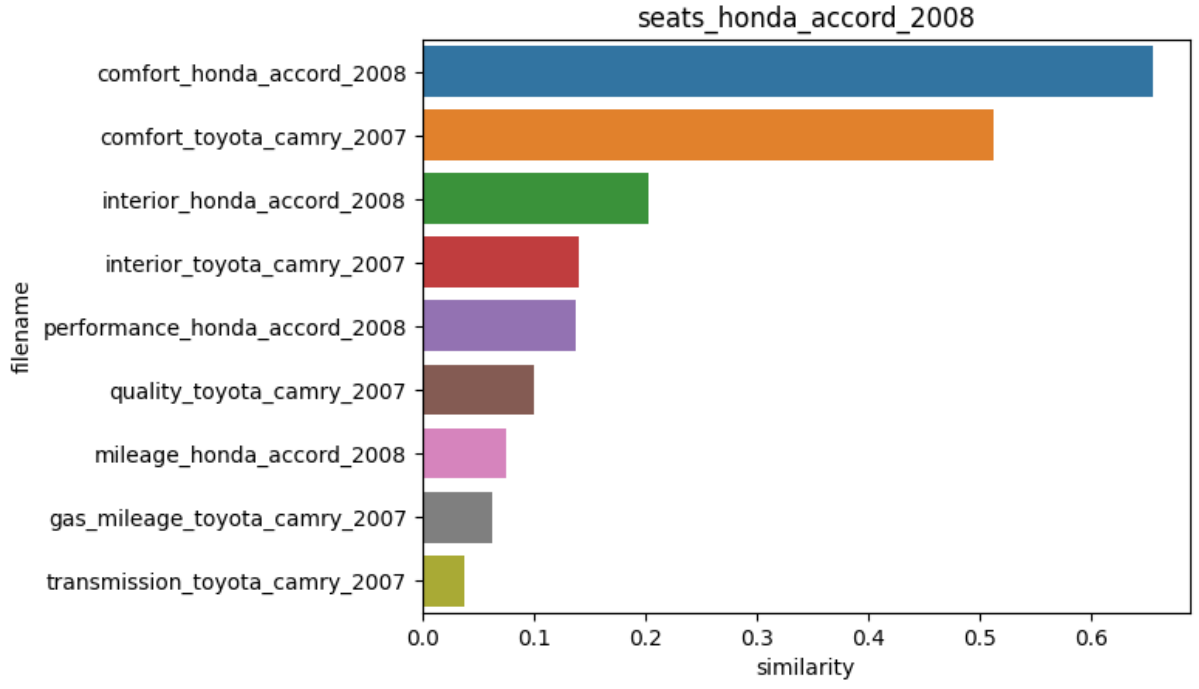
```
from sklearn.metrics.pairwise import cosine_similarity

# cluster_label=2인 데이터는 호텔로 군집화된 데이터임. DataFrame에서
hotel_indexes = document_df[document_df['cluster_label']==2].
print('호텔로 클러스터링 된 문서들의 DataFrame Index:', hotel_index

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

''' document_df에서 추출한 Index 객체를 feature_vect로 입력해 호텔
이를 이용하여 호텔로 군집화된 문서 중 첫 번째 문서와 다른 문서 간의 코사인
similarity_pair = cosine_similarity(feature_vect[hotel_indexe
print(similarity_pair)
```

```
### output ###
호텔로 클러스터링 된 문서들의 DataFrame Index: Int64Index([4, 11, 1
##### 비교 기준 문서명  seats_honda_accord_2008  와 타 문서 유사도##
[[1.         0.09955737 0.07510547 0.13825109 0.13971015 0.06
  0.20216909 0.65502034 0.51181434 0.03767558]]
```

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

# 첫 번째 문서와 타 문서간 유사도가 큰 순으로 정렬한 인덱스 추출하되 자기
```

```
sorted_index = similarity_pair.argsort()[:,::-1]
sorted_index = sorted_index[:, 1:]

# 유사도가 큰 순으로 hotel_indexes를 추출하여 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 유사도값을
hotel_1_sim_df = pd.DataFrame()
hotel_1_sim_df['filename'] = document_df.iloc[hotel_sorted_in
hotel_1_sim_df['similarity'] = hotel_1_sim_value

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

```
### output ###
Text(0.5, 1.0, 'seats_honda_accord_2008')
```



→ 첫 번째 문서인 샌프란시스코의 베스트 웨스턴 호텔 화장실 리뷰(Best Western Hotel Bathroom Review)인 bathroom_bestwestern_hotel_sfo와 가장 비슷한 문서는 room_holidaty_inn_london 으로 나타남. 약 0.514의 코사인 유사도

# 8.9 한글 텍스트 처리 - 네이버 영화 평점 감성 분석

### 한글 NLP 처리의 어려움

- 띄어쓰기
  : 띄어쓰기를 잘못하면 의미가 왜곡됨
- 다양한 조사
  : 어근 추출(Stemming/Lemmatization) 등의 전처리 시 제거하기 까다로움

### KoNLPy 소개

- 파이썬의 대표적인 한글 형태소 패키지
- 형태소 분석(Morphological analysis): 말뭉치를 형태소 어근 단위로 쪼개고 각 형태소에 품사 태깅(POS tagging)을 부착하는 작업

### 데이터 로딩

```
import pandas as pd

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

학습 데이터 세트의 0(부정)과 1(긍정)의 Label 값 비율 살펴보기

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

```
### output ###
0    75173
```

```
1    74827
Name: label, dtype: int64
```

→ 어느 한 쪽으로 치우치지 않고 균등한 분포

데이터 가공(Null → 공백으로 변환)

```
import re

train_df = train_df.fillna(' ')
# 정규 표현식을 이용해 숫자를 공백으로 변경(정규 표현식으로 \d는 숫자를 의
train_df['document'] = train_df['document'].apply( lambda 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 : r

# id 칼럼 삭제 수행
train_df.drop('id', axis=1, inplace=True)
test_df.drop('id', axis=1, inplace=True)
```

각 문장을 한글 형태소 분석을 통해 형태소 단어로 토큰화하기

```
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
```

사이킷런 TfidfVectorizer을 이용해 TF-IDF 피처 모델 생성

```
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_ra
tfidf_vect = TfidfVectorizer(tokenizer=tw_tokenizer, ngram_ra
tfidf_vect.fit(train_df['document'])
tfidf_matrix_train = tfidf_vect.transform(train_df['document'
```

로지스틱 회귀를 이용해 분류 기반의 감성 분석행

```
# 로지스틱 회귀를 이용해 감성 분석 분류 수행.
lg_clf = LogisticRegression(random_state=0, solver='liblinear

# 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, scori
grid_cv.fit(tfidf_matrix_train, train_df['label'])
print(grid_cv.best_params_, round(grid_cv.best_score_,4))
```

```
### output ###
Fitting 3 folds for each of 5 candidates, totalling 15 fits
{'C': 3.5} 0.8593
```

→ C가 3.5 일 때, 최고 0.8593의 정확도

테스트 세트를 이용해 최종 감성 분석 예측 수행

```
from sklearn.metrics import accuracy_score

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

# classifier는 GridSearchCV에서 최적 파라미터로 학습된 classifier를
best_estimator = grid_cv.best_estimator_
```

```
preds = best_estimator.predict(tfidf_matrix_test)

print('Logistic Regression 정확도: ', accuracy_score(test_df['
```

```
### output ###
Logistic Regression 정확도:  0.86172
```