💻Week15_예습과제_이재린

태그	예습과제
주차	15주차

Ch8 텍스트 분석

▼ 06. 토픽 모델링 - 20 뉴스 그룹

1. 토픽 모델링이란?

- 토픽 모델링 : 문서 집합에 숨어 있는 주제를 찾아내는 것
- 사람 vs 머신러닝 기반
    - 사람 : 더 함축적인 의미로 문장 요약
    - 머신러닝 기반 : 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출
- 토픽 모델링에 사용되는 LSA와 LDA(Latent Dirichlet Allocation ; 앞에 차원 축소 LDA와 다른 알고리즘)

2. LDA 적용해보기

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware',
'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space',
'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc',
'talk.religion.misc']
```

>> 20가지 주제를 가진 뉴스그룹의 데이터 중 모토사이클, 야구, 그래픽스, 윈도우, 중동, 기독교, 전자공학, 의학의 8개 주제를 추출하고 텍스트에 LDA 기반의 토픽 모델링 적용하

- LDA는 Count 기반의 벡터화만 사용하므로 Count 기반의 벡터화 변환하기 + max_features=1000으로 word 피처의 개수 제한하기 + ngram_range는 (1,2)로 설정하기

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

# 모토사이클, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 의학, 우주 주제를 추출
cats = ['rec.motorcycles', 'rec.sport.baseball', 'comp.graphics', 'comp.windows.x','talk.poli

# 위에서 cats 변수로 기재된 카터고리만 추출 >> categories에 cats 입력
news_df=fetch_20newsgroups(data_home=data_home,subset='all',remove=('headers','footers','quot

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

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

>> CountVectorizer 객체 변수인 feat_vect 모두 7862개의 문서가 1000개의 피처로 구성된 행렬 데이터

- n_components로 토픽개수 정하기
- LatentDirichletAllocation.fit 수행하기 : components_ 속성값하기 >> 개별 토픽별 각 word 피처가 얼마나 많이 그 토픽에 할당되었는지에 관한 수치(높으면 해당 word 피처는 그 토픽의 중심 word가 됨)

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

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

>> 8개의 토픽별로 1000개의 word 피처가 해당 토픽별로 연관도 값을 가지고 있다. ex) array의 0번째 row, 10번째 col에 있는 값은 topic #0에 대하여 피처 벡터화된 행렬에서 10번째 칼럼에 해당하는 피처가 연관되는 수치 값을 가짐.

- display_topics() 함수를 만들어 각 토픽별로 연관도가 높은 순으로 Word 나열하기

```
def display_topics(model,feature_names,no_top_words):
    for topic_index, topic in enumerate(model.components_):
        print('Topic #',topic_index)
        # components_ array에서 가장 값이 큰 순으로 정렬했을 때, 그 값의 array index를 반환
        topic_word_indexes=topic.argsort()[::-1]
        topic_indexes=topic_word_indexes[:no_top_words]
        # top_indexes대상인 index별로 feature_names에 해당하는 word feature 추출 후 join으로 concat
        feature_concat=' '.join([feature_names[i] for i in topic_indexes])
        print(feature_concat)

# CountVectorizer객체 내의 전체 word들의 명칭을 get_features_names( )를 통해 추출
feature_names=count_vect.get_feature_names_out()

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

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

*최신 Scikit-learn에서는 `get_feature_names`가 `get_feature_names_out`으로 대체됨...

>> 주제별로 주제어가 명확한 것도 아닌것도 있음.

▼ 07. 문서 군집화 소개와 실습

문서 군집화 개념

- 문서 군집화 : 비슷한 텍스트 구성의 문서를 군집화(Clustering)하는 것
- 텍스트 분류 기반의 문서 분류 vs 문서 군집화 : 사전에 결정 카테고리 값을 가진 학습 데이터 세트가 필요함(지도학습) / 학습 데이터 세트가 필요 없음(비지도학습)

Opinion Review 데이터 세트를 이용한 문서 군집화 수행하기

- 각 tripadvisor(Hotel), Edmunds(Car), Amazon에서 가져온 리뷰 문서

1. 여러개의 파일을 한개씩 읽어 df로 로딩해 데이터 처리 진행 >> 해당 dir의 모든 파일을 for로 반복해서 개별 파일명을 리스트에 추가 & 개별 파일은 df로 읽어 문자열로 반환

```python
import pandas as pd
import glob,os

# 디렉터리 재설정
path=r'/Users/bluecloud/Documents/대학/유런/데이터셋/topics'
#path로 지정한 dir 밑에 있는 모든 .data 파일의 파일명을 리스트업
all_files=glob.glob(os.path.join(path,"*.data"))
filename_list=[]
opinion_text=[]

# 개별 파일의 파일명은 filename_list로 취합
# 개별 파일의 파일 내용은 DataFrame 로딩 후 다시 string 으로 변환해 opinion_text list로 취합
for file_ in all_files:
    # 개별 파일 읽어서 df생성
    df=pd.read_table(file_,index_col=None,header=0,encoding='latin1')
    # 절대 경로에서 파일명 추출
    filename_ = os.path.splitext(os.path.basename(file_))[0]
    filename=filename_.split('.')[0]
    # 파일명 list와 파일 내용 list에 파일명과 파일 내용을 추가
    filename_list.append(filename)
    opinion_text.append(df.to_string())

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

	filename	opinion_text
0	battery-life_ipod_nano_8gb	...
1	gas_mileage_toyota_camry_2007	...
2	room_holiday_inn_london	...
3	location_holiday_inn_london	...
4	staff_bestwestern_hotel_sfo	...

>> 참고) python의 os 모듈의 함수 : 여러 개의 경로 구성 요소를 결합하여 하나의 경로를 형성

os.path.join은 Python의 os 모듈에 있는 함수로, 여러 개의 경로 구성 요소를 결합하여 하나의 경로를 형성하는 데 사용됩니다. 이 함수는 운영 체제별로 다른 파일 시스템의 경로 구분자를 자동으로 처리하며, 경로를 구성하는 가장 안전하고 효율적인 방법을 제공합니다. os.path.join의 주요 특징은 다음과 같습니다:

1. **운영 체제별 경로 구분자 처리**: os.path.join은 Windows에서는 백슬래시(\)를, UNIX 기반 시스템(예: Linux, macOS)에서는 슬래시(/)를 사용하여 경로를 결합합니다. 이를 통해 코드가 다양한 플랫폼에서 문제없이 실행될 수 있도록 합니다.
2. **다중 인자 지원**: 이 함수는 두 개 이상의 경로 구성 요소를 인자로 받을 수 있으며, 이들을 순서대로 결합합니다.
3. **절대 경로 처리**: 인자 중 하나가 절대 경로인 경우, 그 이전의 모든 경로 구성 요소는 무시되고, 절대 경로부터 새로운 경로가 구성됩니다.
4. **빈 문자열 처리**: 인자 중 하나가 빈 문자열이면, 무시되고 다음 경로 구성 요소로 넘어갑니다.

2. 문서를 TF-IDF 형태로 벡터화하기 >> LemNormalize() 함수 이용하기

```python
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download('punkt')
nltk.download('wordnet')
import string

remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
lemmar = WordNetLemmatizer()

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

# TfidfVectorizer 객체 생성 시 tokenizer인자로 해당 함수를 설정하여 lemmatization 적용
```

```
# 입력으로 문장을 받아서 stop words 제거-> 소문자 변환 -> 단어 토큰화 -> lemmatization 어근 변환
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))


tfidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english',ngram_range=(1,2),

#opinion_text 컬럼값으로 feature vectorization 수행
feature_vect = tfidf_vect.fit_transform(document_df['opinion_text'])
```

3. 피처 벡터화 행렬 데이터에 대하여 군집화 수행하기(K-means 적용)

   1) 5개의 중심 기반으로 확인해보기 > 군집의 Label 값과 중심별로 할당된 데이터 세트의 좌표값 구하기

```
from sklearn.cluster import KMeans

#5개 집합으로 군집화 수행
km_cluster=KMeans(n_clusters=5,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
document_df.head()
```

	filename	opinion_text	cluster_label
0	battery-life_ipod_nano_8gb	...	1
1	gas_mileage_toyota_camry_2007	...	4
2	room_holiday_inn_london	...	3
3	location_holiday_inn_london	...	0
4	staff_bestwestern_hotel_sfo	...	3

2) cluster_label로 군집화 결과 확인해보기

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

	filename	opinion_text	cluster_label
17	food_holiday_inn_london	...	0
32	food_swissotel_chicago	...	0
3	location_holiday_inn_london	...	0
41	price_amazon_kindle	...	0
28	price_holiday_inn_london	...	0
16	service_bestwestern_hotel_sfo	...	0
27	service_holiday_inn_london	...	0
13	service_swissotel_hotel_chicago	...	0

>> label=0의 경우 호텔 리뷰로 이루어져있다

>> but 5개의 군집을 다 살펴보면 세분화된 경향이 있다 >> 군집의 개수를 줄여야한다

3) 3개 그룹으로 군집화하기

```
from sklearn.cluster import KMeans

# 3개의 집합으로 군집화
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_

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

	filename	opinion_text	cluster_label
50	parking_bestwestern_hotel_sfo	...	0
27	service_holiday_inn_london	...	0
28	price_holiday_inn_london	...	0
30	rooms_swissotel_chicago	...	0
20	staff_swissotel_chicago	...	0
31	bathroom_bestwestern_hotel_sfo	...	0
34	directions_garmin_nuvi_255W_gps	...	1
33	accuracy_garmin_nuvi_255W_gps	...	1
40	speed_windows7	...	1
41	price_amazon_kindle	...	1
44	fonts_amazon_kindle	...	1
48	display_garmin_nuvi_255W_gps	...	1
38	navigation_amazon_kindle	...	1
36	eyesight-issues_amazon_kindle	...	1
18	comfort_honda_accord_2008	...	2
42	quality_toyota_camry_2007	...	2
43	comfort_toyota_camry_2007	...	2
29	seats_honda_accord_2008	...	2
45	interior_honda_accord_2008	...	2
22	interior_toyota_camry_2007	...	2

>> 군집 0은 호텔로, 군집 1은 전자기기로, 군집 2는 자동차로 잘 구성됨.

군집별 핵심 단어 추출하기_각 군집을 구성하는 핵심 단어 확인해보기

- clusters_centers_속성으로 각 군집을 구성하는 단어 피처가 Centroid 기준으로 얼마나 가깝게 위치하였는지 확인 >> 배열을 제공 (행은 개별 군집, 열은 개별 피처, 배열 내 값은 군집 내의 상대 위치 숫자값)

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

```
cluster_centers shape : (3, 4611)
[[0.         0.00099499 0.00174637 ... 0.         0.00183397 0.00144581]
 [0.01005322 0.         0.         ... 0.00706287 0.         0.        ]
 [0.         0.00092551 0.         ... 0.         0.         0.        ]]
```

>> 군집 3개와 word 피처는 4611개로 구성 / 배열의 값이 1에 가까울수록 중심과 가까운 값을 의미함

- 군집별 핵심 단어를 찾아보기 : clusters_centers_속성은 ndarray >> argsort[::-1]로 큰 값을 가진 배열 내 위치 인덱스 값을 반환 >> 위치 인덱스로 핵심 단어 피처의 이름 출력하기

```
# 군집별 top n 핵심단어, 그 단어의 중심 위치 상대값, 대상 파일명들을 반환함.
def get_cluster_details(cluster_model, cluster_data, feature_names, clusters_num, top_n_featu
    cluster_details = {}

    # cluster_centers array 의 값이 큰 순으로 정렬된 index 값을 반환
    # 군집 중심점(centroid)별 할당된 word 피처들의 거리값이 큰 순으로 값을 구하기 위함.
    centroid_feature_ordered_ind = cluster_model.cluster_centers_.argsort()[:,::-1]

    #개별 군집별로 iteration하면서 핵심단어, 그 단어의 중심 위치 상대값, 대상 파일명 입력
    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 index 를 이용하여 top n 피처 단어를 구함.
        top_feature_indexes = centroid_feature_ordered_ind[cluster_num, :top_n_features]
        top_features = [ feature_names[ind] for ind in top_feature_indexes ]

        # top_feature_indexes를 이용해 해당 피처 단어의 중심 위치 상댓값 구함
        top_feature_values = cluster_model.cluster_centers_[cluster_num, top_feature_indexes]
```

```
        # cluster_details 딕셔너리 객체에 개별 군집별 핵심 단어와 중심위치 상대값, 그리고 해당 파일명 입력
        cluster_details[cluster_num]['top_features'] = top_features
        cluster_details[cluster_num]['top_features_value'] = top_feature_values
        filenames = cluster_data[cluster_data['cluster_label'] == cluster_num]['filename']
        filenames = filenames.values.tolist()
        cluster_details[cluster_num]['filenames'] = filenames

    return cluster_details
```

>> dict을 원소로 가지는 리스트 cluster_details를 반환

```
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('==================================================')
```

>> 별도의 함수를 만들어 cluster_details를 깔끔하게 반환

```
feature_names=tfidf_vect.get_feature_names_out()
cluster_details=get_cluster_details(cluster_model=km_cluster,cluster_data=document_df,feature_name
print_cluster_details(cluster_details)
```

```
####### Cluster 0
Top features: ['room', 'hotel', 'service', 'staff', 'food', 'location', 'bathroom', 'clean', 'pri
ce', 'parking']
Reviews 파일명 : ['room_holiday_inn_london', 'location_holiday_inn_london', 'staff_bestwestern_hote
l_sfo', 'service_swissotel_hotel_chicago', 'service_bestwestern_hotel_sfo', 'food_holiday_inn_lon
don', 'staff_swissotel_chicago']
==================================================
####### Cluster 1
Top features: ['screen', 'battery', 'keyboard', 'battery life', 'life', 'kindle', 'direction', 'v
ideo', 'size', 'voice']
Reviews 파일명 : ['battery-life_ipod_nano_8gb', 'voice_garmin_nuvi_255W_gps', 'speed_garmin_nuvi_25
5W_gps', 'size_asus_netbook_1005ha', 'screen_garmin_nuvi_255W_gps', 'battery-life_amazon_kindle',
'satellite_garmin_nuvi_255W_gps']
==================================================
####### Cluster 2
Top features: ['interior', 'seat', 'mileage', 'comfortable', 'gas', 'gas mileage', 'transmissio
n', 'car', 'performance', 'quality']
Reviews 파일명 : ['gas_mileage_toyota_camry_2007', 'comfort_honda_accord_2008', 'interior_toyota_ca
mry_2007', 'transmission_toyota_camry_2007', 'seats_honda_accord_2008', 'mileage_honda_accord_200
8', 'quality_toyota_camry_2007']
==================================================
```

>> cluster 0은 호텔 리뷰 군집 : room, hotel, service, location 등 방과 서비스 등이 핵심 단어로 군집화됨

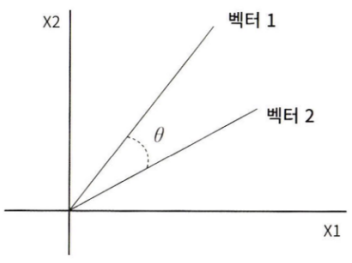
>> cluster 1은 전자제품 리뷰 군집 : screen, battery, life 등 화면과 배터리 수명 등이 핵심 단어로 군집화됨

>> cluster 2은 자동차 리뷰 군집 : interior, seat, mileage 등 실내 인테리어, 좌석 , 연료 효율 등이 핵심 단어로 군집화됨

▼ 08. 문서 유사도

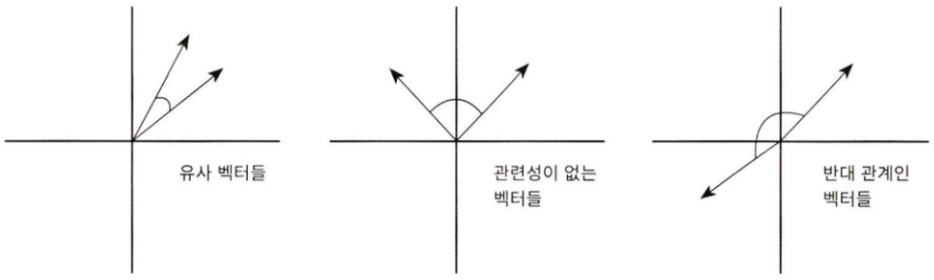
문서 유사도 측저 방법 - 코사인 유사도

- 코사인 유사도 : 문서와 문서 간의 유사도 비교 >> 벡터와 벡터 간의 유사도를 비교할 때 크기보다는 상호 방향성이 얼마나 유사한지에 기반 >> 사잇각 사용



두 벡터 사잇각

- 사잇각에 따른 상호 관계는 다음과 같다.



- 두 벡터 사이의 코사인값 구하기

$$A * B = \| A \| \| B \| cos\theta$$

- 유사도 코사인값 = 두 벡터의 내적을 총 벡터의 크기합으로 나누기

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

- 기존 유사도 지표

  (1) 문서를 피처 벡터화하면 희소 행렬(차원 많음)이 되기 쉬움 >> 이를 바탕으로 한 유사도 지표(ex) 유클리드 거리 기반 지표)는 정확도 떨어지기 쉬움

  (2) 문서가 긴 경우 단어 빈도수가 많음

>> 따라서 간단한 문서에 대해 서로 간의 문서 유사도를 코사인 유사도 기반으로 구하기

cos_similarity()

1. 코사인 유사도 구하는 함수 작성

```
import numpy as np
# 두 벡터의 코사인 유사도
def cos_similarity(v1, v2):
    dot_product = v1 @ v2
    ab_norm = np.sqrt(sum(np.square(v1))) * np.sqrt(sum(np.square(v2)))
    similarity = dot_product / ab_norm

    return similarity
```

2. 문서를 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 the rabbit hole goes']

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

```
(3, 18)
```

>> 반환된 행렬은 희소행렬

3. 희소행렬을 1의 인자 array로 만들기 위해 밀집행렬로 변환 후 배열로 변환

```
# TFidfVectorizer로 transform()한 결과는 Sparse Matrix이므로 Dense Matrix로 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,)

# 코사인 유사도 추출
similarity_simple=cos_similarity(vect1,vect2)
print('문장 1, 문장 2 코사인 유사도 : {0:.3f}'.format(similarity_simple))
```

```
문장 1, 문장 2 코사인 유사도 : 0.402
```

cosine_similarity()

: 희소행렬, 밀집행렬 모두 가능 & 행렬 또는 배열 모두 가능

```
from sklearn.metrics.pairwise import cosine_similarity

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

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

>> 1은 자신과의 유사도, 1&2의 유사도, 1&3의 유사도

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

```
[[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

- 앞 절의 문서 군집화에서 사용한 Opinion Review 데이터 세트로 문서 간의 유사도 측정하기
- 호텔을 주제로 군집화된 문서와 다른 문서간의 유사도 알아보기

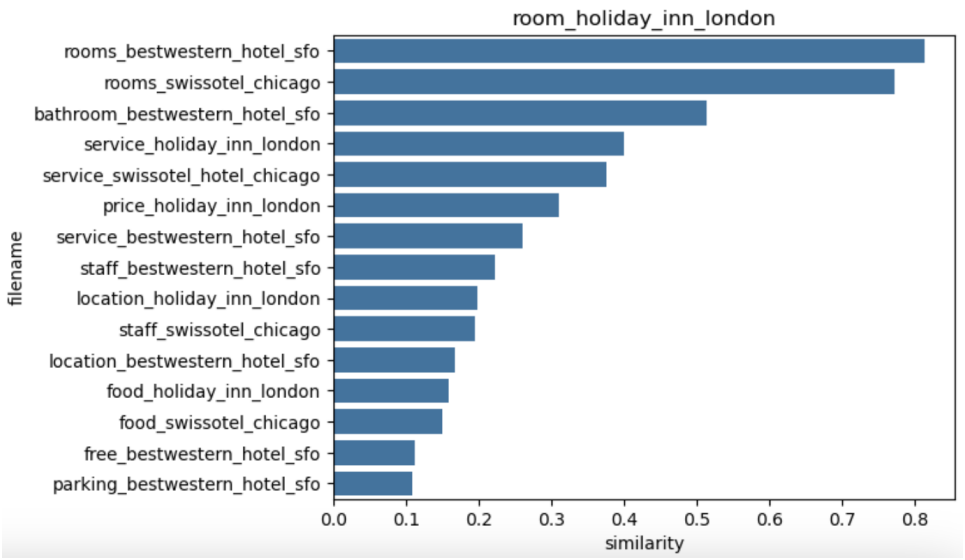
  1)호텔을 주제로 군집화된 데이터 추출해 이에 해당하는 TfidfVectorizer데이터 추출하기(TD-IDF 수행하지 않음)

```
from sklearn.metrics.pairwise import cosine_similarity

# cluster_label=0이 호텔 군집화 데이터임, 인덱스 추출
hotel_indexes=document_df[document_df['cluster_label']==0].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 문서간의 코사인 유사도 측정
'''
similarity_pair = cosine_similarity(feature_vect[hotel_indexes[0]], feature_vect[hotel_indexes])
print(similarity_pair)
```

```
호텔로 군집화 된 문서들의 DataFrame Index :  Index([2, 3, 4, 13, 16, 17, 20, 27, 28, 30, 31, 32, 39, 4
6, 49, 50], dtype='int64')
##### 비교 기준 문서명 room_holiday_inn_london 와 타 문서 유사도 #####
[[1.         0.19917258 0.22235374 0.37631406 0.26026786 0.15836737
  0.19544761 0.40020673 0.31124876 0.77312013 0.51442299 0.15026112
  0.16717527 0.81484367 0.11154184 0.10831277]]
```

>> 교재와는 다르게 cluster_label==0이 호텔 군집화 데이터

2)첫 번째 문서와 다른 문서 간의 유사도가 높은 순으로 정렬해 시각화하기(ndarray를 반환하므로 reshape로 차원 변경하기)



>> rooms_bestwestern_hotel_sfo가 가장 큰 유사도를 보임

▼ 09. 한글 텍스트 처리 - 네이버 영화 평점 감성 분석

한글 NLP 처리의 어려움

- 원인 : 띄어쓰기와 다양한 조사

KoNLPy 소개

- KoNLPy란? 파이썬의 대표적인 한글 형태소(단어로서 의미를 가지는 최소 단위) 패키지, Python Wrapper 기반으로 재작성됨
- 형태소 분석 : 말뭉치를 형태소 어근 단위로 쪼개고 각 형태소에 품사 태깅을 부착하는 작업
- KoNLPy 설치 : 맥북의 경우 터미널에서 가상환경 실행 > pip로 JPype1 install >> KoNLPy 설치 >> 가상환경을 통해 jdk 설치

```
~ % cd
~ % conda activate
 % pip install JPype1
```

```
Installing collected packages: konlpy
Successfully installed konlpy-0.6.0
(base) bluecloud@ijaelin-ui-MacBookAir ~ % conda install -c conda-forge openjdk
```

- 주피터 노트북에서 에러 안남 확인

```
[79]: from konlpy.tag import Okt
      tw_tag = Okt()
```

데이터 로딩

- 네이버 영화 평점 데이터 다운받기

1. train_df의 null 값을 공백으로 변환

```
import re

train_df = train_df.fillna(' ')
#정규 표현식을 이용해 숫자를 공백을 변경
train_df['document'] = train_df['document'].apply(lambda x : re.sub(r"\df+", " ", x))

#테스트 데이터 세트를 로딩하고 동일하게 Null 및 숫자를 공백으로 반환
test_df = pd.read_csv('/Users/bluecloud/Documents/대학/유런/데이터셋/navermovie/ratings_test.txt', sep
test_df = test_df.fillna(' ')
train_df['document'] = train_df['document'].apply(lambda x : re.sub(r"\d+", " ", x))

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

2. TD-IDF 방식으로 단어를 벡터화 : 문장을 형태소 단어로 토큰화(Twitter 클래스 이용 : 입력 인자로 들어온 문장을 형태소 단어 형태로 토큰화해 list 객체로 반환하기)
    - tokenizer를 변행해 tw_tokenizer로 만들기 >> 버전 이슈로 Okt 클래스로 대체

```
from konlpy.tag import Okt
# 사이킷런에서는 twitter가 없어지고 okt로 바뀜
okt = Okt()
def tw_tokenizer(text):
    #입력인자로 들어온 텍스트를 형태소 단어로 토큰화해 리스트 형태로 반환
    tokens_ko = okt.morphs(text)
    return tokens_ko
```

    - 위의 함수로 TF-IDF 피처 모델을 생성하기
3. 로짓스틱 회귀로 분류 기반 감성 분석 수행
- 하이퍼 파라미터 C의 최적화를 위해 GridSearchCV이용하기

>> C가 3.5일 때 최고 0.8593의 정확도를 보임

4. 테스트 세트로 최종 감성 분석 예측 수행
- 앞의 문제처럼 테스트 세트로 예측할 때는 학습할 때 적용한 TfidVectorizer 그대로 사용하기