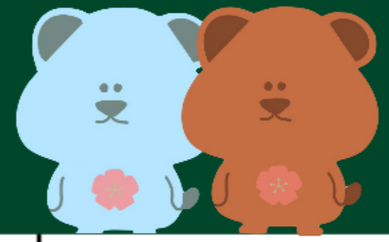
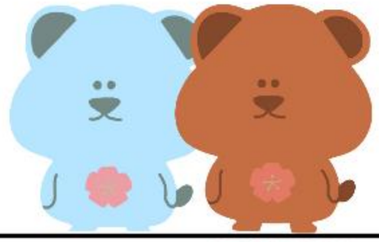


13주차 발표

DA팀 박지운 오연재 최하경

EWHA EURON

목차



EWHA EURON

01. 문서 군집화

EWHA EURON

1.1 문서 군집화

- 비슷한 텍스트 구성의 문서를 군집화하는 것이다.
동일한 군집에 속하는 문서를 같은 카테고리 소속으로 분류,
텍스트 분류 기반의 문서 분류와 유사하다.

- 단어 발생 빈도수에 기반하는 BOW(Bag Of Words) 방식을 이용해
Feature(문서들을 이루고 있는 단어들)를 벡터화시키거나 단어들간의 의미 관계 즉,
단어 벡터들간의 방향을 고려해 Word embedding을 통해 벡터화 시킨다.

텍스트 기반 분류	문서 군집화
사전에 결정 카테고리 값을 가진 학습 데이터 필요	사전에 결정 카테고리 값을 가진 학습 데이터 필요 없는 비지도 학습 기반 동작

1.2 Opinion Reviw를 이용한 문서 군집화

- 여러 개의파일을 한 개의 dataframe으로 로드하여 파일명별로 어떤 리뷰를 담고 있는지 살피기.
- (*절대 경로 디렉터리를 먼저 지정하고 이 위치에서 데이터를 로딩한다.*)

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

```
# 아래는 제 컴퓨터에서 압축 파일을 풀어 놓은 디렉토리이니, 여러분의 디렉토리를 설정해 주십시오
path = r'C:₩Users₩kyeongmoo₩Desktop₩오연재₩OpinosisDataset1.0₩OpinosisDataset1.0₩topics'
# path로 지정한 디렉토리 밑에 있는 모든 .data 파일들의 파일명을 리스트로 취합
```

- 해당 디렉터리 내의 모든 파일에 대해 각각 for반복문을 이용하여 개별 파일명을 파일명 리스트에 추가, 개별 파일은 dataframe 으로 읽은 후 다시 문자열로 반환, 파일 내용 리스트에 추가한다.

```
all_files = glob.glob(os.path.join(path, "*.data"))
filename_list = []
opinion_text = []

# 개별 파일들의 파일명은 filename_list 리스트로 취합,
# 개별 파일들의 파일내용은 DataFrame로딩 후 다시 string으로 변환하여 opinion_text 리스트로 취합
for file_ in all_files:
    # 개별 파일을 읽어서 DataFrame으로 생성
    df = pd.read_table(file_,index_col=None, header=0,encoding='latin1')

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

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

해당 디렉터리 밑에 있는 모든 파일

문자열로 반환

EWHA EURON

1.2 Opinion Reviw를 이용한 문서 군집화

- 파일명 리스트와 파일 내용 리스트를 이용하여 새롭게 파일명과 파일 내용을 칼럼으로 가지는 dataframe을 생성한다.

```
# 파일명 리스트와 파일내용 리스트를  DataFrame으로 생성
document_df = pd.DataFrame({'filename':filename_list, 'opinion_text':opinion_text})
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	...

- 해당 디렉터리 내문서를 TF-IDF형태로 피처 벡터화를 진행 LemNormalize를 이용-> 개별 문서 텍스트에 대해 TF-IDF변환된 피처 벡터화된 행렬을 구할 수 있다.

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

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

def LemTokens(tokens):
    return [lemmar.lemmatize(token) for token in tokens]

def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

1.2 Opinion Reviw를 이용한 문서 군집화

- 군집별 핵심단어 추출하기….
- 각 군집을 구성하는 단어 피처가 군집의 중심(centroid)을 기준으로 얼마나 가깝게 위치하는지 cluster_centers_라는 속성으로 제공한다. 배열 값의 형태이고 행: 개별 군집, 열: 개별 피쳐
- 각 배열 내의 값: 개별 군집 내의 상대위치를 숫자 값으로 표현. (0에서 1의 값으로 나타내며 1에 가까울수록 중심과 가까운 것이다.)

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

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

군집이 3개, word 피처가 4611개라는 뜻

1.2 Opinion Reviw를 이용한 문서 군집화

```
#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_features=10):
    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].tolist()

        # 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
```

- 핵심 단어 피처의 이름을 출력하기 위해 argsort를 이용한다. (배열 내 값이 큰 순으로 정렬된 위치 인덱스 값으로 반환하고 큰 값을 가진 배열 내 위치 인덱스 값을 반환하는 것이다.)

- Get_cluster_details()는 배열 내에서 가장 값이 큰 데이터의 위치 인덱스를 추출, 해당 인덱스를 이용해 핵심 단어 이름, 상대 위치 값을 추출->cluster_detail라는 변수에 기록하여 반환.
- Cluster_details에는 개별 군집번호, 핵심단어, 핵심단어 중심 위치 상댓값, 파일명 속성 값 정보,, 잘 보이게 하기 위해 print_cluster_details()를 이용

1.2 Opinion Reviw를 이용한 문서 군집화

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

```
feature_names = tfidf_vect.get_feature_names()

cluster_details = get_cluster_details(cluster_model=km_cluster, cluster_data=document_df,₩
                                      feature_names=feature_names, clusters_num=3, top_n_features=10 )
print_cluster_details(cluster_details)
```

```
####### Cluster 0
Top features: ['screen', 'battery', 'keyboard', 'battery life', 'life', 'kindle', 'direction', 'vic
Reviews 파일명 : ['accuracy_garmin_nuvi_255W_gps', 'battery-life_amazon_kindle', 'battery-life_ipoc
05ha', 'buttons_amazon_kindle', 'directions_garmin_nuvi_255W_gps', 'display_garmin_nuvi_255W_gps']
==================================================
####### Cluster 1
Top features: ['interior', 'seat', 'mileage', 'comfortable', 'gas', 'gas mileage', 'transmission',
Reviews 파일명 : ['comfort_honda_accord_2008', 'comfort_toyota_camry_2007', 'gas_mileage_toyota_cam
8', 'interior_toyota_camry_2007', 'mileage_honda_accord_2008', 'performance_honda_accord_2008']
==================================================
####### Cluster 2
Top features: ['room', 'hotel', 'service', 'staff', 'food', 'location', 'bathroom', 'clean', 'price
Reviews 파일명 : ['bathroom_bestwestern_hotel_sfo', 'food_holiday_inn_london', 'food_swissotel_chic
'location_bestwestern_hotel_sfo', 'location_holiday_inn_london', 'parking_bestwestern_hotel_sfo']
==================================================
```

- Cluster_details에는 개별 군집번호, 핵심단어, 핵심단어 중심 위치 상댓값, 파일명 속성 값 정보,, 잘 보이게 하기 위해 print_cluster_details()를 이용

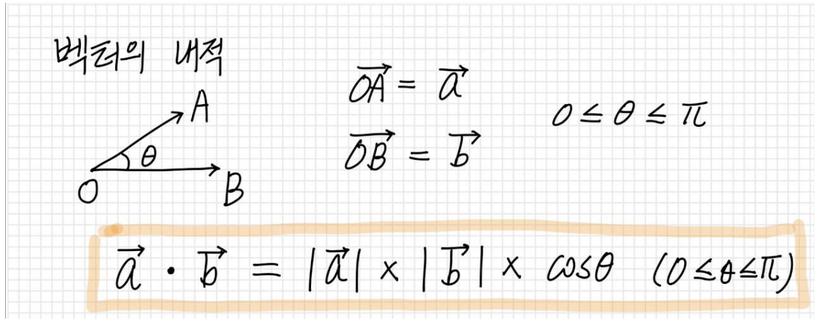
Cluster #0: 포터블 전자제품 리뷰, screen, battery……
Cluster #1: 자동차 리뷰, interior, seat……..
Cluster #2: 호텔 리뷰, room, service……..

02. 문서 유사도

EWHA EURON

2.1 코사인 유사도

- 코사인 유사도: 문서 유사도를 측정, 문서와 문서 간의 유사도를 비교할 때 사용한다.
- 크기보다는 방향에 집중하는 것에 포커스를 맞춘다.



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

Cosine 0° : 두 벡터(단어)간에 매우 유사한 관계를 지닌다. Cosine Similarity값은 1이 된다.
Cosine 90° : 두 벡터간에 관계가 없음을 의미한다. Cosine Similarity값은 0이 된다.
Cosine 180° : 두 벡터간에 반대관계를 의미한다. Cosine Similarity값은 -1이 된다.

하지만 텍스트 즉, 단어들을 Feature화 시킬 때 음수값이 나올 수 없으므로 Cosine 180° 인 경우는 존재하지 않는다. 단어 벡터들간의 유사도는 0 ~ 1사이의 값으로 나오게 된다.

- 코사인 유사도는 두 벡터의 내적을 총 벡터 크기의 합으로 나눈 것이다. (내적 결과를 총 벡터 크기로 정규화 L2 norm한 것이다.)

문서 유사도 비교에 가장 많이 사용되는 이유: 문서를 피처 벡터화 변환하면 차원이 매우 많은 희소 행렬 될 가능성 높고 이러면 정확도 떨어진다. 문서가 매우 길 경우 단어수가 많아지는데 단어 빈도수에만 기반하면 제대로된 분석하지 못한다.

2.2 사이킷런 코사인 유사도

- 사이킷 런의 sklearn.metrics.pairwise.cosine_similarity 를 이용.
- Cosine_similarity는 두 개의 입력 파라미터를 받는다(희소 행렬, 밀집 행렬 모두 가능하고 행렬, 배열 보두 가능, 별도의 변환 작업 필요 없음)
- 첫번째 파라미터: 비교 기준이 되는 문서, 두번째 파라미터: 비교되는 분서

```
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]]
```

첫 번째 문서 자기 자신에 대한 유사도: 1 , 1,2 문서와의 유사도: 0.402 , 1,3 문서와의 유사도: 0.404

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_simple_pair = cosine_similarity(feature_vect_simple[0] , feature_vect_simple[1:])
print(similarity_simple_pair)
```

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

자기 자신에 대한 유사도 제거

```
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)
```

Paris로 한번에 표시

2.3 문서 유사도 시각화

- 데이터 세트를 새롭게 dataframe으로 로드하고 문서 군집화를 적용.
- 문서를 피처 벡터화에 변환하면서 문서 내 단어에 출현 빈도와 같은 값을 부여하여 각 문서가 단어 피처의 값으로 벡터화 된다. 이를 cosine_similarity( )를 이용해 유사도 확인.

```
from sklearn.metrics.pairwise import cosine_similarity

# cluster_label=1인 데이터는 호텔로 클러스터링된 데이터임. DataFrame에서 해당 Index를 추출
hotel_indexes = document_df[document_df['cluster_label']==1].index
print('호텔로 클러스터링 된 문서들의 DataFrame Index:', hotel_indexes)

# 호텔로 클러스터링된 데이터 중 첫번째 문서를 추출하여 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 입력하여 호텔 클러스터링된 feature_vect 추출
이를 이용하여 호텔로 클러스터링된 문서 중 첫번째 문서와 다른 문서간의 코사인 유사도 측정.'''
similarity_pair = cosine_similarity(feature_vect[hotel_indexes[0]] , feature_vect[hotel_indexes])
print(similarity_pair)
```

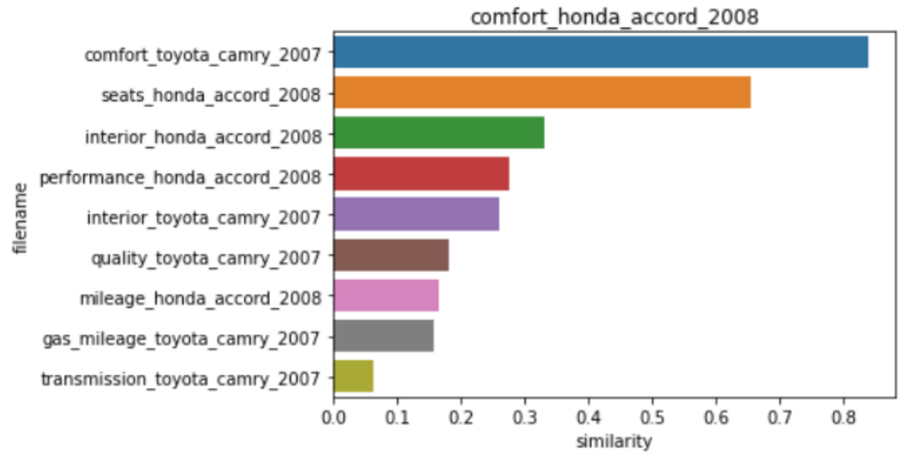
```
호텔로 클러스터링 된 문서들의 DataFrame Index: Int64Index([6, 7, 16, 17, 18, 22, 25, 29, 37, 47], dtype='int64')
##### 비교 기준 문서명  comfort_honda_accord_2008  와 타 문서 유사도######
[[1.         0.83969704 0.15655631 0.33044002 0.25981841 0.16544257
  0.27569738 0.18050974 0.65502034 0.06229873]]
```

- 호텔 데이터 먼저 추출(호텔 데이터로 안나옴..)
- 이 데이터에 해당하는 TfidVectorizer의 데이터를 추출.
- datarame객체 변수인 document_df에서 호텔 추출, 그대로 이용하여 피처벡터 추출.

2.3 문서 유사도 시각화

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

# argsort()를 이용하여 앞예제의 첫번째 문서와 타 문서간 유사도가 큰 순으로 정렬한 인덱스 반환하되 자기 자신은 제외.
sorted_index = similarity_pair.argsort()[:,::-1]
sorted_index = sorted_index[:, 1:]

# 유사도가 큰 순으로 hotel_indexes를 추출하여 재 정렬.
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 Index와 유사도값을 이용하여 파일명과 유사도값을 Seaborn 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)
```

```
Text(0.5, 1.0, 'comfort_honda_accord_2008')
```



- 첫번째 문자와 다른 문서 간에 유사도가 높은 순으로 정렬

- Cosine_similarity( )는 쌍 형태의 ndarray를 반환, 판다스 인덱스로 이용하기 위해 reshape(-1)로 차원을 변경.

Comfort honda accord와 가장 유사한 문서: comfort Toyota camry 2007

Contents based..사용자가 특정한 아이템을 매우 선호할 때 그 아이템과 비슷한 콘텐츠를 가진 다른 아이템을 추천. 영화 추천, 노래 추천…

03. 한글 텍스트 처리

EWHA EURON

3.1 한글 nlp처리

- 한글 nlp 처리의 어려움: 띄어쓰기(의미 왜곡 가능), 다양한 조사

cf) 영어는 띄어쓰기 잘못하면 그냥 없는 단어로 인식, 영어의 띄어쓰기는 매우 명확하여 대부분 명확하게 띄어쓰기를 한다. 하지만 한글을 띄어쓰기 많이들 어려워한다.

- 조사는 어근 추출 등의 전처리 하기가 까다롭다. “너희 집은 어디 있니?”에서 은 이 말하는 것이 조사인지, 금속 은인지,,,,

- **KoNLPy**소개: 파이썬의 대표적인 한글 형태소 패키지 이다.
- 기존의 씨언어와 자바로 만들어진 한글 형태소 엔진을 파이썬 래퍼 기반으로 재작성.
- 꼬꼬마, 한나눔, komoran, mecab, twitter와 같이 5개 형태소 모두 사용 가능.

형태소 분석 참고: https://velog.io/@metterian/%ED%95%9C%EA%B5%AD%EC%96%B4-%ED%98%95%ED%83%9C%EC%86%8C-%EB%B6%84%EC%84%9D%EA%B8%B0POS-%EB%B6%84%EC%84%9D-3%ED%8E%B8.-%ED%98%95%ED%83%9C%EC%86%8C-%EB%B6%84%EC%84%9D%EA%B8%B0-%EB%B9%84%EA%B5%90

3.2 KoNLPY

- **KoNLPY설치(교재 참고)**
- 1) 아나콘다 프롬프트에서 Jpype모듈 설치
- 2) https://www.lfd.uci.edu/~gohlke/pythonlibs/#jpype 에서 버전 맞춰서 Jpype설치
- 3) 2) 완료되면 압출 풀어서 아나콘다 프롬프트 base위치로 이동하여 저장
- 4) 아나콘다 프롬프트에서 pip이용하여 Jpype1을 설치
- 5) JAVA_HOME 환경변수 설정
- (환경변수로 등록한다는 의미는 내가 컴퓨터에 있는 폴더 중 어느 위치에 있어도 환경변수로 등록한 경로에 있는 파일을 바로 불러올 수 있다는 것)

- **Mecap 사용자 사전**
- 기본적으로 mecab은 제공하는 사전 외에 , 사용자가 직접 입력해서 형태소를 입력해서 단어를 인식할 수 있도록 하는 '사용자 사전'을 제공
- 기존 사전에 다양한 단어들이 있지만, 생각보다 원하는 대로 검색이 안되는 경우가 있는데, 이 때 사용자 사전을 사용하면 mecab을 더 다양하게 활용할 수 있다.

참고: https://tape22.tistory.com/6

3.2 Mecab

- Visual studio tool을 미리 설치하기.....

```
$ mecab -d /usr/local/lib/mecab/dic/mecab-ko-dic
mecab-ko-dic은 MeCab을 사용하여, 한국어 형태소 분석을 하기 위한 프로젝트입니다.
mecab	SL,*,*,*,*,*,*,*
-	SY,*,*,*,*,*,*,*
ko	SL,*,*,*,*,*,*,*
-	SY,*,*,*,*,*,*,*
dic	SL,*,*,*,*,*,*,*
은	JX,*,T,은,*,*,*,*
MeCab	SL,*,*,*,*,*,*,*
을	JKO,*,T,을,*,*,*,*
사용	NNG,행위,T,사용,*,*,*,*
하	XSV,*,F,하,*,*,*,*
여	EC,*,F,여,*,*,*,*
,	SC,*,*,*,*,*,*,*
한국어	NNG,*,F,한국어,Compound,*,*,한국/NNG/*+어/NNG/*
형태소	NNG,*,F,형태소,Compound,*,*,형태/NNG/*+소/NNG/*
분석	NNG,행위,T,분석,*,*,*,*
을	JKO,*,T,을,*,*,*,*
하	VV,*,F,하,*,*,*,*
기	ETN,*,F,기,*,*,*,*
위한	VV+ETM,*,T,위한,Inflect,VV,ETM,위하/VV/*+ㄴ/ETM/*
프로젝트	NNG,*,F,프로젝트,*,*,*,*
입니다	VCP+EF,*,F,입니다,Inflect,VCP,EF,이/VCP/*+ㅂ니다/EF/*
.	SF,*,*,*,*,*,*,*
EOS
```

```
import MeCab

m = MeCab.Tagger()
out = m.parse("아빠가가방에들어가신다.")

print(out)

>>> 아빠	NNG,*,F,아빠,*,*,*,*
>>> 가	JKS,*,F,가,*,*,*,*
>>> 가방	NNG,*,T,가방,*,*,*,*
>>> 에	JKB,*,F,에,*,*,*,*
>>> 들어가	VV,*,F,들어가,*,*,*,*
>>> 신다	EP+EF,*,F,신다,Inflect,EP,EF,시/EP/*+ㄴ다/EF/*
>>> .	SF,*,*,*,*,*,*,*
>>> EOS
```

참고: https://doitgrow.com/396
https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=aul-_-&logNo=221557243190 (노래 가사 분석)
https://bitbucket.org/eunjeon/mecab-ko-dic/src/master/

3.3 도메인 특화 언어 (DSL: Domain-Specific Languages)

- 특정 도메인(산업, 분야등 특정 영역)에 특화된 언어를 말한다. (특정 영역의 문제 해결에는 그 영역에 맞는 특화된 도구를 사용하자라는 것), 정규 표현식도 이에 포함
- 어떤 목적이 있고 그 목적만 달성할 수 있는 언어

장점	단점
간결함 유지보수 높은 수준의 추상화 집중 관심사분리	DSL 설계의 어려움 개발비용 새로 배워야 하는 언어 호스팅 언어 한계

3.4 네이버 영화 평점 (감성분석)

```
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	너무재밋었다그래서보는것을추천한다	0

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

```
0    75173
1    74827
Name: label, dtype: int64
```

0과 1의 비율이 균등하다

- Train_df의 경우, document칼럼에 null이 일부 존재하므로 이 값은 공백으로 변환, 문자가 아닌 숫자의 경우 의미해석이 부족하므로 re를 이용해 이것도 공백으로 변환
- TF-IDF방식으로 단어를 벡터화, 각 문장을 한글 형태소 분석->형태소 단어로 토큰화
- Twitter방식으로 진행, twitter의 morphs를 이용

```
import re

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

# 테스트 데이터 셋을 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) )

# 개정판 소스 코드 변경(2019.12.24)
train_df.drop('id', axis=1, inplace=True)
test_df.drop('id', axis=1, inplace=True)
```

```
from konlpy.tag import Twitter

twitter = Twitter()
def tw_tokenizer(text):
    # 입력 인자로 들어온 text 를 형태소 단어로 토큰화 하여 list 객체 반환
    tokens_ko = twitter.morphs(text)
    return tokens_ko
```

EWHA EURON

3.4 네이버 영화 평점 (감성분석)

```
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'])
```

```
C:₩Users₩kyeongmoo₩anaconda3₩lib₩site-packages₩sklearn₩feature_extraction₩text.py:516: UserWar
not be used since 'tokenizer' is not None'
  warnings.warn(
```

- 사이킷런의 TfidVectorizer을 이용하여 TF-IDF모델 생성
- 앞서 만든 tw_tokenizer을 이용
- 8분 가량 소요됨

```
# Logistic Regression 을 이용하여 감성 분석 Classification 수행.
lg_clf = LogisticRegression(random_state=0)

# Parameter C 최적화를 위해 GridSearchCV 를 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))
```

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

- 로지스틸 회귀를 이용, 최적의 하이퍼 파라미터(GridSearchCV) 찾기
- C가 3.5일 때 최고 0.8593의 정확도

```
from sklearn.metrics import accuracy_score

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

# classifier 는 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))
```

```
Logistic Regression 정확도:  0.86186
```

- 테스트 세트를 이용해 분석 예측을 진행
- Logistic Regression 정확도: 86%수준

EWHA EURON

04. 캐글 실습

EWHA EURON

#4.1 캐글 실습 대형 온라인 쇼핑몰 제품 가격 예측

📌 Mercari Price Suggestion

- 목적 : 일본의 대형 온라인 쇼핑몰인 Mercari사의 제품에 대해 가격을 예측하는 것
- 데이터셋
  - train_id : 데이터 id
  - name : 제품명
  - item_condition_id : 판매자가 제공하는 제품 상태
  - category_name : 카테고리 명
  - brand_name : 브랜드 이름
  - shipping : 배송비 무료 여부 (1이면 무료 / 0이면 유료)
  - item_description : 제품에 대한 설명
  - price : 제품 가격, 예측을 위한 타깃 속성

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1482535 entries, 0 to 1482534
Data columns (total 8 columns):
train_id             1482535 non-null int64
name                 1482535 non-null object
item_condition_id    1482535 non-null int64
category_name        1476208 non-null object
brand_name           849853 non-null object
price                1482535 non-null float64
shipping             1482535 non-null int64
item_description     1482531 non-null object
dtypes: float64(1), int64(3), object(4)
memory usage: 90.5+ MB
None
```

null 값들은 추후 Other_Null 로 치환

EWHA EURON

#4.1 캐글 실습 대형 온라인 쇼핑몰 제품 가격 예측

📌 데이터 전처리

- price 칼럼
  - 정규 분포에 가깝도록 로그 값으로 변환
- category_name 칼럼
  - '대분류/중분류/소분류' 값으로 저장되어 있음 -> '/' 기준으로 토큰화

```
대분류 유형 :
 Women                 664385
Beauty                 207828
Kids                   171689
Electronics            122690
Men                     93680
Home                    67871
Vintage & Collectibles  46530
Other                   45351
Handmade                30842
Sports & Outdoors       25342
Other_Null               6327
Name: cat_dae, dtype: int64
중분류 갯수 : 114
소분류 갯수 : 871
```

📌 데이터 인코딩 & 피처 벡터화

- 문자열 칼럼들 중 레이블 또는 원-핫 인코딩 수행 or 피처 벡터화로 변환할 칼럼 선별
  - 예측값이 price이므로 회귀모델 사용 -> 원-핫 인코딩 적용
  - 피처 벡터화의 경우

    (짧은 텍스트) count 기반의 벡터화

    (긴 텍스트) TD-IDF

#4.2 데이터 인코딩 및 피처 벡터화

📌 데이터 인코딩 & 피처 벡터화

- brand_name

  - 대부분 명료한 문자열 & 4180개 -> 원-핫 인코딩 변환

➡ • name

  - 유형이 많고 적은 단어 위주의 텍스트 -> count 기반 피처 벡터화

- category_name

  - 대,중,소 분류 3개의 칼럼으로 분류 & 유형 별로 없음 -> 원-핫 인코딩

- shipping

  - 배송비 여부로 0과 1 -> 원-핫 인코딩

➡ • item_description

  - 평균 문자열이 145자로 비교적 큼 -> TD-IDF

#4.2 데이터 인코딩 및 피처 벡터화

📌 데이터 인코딩 & 피처 벡터화

- name과 item_description 피처 벡터화

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

```
# name 속성에 대한 feature vectorization 변환
cnt_vec = CountVectorizer()
X_name = cnt_vec.fit_transform(mercari_df.name)

# item_description 에 대한 feature vectorization 변환
tfidf_descp = TfidfVectorizer(max_features = 50000, ngram_range= (1,3) , stop_words='english')
X_descp = tfidf_descp.fit_transform(mercari_df['item_description'])
```

- CountVectorizer과 TD-IDF의 fit_transform() 형태로 반환하는 데이터는 희소 행렬 형태

  -> 하나의 데이터셋으로 ML 모델을 실행하기 때문에 다른 인코딩도 모두 희소 행렬 형태로 반환

  -> LabelBinarizer 클래스 사용 (sparse_out=True)

```
from sklearn.preprocessing import LabelBinarizer

# brand_name, item_condition_id, shipping 각 피처들을 희소 행렬 원-핫 인코딩 변환
lb_brand_name= LabelBinarizer(sparse_output=True)
X_brand = lb_brand_name.fit_transform(mercari_df['brand_name'])
```

EWHA EURON

#4.2 데이터 인코딩 및 피처 벡터화

📌 피처 데이터 결합

- 변환된 각각의 희소 행렬(csr_matrix)들을 사이파이 패키지 sparse 모듈의 hstack() 이용해 결합

```
from  scipy.sparse import hstack
import gc

sparse_matrix_list = (X_name, X_descp, X_brand, X_item_cond_id,
            X_shipping, X_cat_dae, X_cat_jung, X_cat_so)

# 사이파이 sparse 모듈의 hstack 함수를 이용하여 앞에서 인코딩과 Vectorization을 수행한 데이터 셋을 모두 결합.
X_features_sparse= hstack(sparse_matrix_list).tocsr()
print(type(X_features_sparse), X_features_sparse.shape)

# 데이터 셋이 메모리를 많이 차지하므로 사용 용도가 끝났으면 바로 메모리에서 삭제.
del X_features_sparse
gc.collect()
```

csr_matrix 타입으로 결합 & float 형

https://domybestinlife.tistory.com/151
https://blog.naver.com/PostView.nhn?isHttpsRedirect=true&blogId=hongyou022&logNo=222037484474&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView

#4.3 모델 구축 및 평가

📌 평가 기준

- RMSLE
  - 오류 값에 로그를 취해 RMSE를 구하는 방식
  - 낮은 가격보다 높은 가격에서 오류가 발생할 경우 오류 값이 커지는 것을 억제하기 위해

$$\text{RMSLE}=\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log\left(y_i+1\right)-\left(\widehat{y+1}\right)\right)^2}$$

```python
def rmsle(y , y_pred):
    # underflow, overflow를 막기 위해 log가 아닌 log1p로 rmsle 계산
    return np.sqrt(np.mean(np.power(np.log1p(y) - np.log1p(y_pred), 2)))

def evaluate_org_price(y_test , preds):

    # 원본 데이터는 log1p로 변환되었으므로 exmpm1으로 원복 필요.
    preds_exmpm = np.expm1(preds)
    y_test_exmpm = np.expm1(y_test)

    # rmsle로 RMSLE 값 추출
    rmsle_result = rmsle(y_test_exmpm, preds_exmpm)
    return rmsle_result
```

#4.3 모델 구축 및 평가

📌 모델 구축 및 평가

- Ridge

```
import gc
from  scipy.sparse import hstack

def model_train_predict(model,matrix_list):
    # scipy.sparse 모듈의 hstack 을 이용하여 sparse matrix 결합
    X= hstack(matrix_list).tocsr()

    X_train, X_test, y_train, y_test=train_test_split(X, mercari_df['price'],
                                                      test_size=0.2, random_state=156)

    # 모델 학습 및 예측
    model.fit(X_train , y_train)
    preds = model.predict(X_test)

    del X , X_train , X_test , y_train
    gc.collect()

    return preds , y_test
```

```
linear_model = Ridge(solver = "lsqr", fit_intercept=False)

sparse_matrix_list = (X_name, X_brand, X_item_cond_id,
                      X_shipping, X_cat_dae, X_cat_jung, X_cat_so)
linear_preds , y_test = model_train_predict(model=linear_model ,matrix_list=sparse_matrix_list)
print('Item Description을 제외했을 때 rmsle 값:', evaluate_org_price(y_test , linear_preds))

sparse_matrix_list = (X_descp, X_name, X_brand, X_item_cond_id,
                      X_shipping, X_cat_dae, X_cat_jung, X_cat_so)
linear_preds , y_test = model_train_predict(model=linear_model , matrix_list=sparse_matrix_list)
print('Item Description을 포함한 rmsle 값:',  evaluate_org_price(y_test ,linear_preds))
```

```
Item Description을 제외했을 때 rmsle 값: 0.5021632139113013
Item Description을 포함한 rmsle 값: 0.47122043277496645
```

EWHA EURON

#4.3 모델 구축 및 평가

📌 모델 구축 및 평가

- LightGBM

```
from lightgbm import LGBMRegressor

sparse_matrix_list = (X_descp, X_name, X_brand, X_item_cond_id,
                      X_shipping, X_cat_dae, X_cat_jung, X_cat_so)

lgbm_model = LGBMRegressor(n_estimators=200, learning_rate=0.5, num_leaves=125, random_state=156)
lgbm_preds , y_test = model_train_predict(model = lgbm_model , matrix_list=sparse_matrix_list)
print('LightGBM rmsle 값:',  evaluate_org_price(y_test , lgbm_preds))
```

```
LightGBM rmsle 값: 0.4558577120744113
```

- Ridge + LightGBM

```
preds = lgbm_preds * 0.45 + linear_preds * 0.55
print('LightGBM과 Ridge를 ensemble한 최종 rmsle 값:',  evaluate_org_price(y_test , preds))
```

```
LightGBM과 Ridge를 ensemble한 최종 rmsle 값: 0.4501738147377442
```

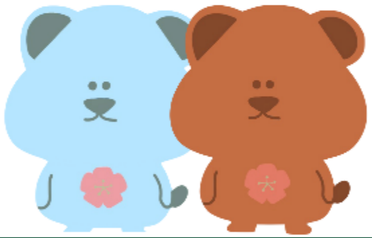
비율은 임의 산정

EWHA EURON

05. 대회 솔루션 분석

EWHA EURON

5.1 로그 분석을 통한 보안 위험도 예측 AI 경진대회



EWHA EURON

#5.1.1 대회 설명

📌 로그 분석을 통한 보안 위험도 예측 AI 경진대회

- 목적 : 로그 데이터를 통해 시스템의 보안 위험도 등급을 예측하고 기존에 없던 패턴의 공격 탐지
- 데이터셋

  (train data)

  - id : 데이터 식별자
  - full_log : 학습 데이터 전체 로그
  - level : 보안 위험 등급 (0~6)

  (test data)

  - id : 데이터 식별자
  - full_log : 테스트 데이터 전체 로그
  - 이때 보안 위험 등급은 0~7

로그 예시

Sep 24 10:02:22 localhost kibana:
{"type":"error","@timestamp":"2020-09-24T01:02:22Z","tags":
["warning","stats-collection"],"pid":6458,"level":"error","error":
{"message":"No Living connections","name":"Error","stack":"Error:
No Living connections₩n at sendReqWithConnection
(/usr/share/kibana/node_modules/elasticsearch/src/lib/transport.js:
226:15)₩n at next
(/usr/share/kibana/node_modules/elasticsearch/src/lib/connection
_pool.js:214:7)₩n at process._tickCallback
(internal/process/next_tick.js:61:11)"},"message":"No Living
connections"}

#05 캐글 노트북 로그 분석을 통한 보안 위험도 예측 AI 경진대회

📌 Baseline

- CountVectorizer()을 이용해 full_log 피처 벡터화

```
#full_log에서 숫자는 마스킹 처리
train['full_log']=train['full_log'].str.replace(r'[0-9]', '<num>')
test['full_log']=test['full_log'].str.replace(r'[0-9]', '<num>')
```

```
#train['full_log'] => train_text로 list
#train['level']=> train_level로 array
train_text=list(train['full_log'])
train_level=np.array(train['level'])
```

```
#CountVectorizer로 벡터화
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(analyzer="word", max_features=10000)

train_features=vectorizer.fit_transform(train_text)
```

```
#랜덤포레스트로 모델링
from sklearn.ensemble import RandomForestClassifier

forest=RandomForestClassifier(n_estimators=100)

forest.fit(train_x, train_y)
```

EWHA EURON

#5.1.2 Baseline Code

📌 Baseline

- 랜덤포레스트 이용

```
#랜덤포레스트로 모델링
from sklearn.ensemble import RandomForestClassifier

forest=RandomForestClassifier(n_estimators=100)

forest.fit(train_x, train_y)
```

```
preds=forest.predict(eval_x)
probas=forest.predict_proba(eval_x)
```

- 새로운 level인 7등급 필요 -> 예측 확률이 0.9 이하일 경우 이상치로 판단해 새로운 level 7 부여

```
preds[np.where(np.max(probas, axis=1)<0.90)]=7
```

#5.1.2 Baseline Code 변형 1등 솔루션 코드

📌 Baseline 변형

- 마찬가지로 threshold를 잡아 7등급 설정
  - 이때 train 문장과 완전히 동일할 경우 threshold에 걸리지 않도록 설정
  - 이후 7등급 이외의 등급은 전처리가 덜 진행된 데이터를 활용해 지도학습 진행

- 실제 분류방법 정리

  ① 로그 문장 데이터 전처리

  ② Train 데이터와 동일한 문장이 있는지 확인해 서로 다른 모델 적용

  ③ 동일한 문장이 있다면 0~6등급 지도 학습된 모델로 predict (threshold 적용 X)

  ④ 동일한 문장이 없다면 0~6등급 지도 학습된 모델로 predict 이후 설정된 threshold를 통해 7 분류

  ⑤ 앞 단계에서 threshold를 통해 걸러지지 않는다면 3번과 동일한 모델로 predict 적용

#5.1.2 Baseline Code 변형 1등 솔루션 코드

📌 Baseline 변형

- 전처리 진행
  - 다양한 기호 및 숫자 제거
  - 영어 단어가 아니거나 3글자 미만인 경우 삭제

➡ Threshold 잡는 것에는 효율적
but 지도 학습 성능은 떨어짐

- Validation을 통해 threshold 설정

```
#Validation을 통한 threshold 조절
results[np.where((np.max(results_proba, axis=1)<0.5) & (results == 0))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.5) & (results == 1))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.58) & (results == 2))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.95) & (results == 3))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.58) & (results == 4))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.58) & (results == 5))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.58) & (results == 6))[0]]=7
```

```
#Validation을 통한 threshold 조절
results[np.where((np.max(results_proba, axis=1)<0.94001) & (np.max(results_proba, axis=1) > 0.93999) & (results == 1))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.611657) & (np.max(results_proba, axis=1) > 0.6116568) & (results == 0))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.571657) & (np.max(results_proba, axis=1) > 0.5716568) & (results == 0))[0]]=7
results[np.where((np.max(results_proba, axis=1)<0.68001) & (np.max(results_proba, axis=1) > 0.67999) & (results == 5))[0]]=7
```

#5.1.2 Baseline Code 변형 1등 솔루션 코드

📌 Baseline 변형

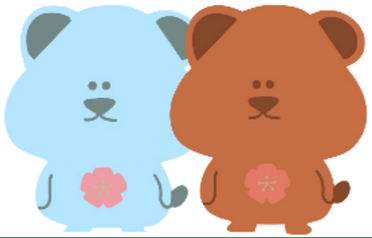
- 해결 방법
  - 7을 제외한 나머지 등급에는 전처리가 덜 된 데이터로 지도학습 분류
  - 이때 TD-IDF 사용

```
vectorizer=TfidfVectorizer(analyzer="word", max_features=20000)
train_features=vectorizer.fit_transform(train_text)
test_features=vectorizer.transform(test_text)
```

  - 이후 7등급 데이터와 병합

```
#위에서 잡은 7 합치기 & train에 있는 경우는 사용하지 않기
only = pd.read_csv("tem_answer.csv")
le7 = list(only[only['level']==7]['id'])
le7_update = list((set(le7))-set(not7_id))
print(len(le7))
print(len(le7_update))
idx = submission[submission['id'].isin(le7_update)].index
submission.iloc[idx] = 7
submission['level'].value_counts()
```

5.2 금융 문자 분석 경진대회



EWHA EURON

#5.2.1 대회 설명

📌 금융 문자 분석 경진대회

- 목적 : 문자 데이터를 활용한 스미싱 탐지 모델 개발
- 데이터셋

  (train data)

  - id : 각 문자가 가지고 있는 고유 구분 번호
  - year_month : 고객이 문자를 전송 받은 연도와 해
  - text : 고객이 전송 받은 문자 내용
  - smishing : 해당 문자의 스미싱 여부 (0 : 아님 / 1 : 스미싱 맞음)

#5.2.2 파이프라인

📌 금융 문자 분석 경진대회

- 파이프라인
  ① Mecab을 이용해 한국어 형태소 토큰화
  ② 필요 없는 단어 제거
  ③ CountVectorizer을 이용해 텍스트 벡터화
  ④ multinomialNB를 이용해 모델 학습 후 적용

```
train_doc = [ ( tokenizer.pos(x), y ) for x, y in tqdm( zip( train_xx['text'], train_yyy['smishing'] ) )  ] # Mecab를 활용하여 text를 토큰화 시킴
test_doc = [ ( tokenizer.pos(x), y ) for x, y in tqdm( zip( test_xx['text'], test_yyy['smishing'] ) )  ]
```

```
stopwords = ['XXX', '.', '을', '를', '이', '가', '-', '(', ')', ':', '!', '?', ')-', '.-', '—', 'XXXXXX', '..', '.(', '은', '는'] #필요없는 단어 리스트

def get_couple(_words): #필요없는 단어들 없애는 함수
    global stopwords
    _words = [x for x in _words if x[0] not in stopwords]
    l = len(_words)
    for i in range(l-1):
        yield _words[i][0], _words[i+1][0]
```

#5.2.2 파이프라인

📌 금융 문자 분석 경진대회

- 파이프라인
  ① Mecab을 이용해 한국어 형태소 토큰화
  ② 필요 없는 단어 제거
  ③ CountVectorizer을 이용해 텍스트 벡터화
  ④ multinomialNB를 이용해 모델 학습 후 적용

```
v=CountVectorizer()

v.fit(X_train)

vec_x_train= v.transform(X_train).toarray()
vec_x_test= v.transform(X_test).toarray()
```

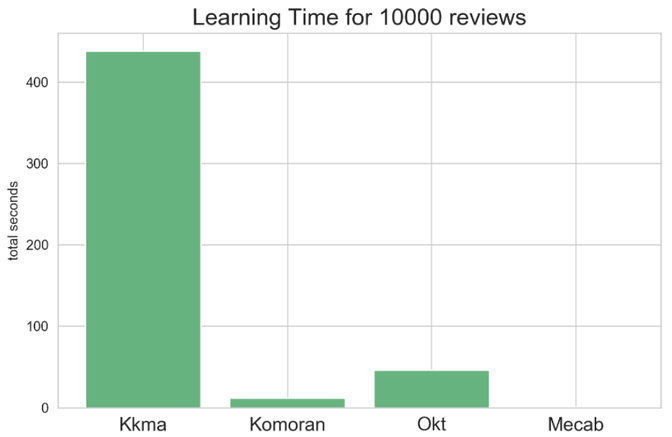
```
m1= MultinomialNB()
m1.fit(vec_x_train,Y_train)

y_train_pred1=m1.predict_proba(vec_x_train)
y_train_pred1_one= [ i[1]  for i in y_train_pred1]

y_test_pred1=m1.predict_proba(vec_x_test)
y_test_pred1_one= [ i[1]  for i in y_test_pred1]
```

#5.2.3 Mecab

📌 Mecab

- 형태소 분석기
- 기능
  - nouns() : 명사만 분류
  - morphs() : 형태소 단위로 끊어주기
  - pos() : 문장을 형태소 단위로 끊어주고 형태소 분석

```
import konlpy
from konlpy.tag import Mecab

tokenizer = Mecab() # setting tokenizer using Mecab()
```



https://soohee410.github.io/compare_tagger

(품사 태그)

실질의미유무	대분류(5언 + 기타)	세종 품사 태그		mecab-ko-dic 품사 태그	
		태그	설명	태그	설명
실질형태소	체언	NNG	일반 명사	NNG	일반 명사
		NNP	고유 명사	NNP	고유 명사
		NNB	의존 명사	NNB	의존 명사
				NNBC	단위를 나타내는 명사
		NR	수사	NR	수사
		NP	대명사	NP	대명사
	용언	VV	동사	VV	동사
		VA	형용사	VA	형용사
		VX	보조 용언	VX	보조 용언
		VCP	긍정 지정사	VCP	긍정 지정사
		VCN	부정 지정사	VCN	부정 지정사
	수식언	MM	관형사	MM	관형사
		MAG	일반 부사	MAG	일반 부사
		MAJ	접속 부사	MAJ	접속 부사
	독립언	IC	감탄사	IC	감탄사

(예시)

```
INPUT>>새파란 하늘 쨍쨍 해가 떴네
[('새파란', 'VA+ETM'), ('하늘', 'NNG'), ('쨍쨍', 'MAG'), ('해', 'NNG'), ('가', 'JKS'), ('떴', 'VV+
EP'), ('네', 'EF')]
INPUT>>어저께랑 날씨는 같지만 똑같은 느낌은 아닐 걸
[('어저께', 'NNG'), ('랑', 'JKB'), ('날씨', 'NNG'), ('는', 'JX'), ('같', 'VA'), ('지만', 'EC'), ('
똑같', 'VA'), ('은', 'ETM'), ('느낌', 'NNG'), ('은', 'JX'), ('아닐', 'VCN+ETM'), ('걸', 'NNB+JKO')
]
INPUT>>나가 여태껏 누려왔던 모든 것들이
```

https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=aul-_-&logNo=221557243190

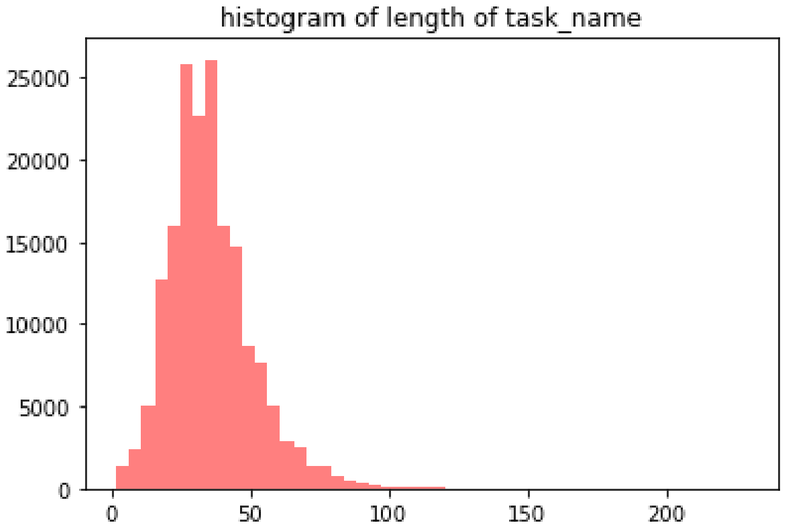
5.3 자연어 기반 기후기술분류 AI 경진대회

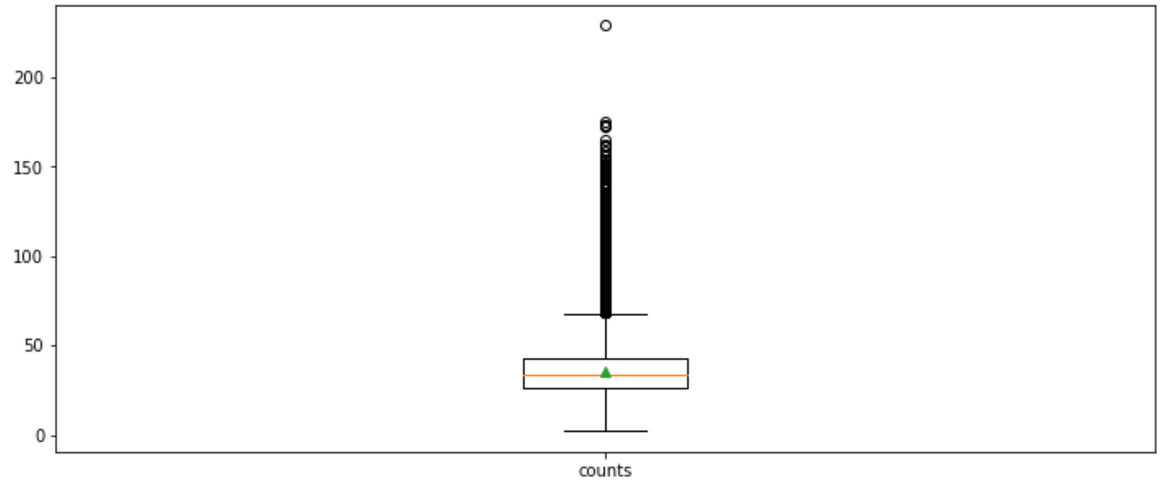
EWHA EURON

#5.3.1 데이터 EDA

기후 기술에 관한 연구개발 문헌을 ‘기후기술분류체계’에 맞추어 라벨링하는 알고리즘 개발

[자연어 기반 기후기술분류 AI 경진대회 - DACON]





‘과제명’ column 분포
(본 baseline 코드에서는 ‘과제명’ column만 사용함)

#5.3.2 okt 전처리 및 불용어 제거

- Okt 전처리

```
#1. re.sub 한글 및 공백을 제외한 문자 제거
#2. okt 객체를 활용해 형태소 단위로 나눔
#3. remove_stopwords로 불용어 제거
def preprocessing(text, okt, remove_stopwords=False, stop_words=[]):
    text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ]","",text)
    word_text = okt.morphs(text, stem=True)
    if remove_stopwords:
        word_review = [token for token in word_text if not token in stop_words]
    return word_review
```

-> 텍스트에서 형태소를 반환

```
my_sen = "넌 찾아 헤매다 무심코 지나친 꽃이 참 예쁘구나"
okt = Okt()
print(okt.morphs(my_sen))
print(okt.nouns(my_sen))
print(okt.pos(my_sen))
```

executed in 102ms, finished 22:53:28 2022-06-13

```
['넌', '찾아', '헤매다', '무심코', '지나친', '꽃', '이', '참', '예쁘구나']
['넌', '무심코', '꽃']
[('넌', 'Noun'), ('찾아', 'Verb'), ('헤매다', 'Verb'), ('무심코', 'Noun'), ('지나친', 'Verb'), ('꽃', 'Noun'), ('이', 'Josa'), ('참', 'Verb'),
('예쁘구나', 'Adjective')]
```

EWHA EURON

#5.3.2 okt 전처리 및 불용어 제거

- 불용어 제거

```
#1. re.sub 한글 및 공백을 제외한 문자 제거
#2. okt 객체를 활용해 형태소 단위로 나눔
#3. remove_stopwords로 불용어 제거
def preprocessing(text, okt, remove_stopwords=False, stop_words=[]):
    text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ]","",text)
    word_text = okt.morphs(text, stem=True)
    if remove_stopwords:
        word_review = [token for token in word_text if not token in stop_words]
    return word_review
```

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

```
for text in tqdm.tqdm(train['과제명']):
    try:
        clean_train_text.append(preprocessing(text, okt, remove_stopwords=True, stop_words=stop_words))
    except:
        clean_train_text.append([])
```

Test 세트에 대해서도 동일한 처리

EWHA EURON

#5.3.3 단어 벡터화

- CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
#tokenizer 인자에는 list를 받아서 그대로 내보내는 함수를 넣어줌
#또한 소문자화를 하지 않도록 설정해야 에러가 나지 않음
vectorizer = CountVectorizer(tokenizer=lambda x: x, lowercase=False)
train_features = vectorizer.fit_transform(clean_train_text)
test_features = vectorizer.transform(clean_test_text) #test data에는 fit 수행하지 않는 것 주의!
```

-> CountVectorizer 돌리기 전 '은', '는 ' 과 같은 조사 처리 필수 (같은 의미인 단어여도 다르게 count되기 때문)

-> train 데이터셋에만 fit 수행 (test 데이터셋은 X!)

-> 벡터화된 단어 데이터들을 원하는 분류 알고리즘으로 모델링 작업 ~~

5.4 Covid Literature clustering

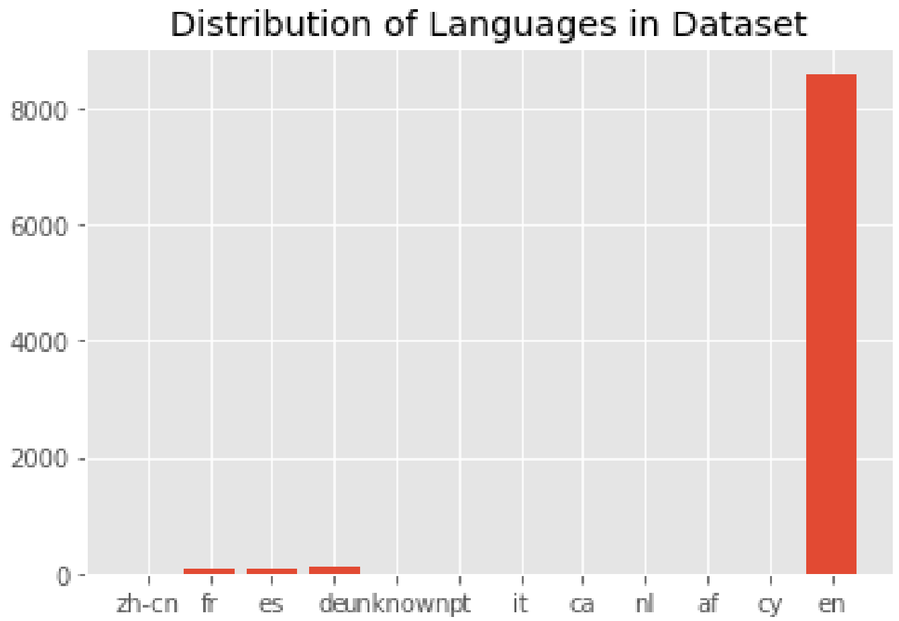
EWHA EURON

#5.4.1 데이터 EDA 및 전처리

COVID 관련 유사한 연구 기사의 clustering을 통해 관련 publications 검색을 단순화해보자

COVID-19 Literature Clustering | Kaggle

1. Language 분포 확인 후 english자료만 남기기




```
df = df[df['language'] == 'en']
df.info()
```

2. 불용어 제거

```
custom_stop_words = [
    'doi', 'preprint', 'copyright', 'peer', 'reviewed', 'org', 'https', 'et', 'al', 'author', 'figure',
    'rights', 'reserved', 'permission', 'used', 'using', 'biorxiv', 'medrxiv', 'license', 'fig', 'fig.',
    'al.', 'Elsevier', 'PMC', 'CZI'
]

for w in custom_stop_words:
    if w not in stopwords:
        stopwords.append(w)
```

#5.4.2 단어 벡터화

- tf-idf 사용(TfidfVectorizer)

```
from sklearn.feature_extraction.text import TfidfVectorizer
def vectorize(text, maxx_features):

    vectorizer = TfidfVectorizer(max_features=maxx_features)
    X = vectorizer.fit_transform(text)
    return X

text = df['processed_text'].values
max_features = 2**12

X = vectorize(text, max_features)
```

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

모든 문서에서 전반적으로 자주 등장하는 단어에 대해서는 패널티를 주고,
해당 문서에서만 자주 등장하는 단어에 높은 가중치를 주는 방식

EWHA EURON

#5.4.3 PCA & Clustering

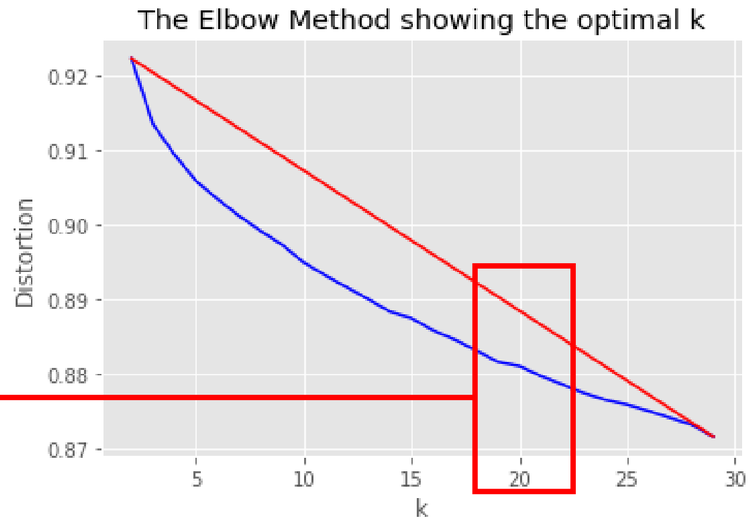
- dimension 줄이기(PCA)

```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95, random_state=42)
X_reduced= pca.fit_transform(X.toarray())
X_reduced.shape
```

-> 95%의 variance를 유지한 채 차원 축소

- k-means clustering (클러스터 개수는 20개)

```
from sklearn.cluster import MiniBatchKMeans
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist

# run kmeans with many different k
distortions = []
K = range(2, 30)
for k in K:
    k_means = KMeans(n_clusters=k, random_state=42).fit(X_reduced)
    k_means.fit(X_reduced)
    distortions.append(sum(np.min(cdist(X_reduced,
                                       k_means.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0])
```



k=20일때 distortion의 감소 최소
=> 최적의 cluster 개수(20개)

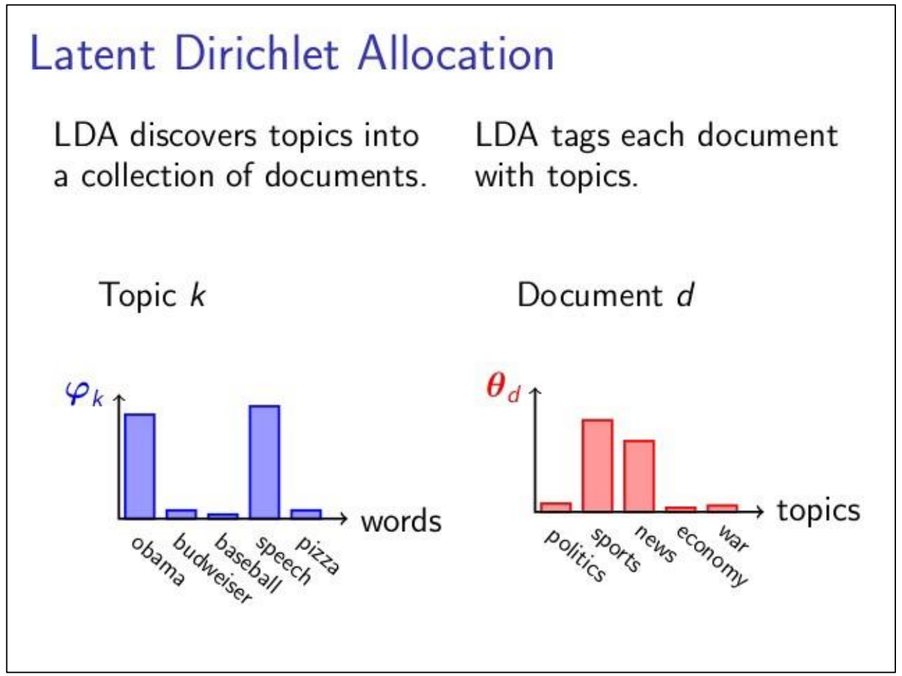
#5.4.4 Topic Modeling on Each Cluster

앞서 했던 K-means clustering은 topics에 대한 라벨링이 없음

-> Topic Modeling을 통해 각각의 cluster에 대한 중요한 term을 찾을 수 있음

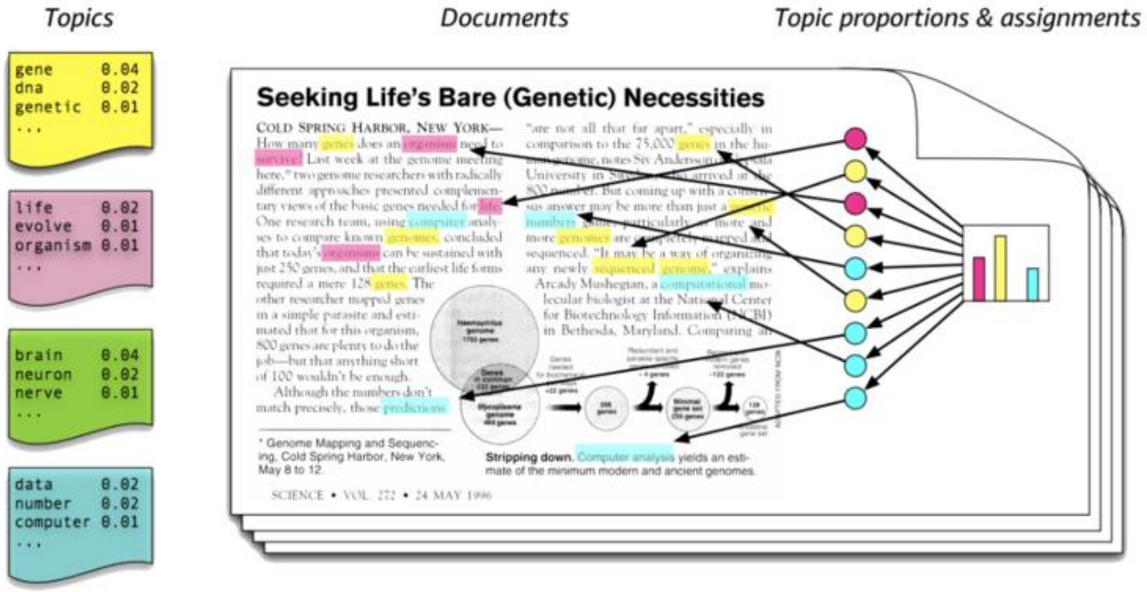
(add more meaning to the cluster)

=> Topic Modeling을 위해 LDA 사용 (Latent Dirichlet Allocation)



LDA모델은 문서별 단어 분포만을 가지고 Document-Term 행렬을 만들어 베이즈 추론을 이용해 **'토픽별 단어 분포'** 와 **'문서별 토픽 분포'**를 알아내는 것

#5.4.4 Topic Modeling on Each Cluster



<LDA모델 수행 프로세스>

1.단순 Count 기반 Document-Term 행렬을 생성 (주어진 단어들의 빈도수에 기반)

2.토픽의 개수를 사전에 설정

3.각 단어들을 임의의 토픽으로 최초 할당한 후 문서별 토픽 분포와 토픽별 단어 분포 결정

4.특정 단어를 하나 추출하고 추출한 해당 단어를 제외하고 문서의 토픽 분포와 토픽별 단어 분포를 다시 계산 및 추출된 단어는 새롭게 토픽 할당 분포를 계산

5.다른 단어를 추출하고 4번 단계를 다시 수행, 또 다른 단어를 추출하고 계속적으로 모든 단어들이 재 계산되도록 반복함

6.지정된 반복 횟수(하이퍼파라미터로 지정)만큼 4,5번 단계를 수행하면서 모든 단어들의 토픽 할당 분포가 변경되지 않고 수렴할 때까지 수행

#5.4.4 Topic Modeling on Each Cluster

1. 20개의 vectorizers 생성

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

vectorizers = []

for ii in range(0, 20):
    # Creating a vectorizer
    vectorizers.append(CountVectorizer(min_df=5, max_df=0.9, stop_words='english', lowercase=True,
                                       token_pattern='[a-zA-ZW-][a-zA-ZW-]{2,}'))
```

주어진 단어들의 빈도수에 기반하므로 Tf-idf 방법이 아닌 CountVectorizer 방법에 기반

2. 각 cluster에서 데이터 vectorize

```
vectorized_data = []

for current_cluster, cvec in enumerate(vectorizers):
    try:
        vectorized_data.append(cvec.fit_transform(df.loc[df['y'] == current_cluster, 'processed_text']))
    except Exception as e:
        print("Not enough instances in cluster: " + str(current_cluster))
        vectorized_data.append(None)
```

EWHA EURON

#5.4.4 Topic Modeling on Each Cluster

3. LDA를 이용해 Topic Modeling 수행

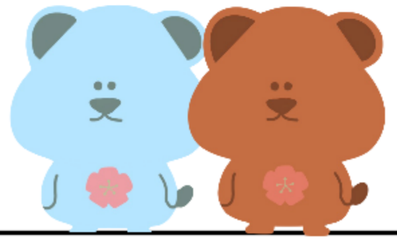
```
# number of topics per cluster
NUM_TOPICS_PER_CLUSTER = 20


lda_models = []

for ii in range(0, 20):
    # Latent Dirichlet Allocation Model
    lda = LatentDirichletAllocation(n_components=NUM_TOPICS_PER_CLUSTER, max_iter=10,
                                    learning_method='online',verbose=False, random_state=42)
    lda_models.append(lda)

clusters_lda_data = []

for current_cluster, lda in enumerate(lda_models):
    print("Current Cluster: " + str(current_cluster))

    if vectorized_data[current_cluster] != None:
        clusters_lda_data.append((lda.fit_transform(vectorized_data[current_cluster])))
```

=>

추출된 topic keyword(20개) 중 10개

```
['disease',
 'covid-',
 'research',
 'food',
 'community',
 'datum',
 'company',
 'case',
 'social',
 'technology']
```

THANK YOU



EWHA EURON