

빅데이터통계기법 과제01

텍스트 마이닝 실습

202050135 우주지질학과 홍지민


한글 논문에 대한 기초통계분석

- Python 패키지의 Konlpy 라이브러리를 이용하여 형태소를 분석 하였습니다.
- 띄어쓰기가 제대로 안된 부분은 KoSpacing 라이브러리를 이용하여 띄어쓰기를 진행한 후에 분석 했습니다.
(KoSpacing은 딥러닝 기반 띄어쓰기 라이브러리입니다)
- Konlpy의 Komoran 클래스를 이용하여 뛰어난 성능을 보여줌을 확인 했습니다.
- Sklearn의 벡터화 라이브러리를 통해 Term-Document-Matrix를 만들고 Count와 TF-IDF차이를 비교 했습니다.

라이브러리 불러오기

```
from konlpy.tag import *
from pykospacing import spacing
from nltk import Text
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from wordcloud import WordCloud
import numpy as np
import pandas as pd
import re, natsort, glob
import tomotopy as tp
```

km = Komoran() # 코모란 클래스 이용
plt.rc('font', family='NanumBarunGothic') # 한글 지정
model = tp.LDAModel(k=7,alpha=0.1,eta=0.01,min_df=1) # 토픽 모델 생성, 토픽 7개



토픽 모델에서 사용할 모델을
미리 지정 해줬습니다

한글 논문에 대한 기초통계분석

```
def han_fun(doc):
```

```
    nouns = km.nouns(doc)
```

```
    nouns = [noun for noun in nouns if len(noun)>=2] # 2글자 이상 필터링(남아있는 불용어 제거 목적)
```

```
    return nouns
```

```
text_list = natsort.natsorted(glob.glob('kp'+ '*.txt'))
```

```
corpus, token_doc = [], []
```

```
for file in text_list:
```

```
    file_data = open(file, 'r', encoding='cp949').readlines()[0]
```

```
    file_data = spacing(file_data) # 딥러닝을 이용한 띄어쓰기
```

```
    file_data = re.sub("#d+", " ", file_data) # 숫자 제거
```

```
    file_data = re.sub("연구|논문", "", file_data) # 본 연구, 본 논문과 같은 불용어 제거
```

```
    file_data = re.sub('[!-+=, #/ #? : ^ $ . @ * # " % ~ & % . ! , " # ( # ) # [ # ] # < # > ` # ' ... ]', '', file_data) # 특수문자 제거
```

```
    token = han_fun(file_data)
```

```
    model.add_doc(Text(token))
```

```
    corpus.append(file_data)
```

```
vector = TfidfVectorizer(tokenizer=han_fun) # TF-IDF Matrix 생성
```

```
vector1 = CountVectorizer(tokenizer=han_fun) # count Matrix 생성
```

```
tdm = vector.fit_transform(corpus).toarray()
```

```
tdm1 = vector1.fit_transform(corpus).toarray()
```

```
column = vector.get_feature_names() # 컬럼명(단어) 얻음
```

```
column1 = vector1.get_feature_names() # 컬럼명(단어) 얻음
```

```
tdm = pd.DataFrame(tdm, index=text_list, columns=column) # Term-Document-Matrix 만들기
```

```
tdm1 = pd.DataFrame(tdm1, index=text_list, columns=column1) # Term-Document-Matrix 만들기
```

명사만 추출해서 토큰화 했습니다

가끔 불용어가 남아있는 경우가
있어 2글자 이상만 필터링 했습니다

띄어쓰기를 적용 후 토큰화

모델 적용

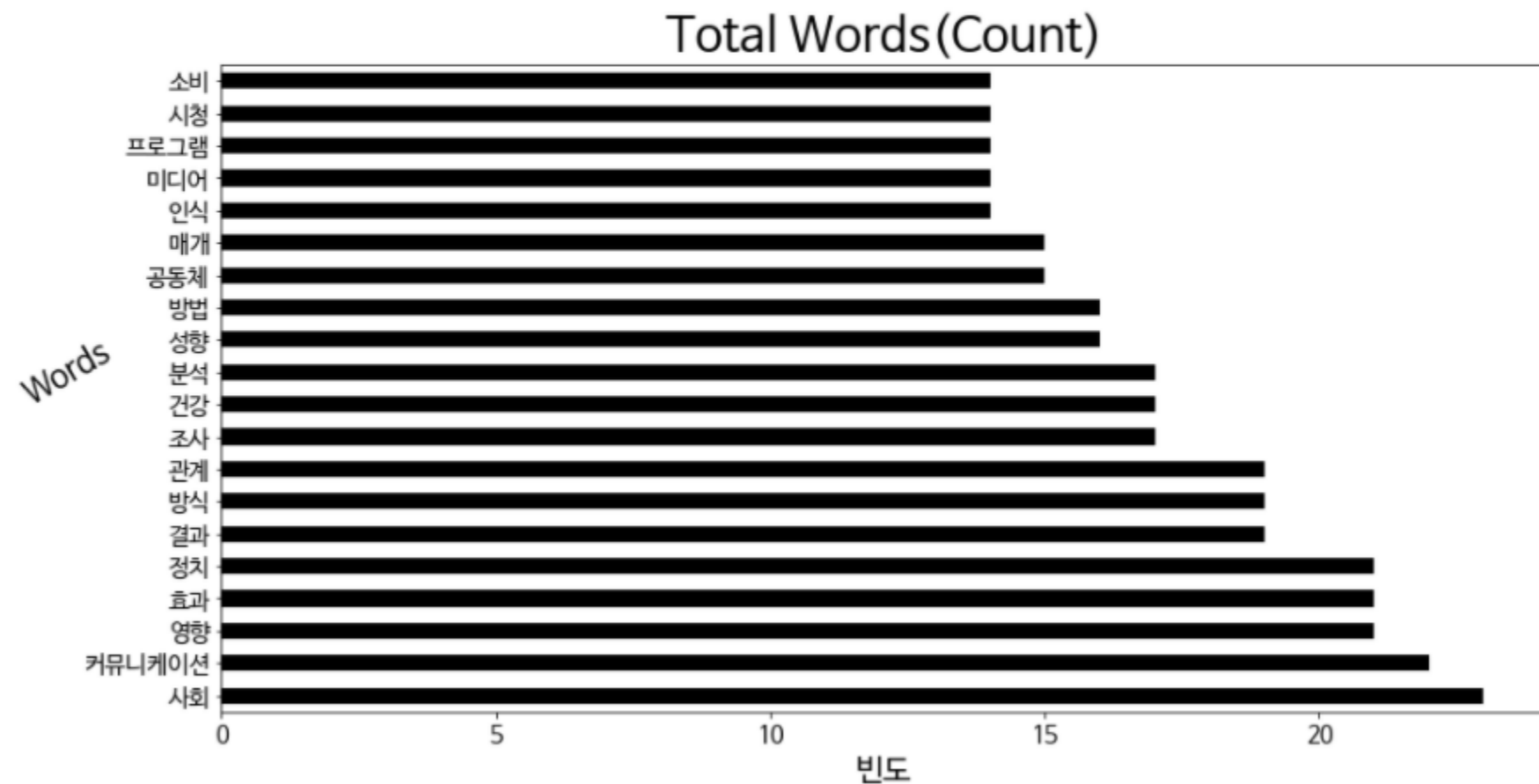
Count
vs
TF-IDF

TDM 생성 후 분석

- Count만을 이용해서 모든 문서를 분석한 결과입니다.

```
count1= tdm1.sum(axis=0).sort_values(ascending=False)[0:20]
plt.figure(figsize=(15,7.5))
count1.plot(kind="barh",color='black')
plt.xlabel("빈도",fontsize=20,rotation=0) ;plt.ylabel("Words",fontsize=20,rotation=30)
plt.xticks(fontsize=15) ; plt.yticks(fontsize=15)
plt.title("Total Words(Count)",fontsize=30)
```

Text(0.5, 1.0, 'Total Words(Count)')

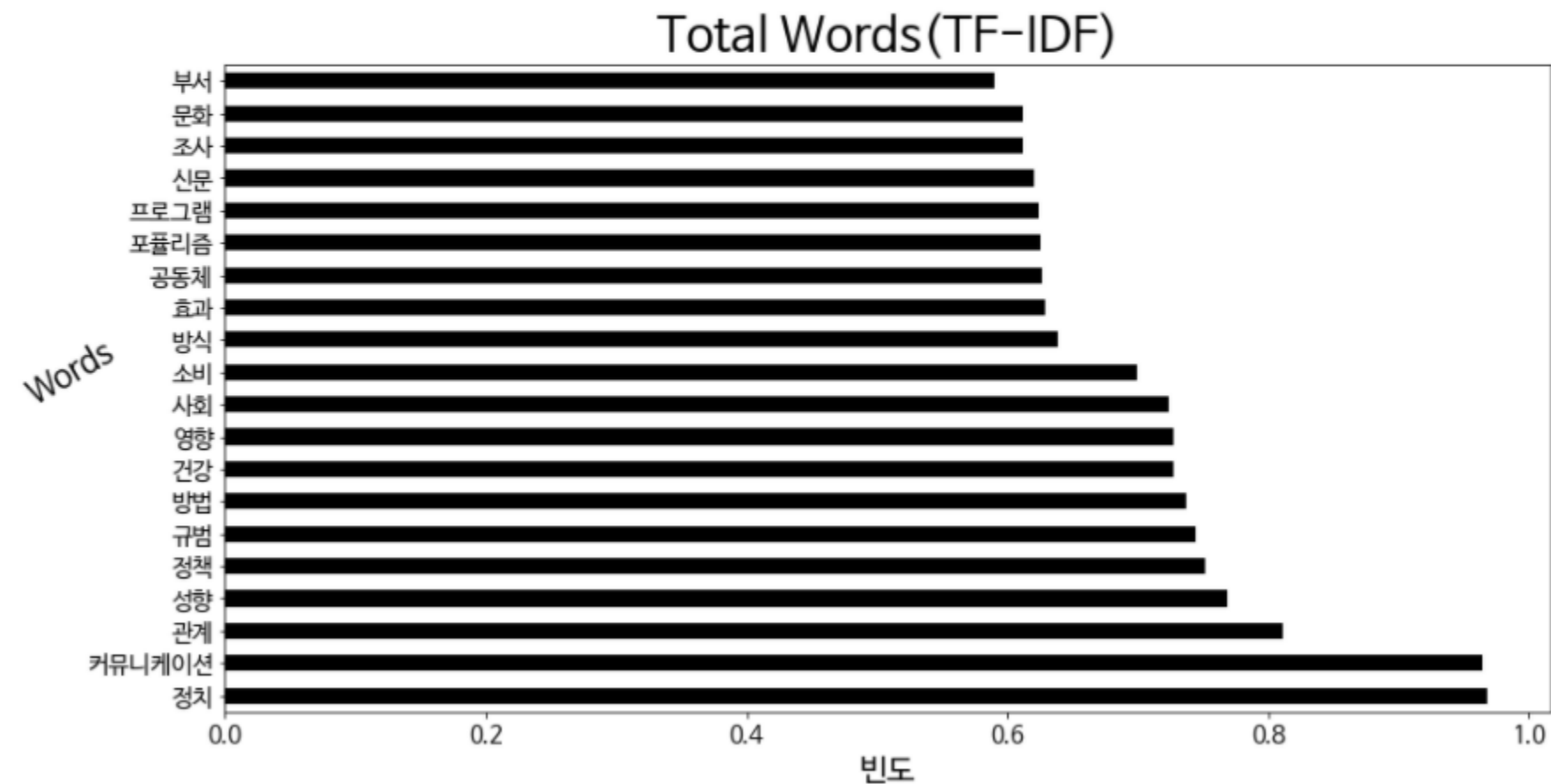


- 상위 20개의 단어를 그렸습니다.
- '사회', '커뮤니케이션', '영향', '효과', '정치'가 높게 나왔습니다.
- '영향' ~ '효과', '사회' ~ '커뮤니케이션'을 비슷한 단어로 본다면 '정치', '사회', '영향' 정도로 요약할 수 있습니다.
- 다만, 한 문서에서 많이 나온 것일 수도 있으므로 TF-IDF도 분석

- TF-IDF를 이용해서 모든 문서를 분석한 결과입니다.

```
count= tdm.sum(axis=0).sort_values(ascending=False)[0:20]
plt.figure(figsize=(15,7.5))
count.plot(kind="barh",color='black')
plt.xlabel("빈도",fontsize=20,rotation=0) ;plt.ylabel("Words",fontsize=20,rotation=30)
plt.xticks(fontsize=15) ; plt.yticks(fontsize=15)
plt.title("Total Words(TF-IDF)",fontsize=30)
```

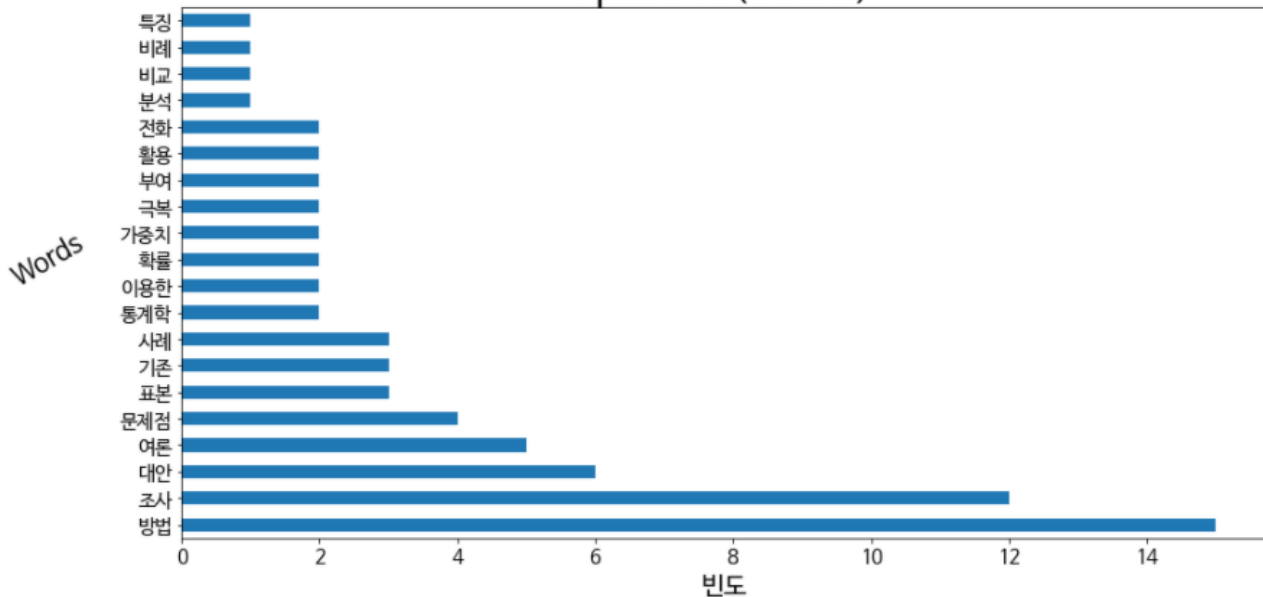
Text(0.5, 1.0, 'Total Words(TF-IDF)')



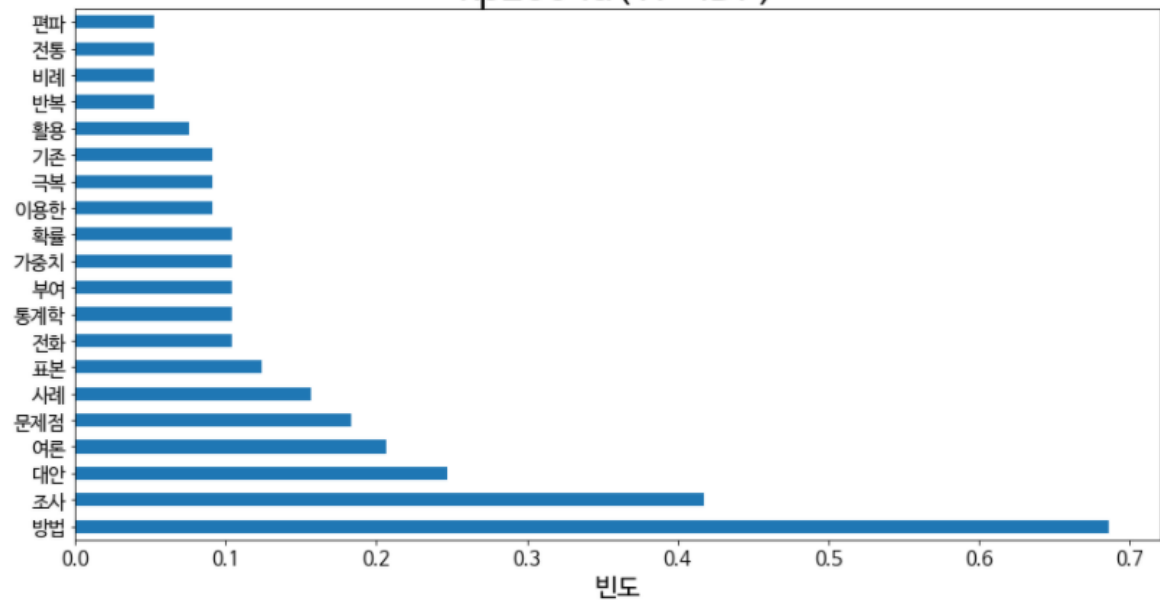
- 상위 20개의 단어를 그렸습니다.
- '사회', '영향', '효과'가 상위단어에서 순번이 떨어진 것으로 보입니다
- 이는 위와 같은 단어들이 다른 문서에서도 여러 번 쓰인 빈번한 단어라는 뜻이라고 생각합니다.
- '포퓰리즘', '규범'과 같은 단어가 생긴 것으로 보아 이는 특정한 문서에서 쓰인 단어라고 생각합니다.
- TF-IDF가 전체 문서의 특징을 잘 나타내는 것 같습니다.

한글 논문에 대한 기초통계분석

kp2004a (Count)



kp2004a (TF-IDF)



- 첫번째 문서인 'kp2004a.txt'만 대표적으로 분석 해보았습니다.
- Count, TF-IDF방법 모두 '방법', '조사', '대안', '여론'과 같은 특정한 단어를 잘 추려냈습니다.
- 실제 문서의 확인 결과 '여론 조사'에 관련한 주제였습니다.
- '특징', '비교'와 같은 단어는 다른 문서에서도 자주 사용 되는 단어이기에 TF-IDF 방식에선 보이지 않는 것 같습니다.
- 둘다 잘 분류하지만 단어수가 많아지면 TF-IDF 방식이 우세할 것으로 생각합니다.

한글 논문에 대한 기초통계분석

```
fontpath = '/usr/share/fonts/truetype/nanum/NanumGothic.ttf'  
wc=WordCloud(font_path=fontpath,relative_scaling=0.2,background_color='white')  
wc.generate_from_frequencies(dict(doc1[:30]))
```

```
plt.figure(figsize=(20,20))  
plt.imshow(wc)  
plt.axis('off')  
plt.show()
```



- Count를 이용한 Matrix를 이용하여 텍스트 클라우드를 작성 했습니다.
- '조사', '여론', '방법', '대안'과 같은 중요성이 높은 단어는 크게 나타남을 확인 했습니다.
- 텍스트 클라우드의 모양을 바꾸는 방법은 조금 더 코드를 찾아봐야 할 것 같습니다.

한글 논문에 대한 토픽모델

```
km = Komoran() # 코모란 클래스 이용
plt.rc('font', family='NanumBarunGothic') # 한글 지정
model = tp.LDAModel(k=7,alpha=0.1,eta=0.01,min_df=1) # 토픽 모델 생성, 토픽 7개
```

```
token = han_fun(file_data)
model.add_doc(Text(token))
corpus.append(file_data)
```

모델 적용

토픽 모델을 훈련 후에

```
model.train(200)
for i in range(model.k):
    res = model.get_topic_words(i, top_n=5)
    print('Topic #{}'.format(i), end='\n\t')
    print(' '.join(w for w, p in res))
```

Topic #0 효과, 건강, 시청, 프로그램, 인식
Topic #1 관계, 영향, 정치, 성향, 이용자
Topic #2 공익, 문화, 가치, 기업, 부서
Topic #3 방식, 행동, 규범, 사회, 인터넷
Topic #4 커뮤니케이션, 포퓰리즘, 정책, 변화, 현상
Topic #5 조사, 방법, 소비, 정치, 대안
Topic #6 공동체, 미디어, 분석, 소통, 행위

토픽 생성 및 상위 5개 단어

	# 1	# 2	# 3	# 4	# 5	주제
토픽 1	효과	건강	시청	프로그램	인식	건강적
토픽 2	관계	영향	정치	성향	이용자	정치와 관계
토픽 3	공익	문화	가치	기업	부서	기업의 문화(?)
토픽 4	방식	행동	규범	사회	인터넷	사회적
토픽 5	커뮤니케이션	포퓰리즘	정책	변화	현상	정책과 관련(?)
토픽 6	조사	방법	소비	정치	대안	여론
토픽 7	공동체	미디어	분석	소통	행위	SNS

한글 논문에 대한 토픽모델

```
topic_model = pd.DataFrame(index=[0,1,2,3,4,5,6])

for i in range(len(model.docs)):
    a = pd.DataFrame(model.docs[i].get_topics(), columns=['index', 'doc_'+str(i+1)])
    a = a.set_index(a.columns[0])
    topic_model = topic_model.join(a)
topic_model
```

	doc_1	doc_2	doc_3	doc_4	doc_5	doc_6	doc_7	doc_8	doc_9	doc_10	doc_11	doc_12	doc_13	doc_14	doc_15	doc_16	doc_17	doc_18	doc_19
0	0.002529	0.100250	0.042142	0.009386	0.126962	0.003335	0.055893	0.025129	0.063938	0.002724	0.636886	0.025754	0.817063	0.662605	0.002724	0.143340	0.071636	0.012375	0.042840
1	0.003483	0.041018	0.062872	0.010165	0.099304	0.205503	0.918728	0.018288	0.900422	0.187122	0.010019	0.828171	0.123386	0.320280	0.302935	0.003646	0.037960	0.177471	0.109492
2	0.001252	0.001692	0.828021	0.154739	0.020454	0.013470	0.001825	0.001070	0.001692	0.011000	0.001008	0.001577	0.009774	0.001230	0.541809	0.732912	0.001209	0.001349	0.132347
3	0.019868	0.075260	0.002141	0.060151	0.709432	0.132572	0.002840	0.062914	0.002634	0.696977	0.304566	0.002456	0.019016	0.001915	0.002100	0.077077	0.079726	0.031053	0.002840
4	0.055664	0.716729	0.002095	0.250432	0.002035	0.014335	0.002779	0.116471	0.026786	0.098566	0.001536	0.036257	0.010401	0.001874	0.002055	0.001997	0.071037	0.774140	0.603180
5	0.870506	0.001960	0.021275	0.403775	0.020665	0.628276	0.002113	0.001239	0.001960	0.001562	0.044453	0.092105	0.001388	0.010227	0.078771	0.029657	0.001400	0.001562	0.002113
6	0.046698	0.063091	0.041453	0.111351	0.021147	0.002509	0.015823	0.774887	0.002569	0.002049	0.001531	0.013680	0.018971	0.001868	0.069606	0.011370	0.737032	0.002049	0.107188

- 예시로 첫번째 문서인 'kp2004a.txt'의 경우 토픽 6에 해당하는 단어가 87%를 차지하고 있습니다.
- 토픽 6은 '여론'과 관련된 내용으로 실제로 '여론조사'와 관련된 내용으로 구성 됐습니다.
- 두번째 문서인 'kp2005a.txt'의 경우 토픽5의 '정책'에 해당하는 단어가 71.6%입니다
- 실제로 이 문서는 커뮤니케이션과 정책에 관련된 내용입니다.

한글 논문에 대한 감성분석

- 감성분석의 경우 Python에선 Keras, Tensorflow와 같은 딥러닝 패키지가 필요 해보였습니다.
- 아직까지 딥러닝 기법을 접한 적이 없고, 코드도 잘 다루지 못해 로직을 이해하는 쪽으로 공부 하였습니다.
- 예제와 참조한 사이트는 <https://devtimes.com/nlp-korea-movie-review> 입니다.

```
train_df = pd.read_csv("ratings_train.txt", "utf") # 훈련 파일 불러오기
train_df = train_df.iloc[0:500] # 양이 너무 많기 때문에 500개로
test_df = pd.read_csv("ratings_test.txt", "utf") # 테스트 파일 불러오기
test_df = test_df.iloc[0:10] # 역시 10개로
train_df = pd.read_csv("ratings_train.txt", "utf") # 훈련 파일 불러오기
train_df = train_df.iloc[0:100] # 양이 너무 많기 때문에 100개로
test_df = pd.read_csv("ratings_test.txt", "utf") # 테스트 파일 불러오기
test_df = test_df.iloc[0:50] # 역시 50개로
```

Train 데이터를 불러온 후
데이터가 너무 많아 500개로
데이터 슬라이싱

```
okt = Okt() # 토큰화를 시킬 라이브러리 로드

def tokenize(doc): # 토큰화 함수 정의
    return ['/'.join(t) for t in okt.pos(doc, norm=True, stem=True)]
```

여기선 과거 Twitter라고 불린
Okt 패키지를 사용 하였습니다.

```
okt = Okt() # 토큰화를 시킬 라이브러리 로드

def tokenize(doc): # 토큰화 함수 정의
    return ['/'.join(t) for t in okt.pos(doc, norm=True, stem=True)]
```

문서 토큰화 진행을 위해 함수화합니다

```
train_df.isnull().any() # null값 제거
train_df['document'] = train_df['document'].fillna('')

test_df.isnull().any()
test_df['document'] = test_df['document'].fillna('')
```

문서의 공란, 공백을 제거합니다

한글 논문에 대한 감성분석

토큰화 과정 토큰화 과정

```
train_docs = [(tokenize(row[1]), row[2]) for row in train_df.values]
test_docs = [(tokenize(row[1]), row[2]) for row in test_df.values]
train_docs = [(tokenize(row[1]), row[2]) for row in train_df.values]
test_docs = [(tokenize(row[1]), row[2]) for row in test_df.values]
```

문서 토큰화

```
FREQUENCY_COUNT = 100
tokens = [t for d in train_docs for t in d[0]]
text = Text(tokens, name='NMSC')
selected_words = [f[0] for f in text.vocab().most_common(FREQUENCY_COUNT)]
def term_frequency(doc):
    return [doc.count(word) for word in selected_words]
x_train = [term_frequency(d) for d, _ in train_docs]
x_test = [term_frequency(d) for d, _ in test_docs]
```

빈도를 지정한 후에 train, test 세트로 분리합니다

```
y_train = [c for _, c in train_docs]
y_test = [c for _, c in test_docs]
```

```
x_train = np.asarray(x_train).astype('float32')
x_test = np.asarray(x_test).astype('float32')
```

```
y_train = np.asarray(y_train).astype('float32')
y_test = np.asarray(y_test).astype('float32')
```

```
import tensorflow as tf

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(FREQUENCY_COUNT,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Tensorflow의 레이어의 개수 및 함수 설정

#학습 프로세스 설정

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
              loss=tf.keras.losses.binary_crossentropy,
              metrics=[tf.keras.metrics.binary_accuracy]
            )
```

학습기 설정 후 학습

#학습 데이터로 학습

```
model.fit(x_train, y_train, epochs=20, batch_size=512)
```

한글 논문에 대한 감성분석

```
results = model.evaluate(x_test, y_test)
```

```
2/2 [=====] - 0s 4ms/step - loss: 0.6325 - binary_accuracy: 0.6800
```

모델 생성

```
review = "아주 재미 있어요"
```

```
token = tokenize(review)
```

```
tf = term_frequency(token)
```

```
data = np.expand_dims(np.asarray(tf).astype('float32'), axis=0)
```

```
float(model.predict(data))
```

```
0.5598297119140625
```

Test Data를 이용하여 모델의 성능을 검증
(Data 수를 줄여서 Train한 결과.
그닥, 좋지 못한 성능을 보입니다)

```
def predict_review(review):
```

```
    token = tokenize(review)
```

```
    tfq = term_frequency(token)
```

```
    data = np.expand_dims(np.asarray(tfq).astype('float32'), axis=0)
```

```
    score = float(model.predict(data))
```

```
    if(score > 0.5):
```

```
        print(f"{review} ==> 긍정 ({round(score*100)}%)")
```

```
    else:
```

```
        print(f"{review} ==> 부정 ({round((1-score)*100)}%)")
```

모델을 토대로 감성분석 예측기 생성

```
predict_review("재미 정말 없어요")
```

```
재미 정말 없어요 ==> 부정 (54%)
```

```
predict_review(review)
```

```
아주 재미 있어요 ==> 긍정 (56%)
```

임의의 말을 넣어 테스트합니다

[참조한 블로그에선 많은 데이터셋을 활용해 잘 분석하는 결과를 확인했습니다.]