프로젝트	한국어 문서 추출 요약
파일명	text_extraction_TFIDF.ipynb
용도	TF, TF-IDF 단어 빈도 및 중요도 분석

코드

```
import json
import math

# json 파일 로딩
data = []
for line in open('/content/drive/MyDrive/빅데이터/문서 추출요약/train.jsonl', 'r',
encoding='utf-8'):
    data.append(json.loads(line))

# TF, TF-IDF 함수 선언
def tf(t, d):
    return d.count(t)

def idf(N, t, docs):
    df = 0
    for doc in docs:
        df += t in doc
    return log(N/(df + 1))

def tfidf(N, t, d, docs):
    return tf(t,d) * idf(N, t, docs)


# 전처리 함수
def CleanText(article):
    article = ''.join(re.findall('[가-힣a-zA-Z0-9 ]',  article) )
    return article.strip(' ')

# TF, TF-IDF 분석을 위한 데이터 정리
clean_4_TF = []
for i in range(len(clean_article)):
    clean_4_TF.append(' '.join(clean_article[i]))
```

```
# 각 인덱스 별 TF 분석
# TF 분석을 위해 space 단위로 vocab 생성
vocab = []
for i in range(len(clean_4_TF)) :
        vocab.append(clean_4_TF[i].split())

# TF 분석
result_tf = []
TF_list = []
tf_dic = {}
for i in range(len(clean_4_TF)):
    d = clean_4_TF[i]
    for j in range(len(vocab[i])):
        t = vocab[i][j]
        result_tf.append(tf(t,d))
        tf_dic[vocab[i][j]] = result_tf[j]
    TF_list.append(tf_dic)

# TF-IDF 분석
result_tfidf = []
tfidf_dic = {}
TFIDF_list = []
for i in range(len(clean_4_TF)):
    d = clean_4_TF[i]
    for j in range(len(vocab[i])):
        t = vocab[i][j]
        result_tfidf.append(tfidf(len(clean_4_TF),t,d,clean_4_TF))
        tfidf_dic[vocab[i][j]] = result_tfidf[j]
    TFIDF_list.append(tf_dic)
    break
```

프로젝트	한국어 문서 추출 요약
파일명	text_extraction.ipynb
용도	데이터 전처리 핵심 문장 추출 및 저장 성능 확인

코드

```
import json
import pandas as pd
import re
import numpy as np
import math
from collections import Counter
from scipy.sparse import csr_matrix
from sklearn.preprocessing import normalize
from math import log
from google.colab import drive
from konlpy.tag import Komoran, Okt, Kkma, Mecab

# 키워드 추출을 위해 단어 그래프 생성하는 함수
def scan_vocabulary(sents, tokenize, min_count=2):
    counter = Counter(w for sent in sents for w in tokenize(sent))
    counter = {w:c for w,c in counter.items() if c >= min_count}
    idx_to_vocab = [w for w, _ in sorted(counter.items(), key=lambda x:-x[1])]
    vocab_to_idx = {vocab:idx for idx, vocab in enumerate(idx_to_vocab)}
    return idx_to_vocab, vocab_to_idx

# 핵심 문장 선택을 위한 PageRank(Graph ranking 알고리즘) 함수
# PageRank : 문장간 유사도를 기반으로 핵심 문장의 점수 생성.
# 다른 문장들과 유사도가 높을수록 핵심 문장
def pagerank(x, df=0.85, max_iter=30):
    assert 0 < df < 1

    # initialize
    A = normalize(x, axis=0, norm='l1')
    R = np.ones(A.shape[0]).reshape(-1,1)
    bias = (1 - df) * np.ones(A.shape[0]).reshape(-1,1)

    # iteration
    for _ in range(max_iter):
```

```
        R = df * (A * R) + bias
    return R

# 그래프 각 파라미터를 함수들에 적용
# scan_vocabulary 함수를 통해 문장별 단어 토큰화를 진행
# 문장 간 단어(토큰)의 중첩 정도를 csr_matrix를 통해 반환
def sent_graph(sents, tokenize, similarity, min_count=2, min_sim=0.1):
    _, vocab_to_idx = scan_vocabulary(sents, tokenize, min_count)
    tokens = [[w for w in tokenize(sent) if w in vocab_to_idx] for sent in sents]
    rows, cols, data = [], [], []
    n_sents = len(tokens)
    for i, tokens_i in enumerate(tokens):
        for j, tokens_j in enumerate(tokens):
            if i >= j:
                continue
            sim = similarity(tokens_i, tokens_j)
            if sim < min_sim:
                continue
            rows.append(i)
            cols.append(j)
            data.append(sim)
    return csr_matrix((data, (rows, cols)), shape=(n_sents, n_sents))

# 문서 간 혹은 문장 간 유사도를 측정하기 위한 함수
def cosine_sent_sim(s1, s2):
    if (not s1) or (not s2):
        return 0
    s1 = Counter(s1)
    s2 = Counter(s2)
    norm1 = math.sqrt(sum(v ** 2 for v in s1.values()))
    norm2 = math.sqrt(sum(v ** 2 for v in s2.values()))
    prod = 0
    for k, v in s1.items():
        prod += v * s2.get(k, 0)
    return prod / (norm1 * norm2)

# 문서 간 혹은 문장 간 유사도를 측정하기 위한 함수
def textrank_sent_sim(s1, s2):
    n1 = len(s1)
    n2 = len(s2)
```

```
    if (n1 <= 1) or (n2 <= 1):
        return 0
    common = len(set(s1).intersection(set(s2)))
    base = math.log(n1) + math.log(n2)
    return common / base

# 핵심 문장 인덱스 추출을 위한 함수
# sent_graph를 문장 간 유사도를 나타내는 행렬 생성
# 행렬에 pagerank 알고리즘를 사용하여 문장 간 중요 정도 점수 생성
# 점수가 높은 상위 3개의 문장만 반환
def textrank_keysentence(sents, tokenize, min_count, min_sim, similarity, df=0.85,
max_iter=30, topk= 3 ):
    g = sent_graph(sents, tokenize,  similarity ,min_count, min_sim )
    R = pagerank(g, df, max_iter).reshape(-1)
    idxs = R.argsort()[-topk:]
    keysents = [(idx) for idx  in reversed(idxs)]
    return keysents

kkma = Kkma()
komoran = Komoran()
okt = Okt()
mecab = Mecab()

# 토크나이저
def komoran_tokenize(sent):
    words = mecab.pos(sent, join=True)
    return words

# 전처리 함수
def CleanText(article):
    article = re.sub('"', ' ', article)
    bracket = re.findall(r'₩([^)]*₩)', article )
    for i in bracket:
        word = i.strip('()')
        if word.isupper():
            end_index = article.find(i)
            word_len = article[end_index:0:-1].find(' ')
            start_index = end_index - word_len +1
            origin = article[start_index : end_index]
            article = article[:end_index+len(i)] + article[end_index+len(i):].replace(word,
```

```
origin)
            else:
                if '이하' in word:
                    word = word[3:]
                    n_space = word.count(' ')
                    end_index = article.find(word)-4
                    range_candidate = article[end_index-30:end_index].split(' ')[::-1]
                    origin = ' '.join(range_candidate[:n_space+1][::-1])
                    article = article[:end_index+len(i)] +
article[end_index+len(i):].replace(word, origin)
            article = article.replace(i,'')

        article = ''.join(re.findall('[가-힣a-zA-Z0-9 ]', article) )

        return article.strip(' ')

# 사용할 데이터 정리
clean_article = []
ori_article = []

id_list = []
for i in range(len(data)):
    s1 = []
    s2 = []
    id_list.append(data[i]['id'])
    for j in range(len(data[i]['article_original'])):
        s1.append(data[i]['article_original'][j])
        s2.append(CleanText(data[i]['article_original'][j]))
    ori_article.append(s1)
    clean_article.append(s2)

# 핵심 문장의 인덱스 추출
idx_list = []
for sents in clean_article:
    keysents = textrank_keysentence(sents,
                                    komoran_tokenize,
                                    min_count= 2,
                                    min_sim = 0.1,
                                    similarity = textrank_sent_sim,
                                    topk=3
```

```
                                        )
    idx = [sent_idx for sent_idx in keysents]
    idx_list.append(idx)

# 성능 확인을 위한 데이터 정리
ori_idx_list = []
for i in range(len(data)):
    ori_idx_list.append(data[i]['extractive'])

result = pd.DataFrame(
                    {'ori_index' : ori_idx_list,
                     're_index' : idx_list},
                    )

compare_index_list = []
for i in range(len(result)):
    result_sum = 0
    for j in range(len(result['ori_index'][i])):
        if result['ori_index'][i][j] == result['re_index'][i][j] :
            result_sum += 1
        else :
            result_sum += 0
    compare_index_list.append(result_sum)

result = pd.DataFrame(
                    {'ori_index' : ori_idx_list,
                     're_index' : idx_list,
                     'compare_index' : compare_index_list}
                    )

# 성능 확인
result['compare_index'].sum()

# 추출한 인덱스를 원문에 적용하여 핵심 문장 정리
result = []
for i in range(len(idx_list)) :
    ori_str = []
    for j in range(3) :
        ori_str.append(ori_article[i][idx_list[i][j]])
    result.append('₩n'.join(ori_str))
```

```
result = pd.DataFrame(
                    {'id' : id_list,
                     'summary' : result},
                    )
```