프로젝트	한국어 문서 추출 요약
파일명	parameter_test.ipynb
용도	핵심 문장 추출 함수의 최적 파라미터 도출

코드

```python
import json
import pandas as pd
import re

from collections import Counter
# from collections import defaultdict
from scipy.sparse import csr_matrix
import numpy as np
from sklearn.preprocessing import normalize
import math


# 전처리 함수 선언
# ( ) 안 단어가 특정 단어의 동의어를 뜻하는 경우 => 동의어들을 한 단어로 통일
# ( ) 안 단어가 부연 설명일 경우 => 제거
def preprocess_sentence(article):
    article = re.sub('"',"'", 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)
        # 괄호는 다 제거
        article = article.replace(i,'')
    article = ''.join(re.findall('[ 가-힣a-zA-Z0-9]',  article) )
    if len(article.strip(' ')) > 1:
        return article
    else:
        return ''
```

# 토크나이저 선언

```
!pip install konlpy
from konlpy.tag import Komoran

# 문장을 입력받아 토큰화된 단어 리스트로 반환
komoran = Komoran()
def komoran_tokenize(sent):
    words = komoran.pos(sent, join=True)
    words = [w for w in words if ('/NN' in w or '/XR' in w or '/VA'
in w or '/VV' in w)]
    return words
```

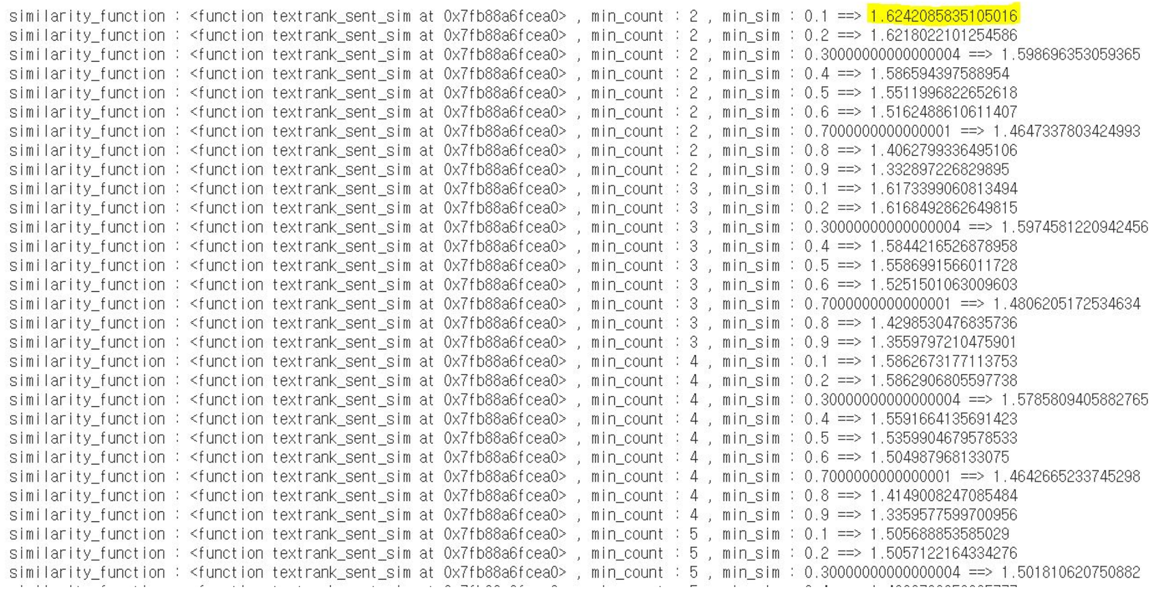
# 문장들을 입력받아 설정된 최소 등장 횟수 이상의 단어들만 반환

```
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
```

# 문장간 유사도를 기반으로 중요 문장 점수를 도출하는 pagerank 알고리즘 구현

```
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

# 유사도 판단 함수 1
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

# 유사도 판단 함수 2
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)

# 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=0.3):
    _, 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))

# 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:]
    key_index = [ idx for idx in reversed(idxs)]
    return key_index

input_file_name = './train.jsonl'

# train 데이터를 통해 TextRank 함수의 파라미터 값들의 최적값을 찾음.
# textrank 함수가 추출한 함수가 어느 정도 정답 index와 겹치는지를 확인하여
평균을 냄
for similar in [textrank_sent_sim , cosine_sent_sim ]:
    for mc in range(2,7):
        for ms in np.arange(0.1,1,0.1):
            with open(input_file_name, 'r', encoding = 'utf-8',
newline = '') as input_file:
correct_list = []
i = 0
for line in input_file:
    line = json.loads(line)
    id_num, sents , _ ,answer_index = list(line.values())[1:]
    preprocessed = [ preprocess_sentence(sent) for sent in sents ]
    key_index = textrank_keysentence(preprocessed , komoran_tokenize
, mc , ms , similar )
    correct = len([ind for ind in key_index if ind in answer_index])
    correct_list.append(correct)

print(f'similarity_function : {similar} , min_count : {mc} , min_sim
: {ms} ==> {sum(correct_list)/len(correct_list)}')
```

결과물



```
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.1 ==> 1.6242085835105016
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.2 ==> 1.6218022101254586
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.30000000000000004 ==> 1.598696353059365
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.4 ==> 1.586594397588954
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.5 ==> 1.5511996822652618
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.6 ==> 1.5162488610611407
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.7000000000000001 ==> 1.4647337803424993
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.8 ==> 1.4062799336495106
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 2 , min_sim : 0.9 ==> 1.332897226829895
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.1 ==> 1.6173399060813494
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.2 ==> 1.6168492862649815
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.30000000000000004 ==> 1.5974581220942456
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.4 ==> 1.5844216526878958
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.5 ==> 1.5586991566011728
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.6 ==> 1.5251501063009603
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.7000000000000001 ==> 1.4806205172534634
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.8 ==> 1.4298530476835736
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 3 , min_sim : 0.9 ==> 1.3559797210475901
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.1 ==> 1.5862673177113753
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.2 ==> 1.5862906805597738
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.30000000000000004 ==> 1.5785809405882765
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.4 ==> 1.5591664135691423
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.5 ==> 1.5359904679578533
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.6 ==> 1.504987968133075
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.7000000000000001 ==> 1.4642665233745298
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.8 ==> 1.4149008247085484
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 4 , min_sim : 0.9 ==> 1.3359577599700956
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 5 , min_sim : 0.1 ==> 1.505688853585029
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 5 , min_sim : 0.2 ==> 1.5057122164334276
similarity_function : <function textrank_sent_sim at 0x7fb88a6fcea0> , min_count : 5 , min_sim : 0.30000000000000004 ==> 1.501810620750882
```