20181695 조익현

입력파일 이름은 input.txt 로..

Mecab, Hannanum, Okt 사용

멀티 프로세싱 사용

Word2Vec 워드 임베딩 사용

형태소 분석 결과 파일 입출력

Word2Vec 워드 임베딩 학습 결과 저장

속도

Mecab >>>>>>>> Okt > Hannanum >>>>>> kkma

소스 코드

```python
import re
from konlpy.tag import Kkma
from konlpy.tag import Mecab
from konlpy.tag import Hannanum
from konlpy.tag import Okt
from multiprocessing import Process, Queue
import gensim
import numpy as np
import time

konlpy = {"Kkma":Kkma,"Mecab":Mecab,"Okt":Okt,"Hannanum":Hannanum}

def make_dictionary(sentence):
    cnt = {}
    for c in sentence:  # 문장의 최소단위 (음절은 한글자. 어절은 공백단위와 특수문자)
        if c in cnt.keys():
            cnt[c] += 1
        else:
            cnt[c] = 1
    return cnt

def sort_result(result, N):
    if len(result) > N:
        result = sorted(result, key= lambda sentence : sentence[0], reverse=True)[:N]

    return set(result)
```

make_dictionary – 워드 임베딩 하지 않을 경우 단어 빈도를 세어 딕셔너리를 만드는 함수
sort_result – 출력할 유사도 상위 N개를 추려내는 함수

```python
def cos_sim(A,B):
    return np.dot(A,B)/(np.linalg.norm(A)*np.linalg.norm(B))

def find_equal_by_cos(word2vec, lex_name,N,s1,s2,corpus):
    start_time = time.time()

    result = set()

    A = np.zeros(200)
    for i in s1:
        if i in word2vec:
            A = A + word2vec[i]

    i = 0
    for B_ in s2:
        if i % 50000 == 0:
            print(lex_name + " (word_embed): 현재 {} 개 비교".format(i))
        original_B = corpus[i]
        i+=1

        B = np.zeros(200)
        for b in B_:
            if b in word2vec:
                B = B + word2vec[b]

        similar_rate = cos_sim(A,B)
        result.add((similar_rate,original_B))
        result = sort_result(result,N)

    print(lex_name + " : 비교 완료")
    end_time = time.time()
    total_time = end_time - start_time

    return total_time,result
```

Cos_sim – 밀집벡터 둘을 cos하여 유사도를 출력하는 함수

Find_equal_by_cos – word2vec을 이용해 워드임베딩된 단어들을 문장에서 찾아 두 문장의 유사도를 구하는 함수

```python
def find_equal(lex_name, N, s1, s2, corpus):
    start_time = time.time()

    result = set()
    s1 = make_dictionary(s1)
    i = 0
    for s2_cnt in s2:
        if i % 50000 == 0:
            print(lex_name + " : 현재 {} 개 비교".format(i))

        original_s2 = corpus[i]
        i += 1
        s1_cnt = s1
        s2_cnt = make_dictionary(s2_cnt)

        if sum(s1_cnt.values()) > sum(s2_cnt.values()):
            s1_cnt,s2_cnt = s2_cnt,s1_cnt   # s1_cnt의 형태소의 총개수가 항상 더 작다

        cnt = 0     # 공통음절 cnt

        for k in s1_cnt.keys():
            if k in s2_cnt.keys():
                cnt += s1_cnt[k] > s2_cnt[k] and s2_cnt[k] or s1_cnt[k]

        similar_rate = cnt / sum(s1_cnt.values())
        result.add((similar_rate, original_s2))
        result = sort_result(result, N)

    print(lex_name + " : 비교 완료")
    end_time = time.time()
    total_time = end_time - start_time

    return total_time, result
```

Find_equal – 워드임베딩 하지 않을 경우 같은 단어의 개수를 세어 유사도를 구하는 함수

```python
def extract_morphs(lexical_analyzer, lex_name, corpus, s1):
    start_time = time.time()

    s1 = lexical_analyzer.morphs(s1)
    s2 = list()

    i = 0
    for line in corpus:
        if i % 50000 == 0:
            print(lex_name + " : 현재 형태소 추출 {} 개".format(i))
        i+=1
        if not line: break
        try:
            s2.append(lexical_analyzer.morphs(line))
        except Exception as err:       # 일부 문장에서 단어중 추출이 안되는 경우가 있음. 그 문장은 비교에서 배제
            print(lex_name + " : 분석 불가 문장 - " + line)
            continue
    print(lex_name + " : 형태소 추출 완료")
    end_time = time.time()
    total_time = end_time - start_time

    return total_time, s1, s2
```

Extract_morphs – 형태소 추출기로 형태소를 추출하는 함수

```python
def run_calc(lex_name, corpus, sentence, N, w2v, lex, message):  # w2v : 0 = 사용안함 | 1 = 이미 train한 가중치 사용 | 2 = 지금 train하기
    result_info = ""
    lexical_anayzer = konlpy[lex_name]()
    time1 = 0
    s1 = sentence
    s2 = corpus
    start_time = time.time()
    if lex > 0:
        try:
            tokenized_file = open(lex_name+"_tokenized.txt","r")
            tmp = tokenized_file.read().strip().split('\n')
            tokenized_file.close()
        except FileNotFoundError as err:
            lex = 0
            print("형태소 추출 파일이 없습니다. 추출을 실행합니다.")
        else:
            if len(tmp) < lex * 10000:
                lex = 0
                print("형태소 추출양이 부족합니다. 추출을 실행합니다.")
            else:
                s1 = lexical_anayzer.morphs(s1)
                s2 = [i.split(' ') for i in tmp]

    if lex == 0:
        time1, s1, s2 = extract_morphs(lexical_anayzer, lex_name, corpus, sentence)
        result_info = result_info + "형태소 추출 걸린 시간 : {:.2f} 초\n".format(time1)

        with open(lex_name + "_tokenized.txt","w") as f:
            for i in s2:
                f.write(" ".join(i) + "\n")
```

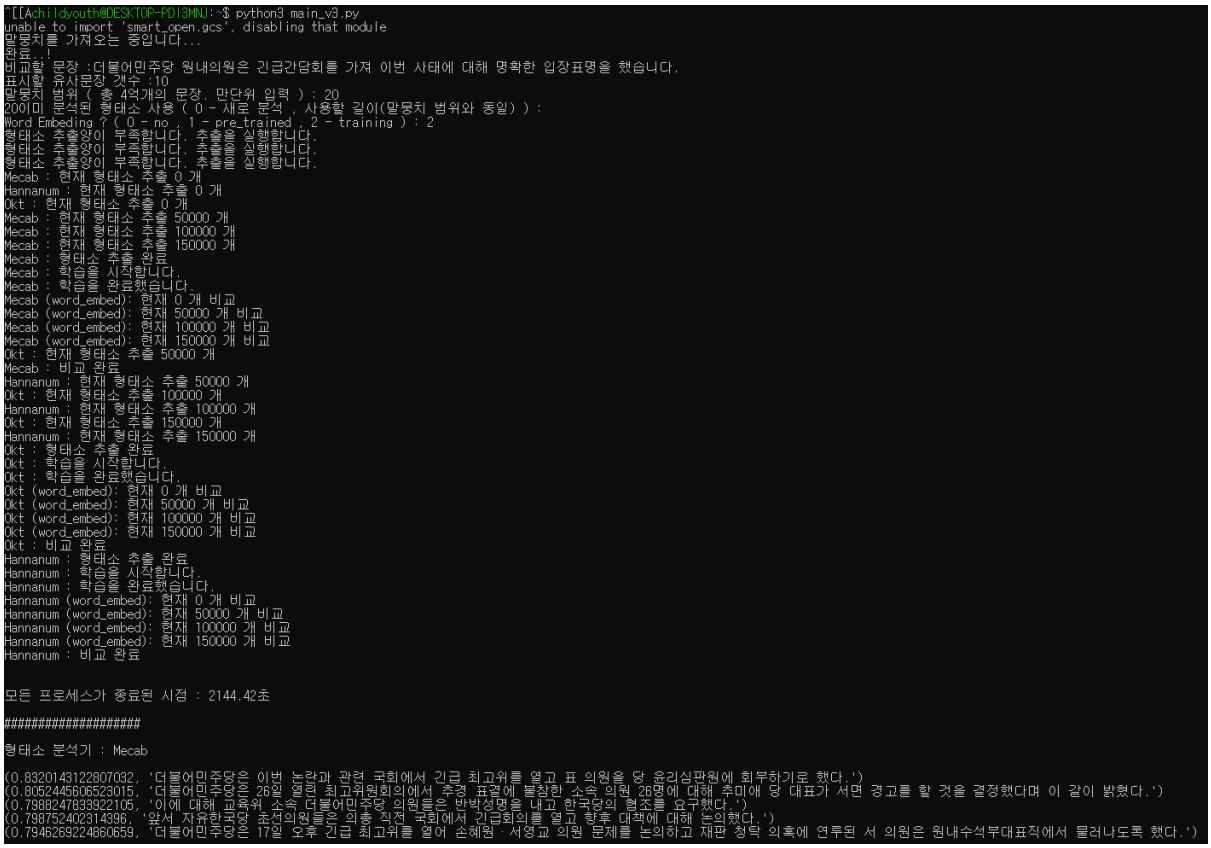
```
    if w2v == 0:
        time2, results = find_equal(lex_name, N, s1, s2, corpus)

    elif w2v == 1:
        try:
            word2vec = gensim.models.KeyedVectors.load_word2vec_format(lex_name + "_word2vec_weight")
            time2, results = find_equal_by_cos(word2vec, lex_name, N, s1,s2, corpus)
        except FileNotFoundError as err:
            w2v = 2
            print(lex_name + " : 사전 학습된 w2v 가 없습니다.")

    if w2v == 2:
        word_embed_time1 = time.time()
        print(lex_name + " : 학습을 시작합니다.")

        word2vec = gensim.models.Word2Vec(sentences=s2, size=200, window=3, min_count=2, workers=4,
                                          sg=0)  # sg=1 과 속도비교 해보기
        word2vec.wv.save_word2vec_format(lex_name + "_word2vec_weight")
        print(lex_name + " : 학습을 완료했습니다.")
        word_embed_time = time.time() - word_embed_time1
        time2, results = find_equal_by_cos(word2vec,lex_name,N,s1,s2, corpus)



    results = sorted(results, key=lambda sentence: sentence[0], reverse=True)
    end_time = time.time()

    if w2v == 2:
        result_info = result_info + "워드 임베딩 학습에 걸린 시간 : {:.2f} 초₩n".format(word_embed_time)

    result_info = result_info + "순위 비교 걸린 시간 : {:.2f} 초₩n".format(time2)
    result_info = result_info + "프로세스 완료까지 걸린시간 : {:.2f}초".format(end_time - start_time)

    result_type = "형태소 분석기 : " + lex_name
    message.put([result_type,results,result_info])
```

Run_calc – 멀티프로세싱 할 함수. 형태소 분석기마다 프로세스로 실행
다시 형태소 분석기를 실행할지, 워드 임베딩을 학습시킬지 입력받아 파일입출력하여 관련 일을 수행하고, 형태소 분석기별 유사도를 측정하는 프로세스의 main함수

```python
def MAIN():
    f_in = open("input.txt", "r",encoding='UTF8')
    print("말뭉치를 가져오는 중입니다...")
    corpus = f_in.read().strip().split('₩n')[:-1]
    print("완료..!")
    f_in.close()

    s1 = (input("비교할 문장 :")).strip()
    N = int(input("표시할 유사문장 갯수 :"))
    corp_len = int(input("말뭉치 범위 ( 총 4억개의 문장. 만단위 입력 ) : "))
    corpus = corpus[:corp_len+10000]
    lex = int(input("이미 분석된 형태소 사용 ( 0 - 새로 분석 , 사용할 길이(말뭉치 범위와 동일) ) : "))
    w2v = int(input("Word Embeding ? ( 0 - no , 1 - pre_trained , 2 - training ) : "))
    #
    # word2vec = gensim.models.Word2Vec.load('ko.bin')

    # a = word2vec.wv.most_similar("강아지")
    # print(a)

    result = Queue()

    #ps_kkma = Process(target=run_calc, args=("Kkma",corpus,s1,N,w2v, lex,result))   # 너무 느려서 사용하지 않을것
    ps_mecab = Process(target=run_calc, args=("Mecab",corpus,s1,N,w2v, lex,result))
    ps_okt = Process(target=run_calc, args=("Okt",corpus,s1,N,w2v, lex,result))
    ps_hannanum = Process(target=run_calc, args=("Hannanum",corpus,s1,N,w2v, lex,result))

    main_t1 = time.time()

    #ps_kkma.start()
    ps_mecab.start()
    ps_okt.start()
    ps_hannanum.start()

    #ps_kkma.join()
    ps_mecab.join()
    ps_okt.join()
    ps_hannanum.join()

    print("₩n₩n모든 프로세스가 종료된 시점 : {:.2f}초".format(time.time()-main_t1),end="₩n₩n")

    result.put(None)
    while True:
        r = result.get()
        if r is None:
            break
        else:
            print("#"*20)
            print()
            print(r[0]+"₩n")
            for i in r[1]:
                print(i)
            print(r[2] + "₩n₩n")
            print("#"*20)


if __name__ == "__main__":
    MAIN()

    try:
        print()
    except Exception:
        print()
```

Main – 메인 함수
형태소 분석기별 프로세스를 생성하고 메인 프로세스에 join한다. 모든 프로세스가 완료되면 완료된 순서로 프로세스의 출력결과를 보여준다

실행 결과



형태소 분석과 워드 임베딩을 처음 실행할 경우 형태소 분석과 워드 임베딩 학습 경과를 출력하고 시간을 출력한다.

말뭉치를 전부 다 사용하려면 시간이 너무 오래 걸리기 때문에 말뭉치를 얼마나 자를지 입력받아 사용한다.

Word2vec 으로 워드 임베딩을 하면 유사도를 똑 같은 단어가 아니더라도 추출할 수 있다. 이는 word2vec이 CBOW 혹은 Skip-Gram 방식으로 단어별 방향벡터를 밀집벡터로 만들어주기 때문이다. 주변 window개 만큼의 단어에 대해 주변 점수를 갱신하거나 본인의 점수를 갱신해 비슷한 위치로 사용되는 단어들이 비슷한 방향을 바라보기 때문에 같은 단어가 아니더라도 유사성이 높게 나올 수 있다. 그 예로 지금 위의 실행결과는

Mecab의 네번째 문장은 입력문장인 더불어민주당이 아닌 자유한국당인데도 높게 측정되는 모습을 확인할 수 있다. 같은 정당으로서 비슷한 방향벡터를 가지기 때문이다.

실행 속도를 살펴보면 Mecab이 가장 빠르게 형태소를 추출해냈고, 한참 뒤에 Okt, Hannanum이 뒤를 이었다.

위의 실행결과는 word2vec의 min_count를 2로 주었을 때의 실행결과다. Min_count는 단어의 최소 빈도수를 의미하는데, 그렇다면 min_count를 1로 주면 어떤 결과가 나올까?

같은 문장에 대해 min_count를 1로 주고 다시 임베딩을 학습한 결과이다. 이전의 실행결과와 크게 다른 느낌을 받을 수 없다. 하지만 임베딩 학습시간은 Mecab을 제외하고 모두 30~ 40초 가량 더 늦게 완료되었다.

Word_count = 2

Word_count = 1

"당" 이라는 글자와 가장 유사한 단어를 찾아보았을 때 유사도에 자잘한 변화가 있음을 확인 할 수 있다. 데이터의 내용이 많아질수록 영향도 커질 테니 word_count는 '적당히 잘' 줘야 한다.

형태소 분석기들 마다 속도차이가 나는 이유에 대해서는 형태소 분석결과를 보면 생각할 수 있다.

Mecab

Okt

Hannanum

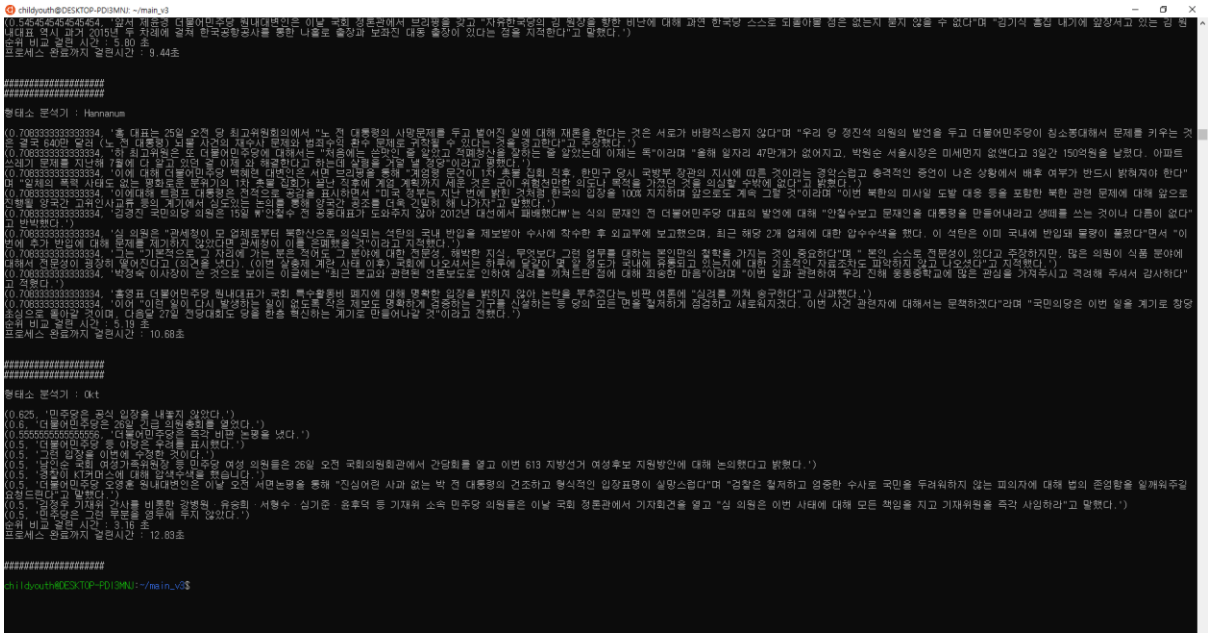
우선 Mecab과 Okt에서 가장 첫번째 글자를 확인하면 '뒤이어' 인데, Mecab은 '이어'를 한 단어로 생각하지 않고 '이' '어' 로 나누었지만, Okt는 '뒤' '이어' 로 잘 나눈 것을 확인할 수 있다. 또한 '국민의당' 이라는 단어를 Mecab은 '국민' '의' '당' 이지만 Okt는 '국민의당' 으로 합성어를 잘 잡아냈다. 이처럼 Mecab은 다른 두 형태소 분석기에 비해 단어를 깊게 생각해 나눠내지 않고 사전이 상대적으로 작다고 볼 수 있다.

하지만 그렇다고 성능에 문제가 있다고 볼 수 는 없다. 국민의당이 어떤 때는 국민의 당 이 맞을 수도 있기 마련이다.

Kkma나 Hannanum은 가장 눈에 띄는 점은 받침인 'ㄴ' 과 'ㄹ' 까지 나눈다는 것이다. 이는 당연

하게도 실행속도를 늦추는 원인이며, 사전을 더욱 깊게 바라본다는 뜻이기도 하다.

이것도 마찬가지로 성능에 문제가 있다고 보기는 어려우며, 모든 형태소 분석기는 각자의 특색에 따라 적절한 용도로 사용하면 된다.



마지막으로 워드 임베딩을 하지 않고 실행하는 경우인데, 워드 임베딩과 크게 느껴지는 차이점은 실행결과인 유사도있는 문장이 모두 입력문장의 단어가 포함되어 있다는 사실이다.

워드임베딩을 사용하면 일치하는 단어가 없는 문장이 유사함을 찾아 낼 수 있고, 이는 검색엔진 등에서 비슷한 단어를 찾아 추천하는 서비스 등을 구현할 수 있을 것 같다.