빅데이터 최신기술
한글 문장의 유사도 계산

소프트웨어 공학과

20153235
조정근

1. 실험 환경

OS : Ubuntu 18.04 LTS

CPU : Pentium E5800

Memory : 4GB

2. 성능분석

성능 분석은 몇 개의 샘플 문장을 비교하는 것으로 대체하였다.

- **kma**

```
C:₩Users₩barunpuri₩Downloads₩KLT2010-TestVersion₩EXE>kma.exe -s
------------------------------------------------------------------
     Welcome to KLT(Korean Language Technology) version 3.0.0a
  <<< BEFORE USING IT, YOU SHALL READ THE FILE LICENSE.TXT >>>
------------------------------------------------------------------
아버지가방에들어가신다
아버지가방에들어가신다
        (N "아버지가방에들어가신다")< :60>
아버지가 방에 들어가신다
아버지가
        (N "아버지")<N:20> + (j "가")<1>
방에
        (N "방")<NU:27> + (j "에")
들어가신다
        (V "들어가")<lg:22> + (f "시") + (e "ㄴ다")<13>
```

우선 kma를 사용하여 형태소 분석을 진행하였다. kma는 띄어쓰기가 된 문장에 대해서는 제대로 분석하였지만, 띄어쓰기가 없는 문장에 대해서는 분석하지 못하였다. 또한 ubuntu 환경에서 실행하고 싶었지만, 해당하는 파일을 찾기 어려워 이후 실험에서는 제외되었다.

<실행 방법>

```
~$ python3 komoran_test.py < test_input.txt > komoran_test.txt
~$ python3 khaiii_test.py < test_input.txt > khaiii_test.txt
~$ python3 mecab_test.py < test_input.txt > mecab_test.txt
```

test_input.txt 파일에는 비교하기 위한 10개의 문장이 들어있고, 각각의 python파일을 실행하여 --_test.txt파일에 저장하였다.

<소스코드>

- **komoran**

```
from konlpy.tag import Komoran
komo = Komoran()

for i in range(10):
        s = str(input())

        print( komo.pos(s) )
```

Konlpy에 포함된 모듈중 하나인 komoran을 사용하였다.

- **Mecab**

```
import MeCab
m = MeCab.Tagger('-d ./mecab-ko-dic-2.1.1-20180720/')

for i in range(10):
        s = str(input())
        res = m.parse(s)

        res = res[:-5].split('₩n')
        res = [word.replace('₩t', '/') for word in res]
        res = [word.split(',')[0] for word in res]

        print(res)
```

Mecab역시 konlpy에 포함되어 있지만, 윈도우 환경에서는 실행이 불가능 하여 우분투 환경에서 진행하였고, konlpy를 통해서 수행하지 않고 Mecab을 직접 python에서 호출하여 사용하였다.

- **Khaiii**

```
from khaiii import KhaiiiApi
api = KhaiiiApi()


for i in range(10):
        s = str(input())

        res = []
        for word in api.analyze(s):
                res.append(str(word))
        res = [word.split('₩t')[1] for word in res]

        print(res)
```

Khaiii는 kakao에서 개발한 형태소 분석기로 mecab과 마찬가지로 window환경에서 실행이 불가능하여 우분투 환경에서 진행하였다. Mecab처럼 python에 연결하여 사용할 수 있다.

<실행 결과>

1. 너무기대안하고갔나재밌게봤다　　(띄어쓰기가 없는 문장)

Komoran	MeCab	Khaiii
너무/MAG	너무/MAG	너무기대안/MAG
기대/NNG	기대/NNG	하/XSV
안/NNG	안/MAG	고/EC
하/XSV	하/VV	가/VX
고/EC	고/EC	았/EP
가/VX	갔/VV+EP	나/EC
았/EP	나/EC	재밌/VA
나/EC	재밌/VA	게/EC
재밌/VA	게/EC	보/VV
게/EC	봤/VX+EP	았/EP
보/VX	다/EC	다/EC
았/EP		
다/EC		

Komoran과 MeCab은 띄어쓰기가 없는 문장에 대해서도 좋은 성능을 보였다. Khaiii 는 대부분의 경우에서는 잘 해냈지만, 특정 경우(너무 기대 안 하고 -> 너무기대안 하고)에 대해서는 제대로 분류해내지 못하였다.

2. 굉장히잘만든수작지루할틈이없음　(띄어쓰기가 없는 문장)

Komoran	MeCab	Khaiii
굉장히/MAG	굉장히/MAG	굉장히잘/MAG
잘/MAG	잘/MAG	만들/VV
만들/VV	만든/VV+ETM	ㄴ/ETM
ㄴ/ETM	수작/NNG	수/MAG
수작/NNG	지루/XR	작지/NNG
지루/XR	할/XSA+ETM	루/JKB
하/XSA	틈/NNG	할/NNG
ㄹ/ETM	이/JKS	틈이/MAG
틈/NNG	없/VA	없/VA
이/JKS	음/ETN	음/ETN
없/VA		
음/ETN		

Komoran과 MeCab은 띄어쓰기가 없는 문장에 대해서도 좋은 성능을 보였다. Khaiii 는 대부분의 경우에서는 잘 해냈지만, 특정 경우(수작 지루할 틈이 -> 수 작지 루 할 틈이)에 대해서는 제대로 분류해내지 못하였다.

3. 아버지가방에들어가신다 (띄어쓰기가 없는 문장)

Komoran	MeCab	Khaiii
아버지/NNG	아버지/NNG	아버지/NNG
가방/NNP	가/JKS	가/JKS
에/JKB	방/NNG	방/NNG
들어가/VV	에/JKB	에/JKB
시/EP	들어가/VV	들어가/VV
ㄴ다/EC	신다/EP+EC	시/EP
		ㄴ다/EC

해당 경우에 대해서는 MeCab과 Khaiii는 좋은 성능을 보였지만, Komoran은 그렇지 못하였다.(아버지가 방에 ->아버지 가방에)

4. 아버지가 방에 들어가신다 (띄어쓰기가 없는 문장 3번과 비교)

Komoran	MeCab	Khaiii
아버지/NNG	아버지/NNG	아버지/NNG
가/JKS	가/JKS	가/JKS
방/NNG	방/NNG	방/NNG
에/JKB	에/JKB	에/JKB
들어가/VV	들어가/VV	들어가/VV
시/EP	신다/EP+EC	시/EP
ㄴ다/EC		ㄴ다/EC

Komoran, MeCab, Khaiii 모두 띄어씌가 되어 있는 경우에 대해서는 훌륭하게 분석해 내는 것을 볼 수 있다.

5. ㄱㅐOOO같은영화 뭐가무섭다는건지ㅡㅡ (자소 분리 및 오탈자가 포함된 문장)

Komoran	MeCab	Khaiii
개/NNB	ㄱ/NNG	ㄱ/NNG
OOO/NNP	ㅐ/UNKNOWN	ㅐO/NNG
같/VA	OOO/SL	OO/SL
은/ETM	같/VA	같/VA
영화/NNP	은/ETM	은/ETM
뭐가무섭다는건지ㅡ	영화/NNG	영화/NNG
	뭐/NP	뭐/NP
	가/JKS	가/JKS
	무섭/VA	무섭/VA
	다는/ETM	다/ETM
	건지/NNB+VCP+EC	는/EF
	ㅡㅡ/UNKNOWN	것/NNB
		이/VCP
		ㄴ지ㅡ/EC
		ㅡ/JKB

신기하게도 Komoran에서는 자소 분리가 발생하여도 올바르게 고쳐 형태소 분석을 진행하였다. 하지만, 이모티콘으로 사용된 'ㅡㅡ'는 분류하지 못하고 선행되는 형태소와 붙어 분석되었다. 'OOO'과 'ㅡㅡ'에 관해서는 MeCab에서 가장 우수한 성능을 발휘하였다. Khaiii는 자소 분리 및 오탈자 부분에 대해서는 처리하지 못하는 모습을 보여준다.

6. ㄴㅓ무합니다이무슨..유치찬란..오글거려못보겠네요 (자소 분리 및 오탈자가 포함된 문장)

Komoran	MeCab	Khaiii
너무/MAG	ㄴ/NNG	ㄴㅓ무합/NNG
하/XSV	ㅓ무합니다이무슨/UI	니다/EC
ㅂ니다/EC	./SF	이/NNP
이/NNB	./SY	무슨/MM
무스/NNP	유치찬란/XR	../SE
ㄴ/JX	./SF	유치/NNG
../SE	./SY	찬/NNP
유치/NNP	오/NR	란/NNG
찬란/XR	글/NNG	./SF
../SE	거려/XSV+EC	./SE
오/NNP	못/MAG	오글거리/VV
글/NNG	보/VV	어/EC
걸/VV	겠/EP	못/MAG
려/EC	네요/EC	보/VV
못/MAG		겠/EP
보/VV		네요/EC
겠/EP		
네요/EC		

마찬가지로 Komoran에서는 자소 분리된 경우도 수정하여 인식하였고, '..'과 같은 말 줄임표까지도 인식하는 결과를 보여준다. 이 외에 MeCab이나 Khaiii에서는 '..'를 제대로 인식하지 못하는 결과를 보여준다. Khaiii에서는 한번은 '..'을 SE로, 한번은 SF + SE로 인식하였다.

7. 개봉했을때부터 지금까지 마음이답답하거나 힘들때 이영화 보고있어요 그때마다 심적인 위로를 받을수있는영화같아요 장면 하나하나가 너무예쁘고 마음에 남아서 진한 여운까지 주는영화 감사합니다 (긴 문장)

Komoran	MeCab	Khaiii
개봉/NNG	개봉/NNG	개봉/NNG
하/XSV	했/XSV+EP	하/XSV
았/EP	을/ETM	였/EP
을/ETM	때/NNG	을/ETM
때/NNG	부터/JX	때/NNG
부터/JX	지금/NNG	부터/JX
지금/NNG	까지/JX	지금/NNG
까지/JX	마음/NNG	까지/JX
마음/NNG	이/JKS	마음/NNG
이/JKS	답답/XR	이/JKS
답답/XR	하/XSA	답답/NNG
하/XSA	거나/EC	하/XSA
거나/EC	힘들/VA+ETM	거나/EC
힘/NNG	때/NNG	힘들/VA
들/XSN	이/MM	ㄹ/ETM
때/NNG	영화/NNG	때/NNG
이영화/NNP	보/VV	이영/NNP
보/VV	고/EC	화/NNG
고/EC	있/VX	보/VV
있/VX	어요/EF	고/EC
어요/EC	그때/NNG	있/VX
그때/NNG	마다/JX	어요/EC
마다/JX	심/NNG	그때/NNG
심/NNG	적/XSN	마다/JX
적/XSN	인/VCP+ETM	심/NNG
이/VCP	위로/NNG	적/XSN
ㄴ/ETM	를/JKO	이/VCP
위로/NNG	받/VV	ㄴ/ETM
를/JKO	을/ETM	위로/NNG
받/VV	수/NNB	를/JKO
을/ETM	있/VV	받/VV
수/NNB	는/ETM	을/ETM
있/VX	영화/NNG	수/NNB
는/ETM	같/VA	있/VV
영화/NNP	아요/EF	는/ETM
같/VA	장면/NNG	영화/NNG
아요/EC	하나하나/NNG	같/VA
장면/NNG	가/JKS	아요/EC
하나하나/NNG	너무/MAG	장면/NNG
가/JKS	예쁘/VA	하나하나/NNG
너무/MAG	고/EC	가/JKS
예쁘/VA	마음/NNG	너무/MAG
고/EC	에/JKB	예쁘/VA
마음/NNG	남/VV	고/EC
에/JKB	아서/EC	마음/NNG
남/VV	진한/VA+ETM	에/JKB
아서/EC	여운/NNG	남/VV
진한/NNP	까지/JX	아서/EC
여운/NNP	주/VX	진하/VA
까지/JX	는/ETM	ㄴ/ETM
주/VX	영화/NNG	여운/NNG
는/ETM	감사/NNG	까지/JX
영화/NNP	합니다/XSV+EC	주/VV
감사/NNG		는/ETM
하/XSV		영화/NNG
ㅂ니다/EC		감사/NNG
		하/XSV
		ㅂ니다/EC

긴 문장이 들어와도 모두 제대로 처리하는 모습을 보여준다.

8. 나는 밥을 먹는다 (분석 시 주변 반영 확인, 9번과 연계)

Komoran	MeCab	Khaiii
나/NP	나/NP	나/NP
는/JX	는/JX	는/JX
밥/NNG	밥/NNG	밥/NNG
을/JKO	을/JKO	을/JKO
먹/VV	먹/VV	먹/VV
는다/EC	는다/EC	는다/EC

9. 하늘을 나는 자동차 (분석 시 주변 반영 확인, 8번과 연계)

Komoran	MeCab	Khaiii
하늘/NNG	하늘/NNG	하늘/NNG
을/JKO	을/JKO	을/JKO
나/NP	나/NP	나/NP
는/JX	는/JX	는/JX
자동차/NNG	자동차/NNG	자동차/NNG

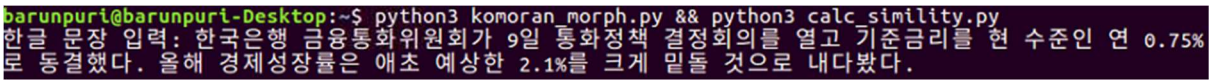
8번과 같이 볼 때, Komoran, MeCab, Khaiii모두 '나는'을 NP + JX로 인식하여 대명사 + 보조사 로 인식한 것을 볼 수 있다. 3가지 모두 주변을 고려한다고 보는 것은 어려워 보인다.

10. 아이폰 기다리다 지쳐 애플공홈에서 언락폰질러버렸다 6+ 128기가실버ㅋ
(사전에 포함되지 않은 단어 분석)

Komoran	MeCab	Khaiii
아이폰/NNP	아이폰/NNP	아이폰/NNG
기다리/VV	기다리/VV	기다리/VV
다/EC	다/EC	다/EC
지치/VV	지쳐/VV+EC	지치/VV
어/EC	애플/NNP	어/EC
애플/NNP	공홈/NNG	애플공/NNP
공/NNP	에서/JKB	홈/NNG
홈/NNG	언락/NNG	에서/JKB
에서/JKB	폰/NNG	언락폰/NNG
언/NNG	질러/VV+EC	질/VV
락/NNG	버렸/VX+EP	어/EC
폰/NNG	다/EC	버리/VX
지르/VV	6/SN	었/EP
어/EC	+/SY	다/EC
버리/VX	128/SN	6/SN
었/EP	기/NNG	+/SW
다/EC	가/JKS	128/SN
6/SN	실버/NNP	기/NNB
+/SW	ㅋ/IC	가실버/NNG
128기가실버ㅋ/NA		ㅋ/VA

3. 수행 시간 측정

형태소 분석하는 부분과 word embedding을 하나의 part로 묶고, 각 형태소의 가중치(tf-idf)계산과 유사도(cosine- similarity)계산 부분을 하나의 part로 묶어 진행하였다. 따라서 첫번째 part에서는 각 형태소 모듈에 맞게 형태소 분석이 진행되고 다른 part에서는 유사도 계산을 하나의 파일에서 진행한다. 형태소 분석이 끝나면 morph.txt, morph_analysis_result.txt파일에 데이터를 저장하였다가 두번째 part인 calc_simility.py에서 불러와 유사도 계산을 수행한다. 이후 최종적으로 10개의 가장 유사한 문장을 출력하며, 수행시간은 유사도 계산 시간을 제외한, 형태소 분석, word embedding으로 걸린 시간을 출력한다.



<실행 방법>

$ python3 [komoran | khaiii | mecab]_morph.py && python3 calc_similarity.py

<소스코드 – [komoran | khaiii | mecab]_morph.py >

- Komoran

```
from konlpy.tag import Komoran
komo = Komoran()

import time

print( '한글 문장 입력: ')
input_origin = str(input())

# 출력할 문장 수
n = 10

#timer start
start = time.time()

f = open('KCC150_K01_utf8.txt') #

word = []       #형태소 list
sentence_morph_list = []        #각 문장을 형태소 분석한 결과, 출현 횟수
sentence_cnt_of_morph_appear = []       #각 형태소가 출현한 문서 수

# % 출력
total_lines = 1000000
line_cnt = 0
percent = 0

#입력 문장
input_morph = dict()
for m in komo.morphs(input_origin):
    if(m not in word):
        word.append(m)
        sentence_cnt_of_morph_appear.append(0)
    morph_index = word.index(m)
    input_morph[morph_index] = input_morph.get(morph_index,0) + 1

    for key,val in input_morph.items():
        sentence_cnt_of_morph_appear[key] += 1
```

필요한 module을 import 하고 비교할 한글 문장 입력, timer를 시작한다.
자료구조를 선언, 입력된 한글 문장에 대해서 형태소 분석을 하고 각 형태소를 word배열에 넣고 해당하는 index로 이후 배열들을 구성한다. input_morph는 <index, 빈도>의 값을 가진다. sentence_morph_list는 파일에서 입력된 각 입력의 형태소 분석 결과를 저장하는 배열이다. sentence_cnt_of_morph_appear는 idf 계산을 위해 각 형태소마다 출현 문장의 수를 저장한다.

```
#word embedding, 형태소 분석
while(True):
    line_cnt += 1
    if( int(100 * line_cnt / total_lines) != percent):
        percent += 1
        print("current₩t{}% take₩t{}s".format(percent, time.time()-start ))

    line = f.readline()
    if not line:
        f.close()
        break

    sentence_morph = dict()
    morph = komo.morphs(line)
    for m in morph:
        if( m not in word ):
            word.append(m)
            sentence_cnt_of_morph_appear.append(0)

        morph_index = word.index(m)
        sentence_morph[morph_index] = sentence_morph.get(morph_index,0) + 1

    for key,val in sentence_morph.items():
        sentence_cnt_of_morph_appear[key] += 1

    sentence_morph_list.append(sentence_morph)

time_taken = time.time() - start
print('morphological analysis time: {}s'.format(time_taken))

f.close()
```

```
current 39% take        2676.541920900345s
current 40% take        2748.9233405590057s
current 41% take        2822.3778734207153s
current 42% take        2895.1982839107513s
current 43% take        2967.7682740688324s
current 44% take        3041.023577928543s
current 45% take        3114.3724150657654s
current 46% take        3187.7838966846466s
```

진행상황 확인을 위해 1%가 진행될 때마다 걸린 시간을 출력하였다.
파일에서 입력된 문장 역시 유사도 계산을 위해 입력된 문장과 마찬가지로 형태소 분석을 진행하고 tf-idf계산을 위한 값들을 저장한다. Sentence_morph는 파일에서 입력된 문장의 형태소 분석 결과를 <word_index, 빈도수>로 저장하고 이 sentence_morph는 다시 sentence_morph_list에 저장된다.
형태소 분석이 끝난 후 형태소 분석에 걸린 전체 시간을 출력한다.

```
current 96% take        6916.4895877838135s
current 97% take        6991.065686225891s
current 98% take        7065.826820850372s
current 99% take        7140.1314532756805s
current 100% take       7215.359827756882s
morphological analysis time: 7215.364519834518s
```

```
f.close()

f = open('morph.txt', 'w')
for m in word:
    f.write(m +'₩n')
f.close()

f = open('morph_analysis_result.txt', 'w')
f.write(str(input_origin) +'₩n')
f.write(str(input_morph) +'₩n')
f.write(str(n) +'₩n')
f.write(str(len(word)) +'₩n')
for n in sentence_cnt_of_morph_appear:
    f.write(str(n) +'₩n')
f.write(str(total_lines) +'₩n')
for sentence in sentence_morph_list:
    f.write(str(sentence) +'₩n')
f.close()
```

이후 유사도 계산을 위해 필요한 값들을 파일에 저장한다. 'morph.txt'파일은 모든 형태소 목록을 가진다. 'morph_analysis_result.txt'파일은 유사도 계산을 위해 필요한 값들이 저장된다. 우선 유사도 계산을 위한 입력문장(input_origin), 그 문장의 형태소 분석 결과(input_morph), 출력 문장의 수(n), 전체 형태소의 수(len(word)), idf계산을 위한 각 형태소의 출현 문장 횟수(sentence_cnt_of_morph_appear), 전체 비교 문장의 수(total_lines), 각 문장의 형태소 분석 결과(sentence_morph_list)를 저장한다.

khaiii_morph.py 와 mecab_morph.py 는 komoran_morph.py와 달라지는 부분만 비교한다.
왼쪽이 komoran이고 오른쪽이 khaiii 또는 mecab이다.

- Khaiii

```
from konlpy.tag import Komoran
komo = Komoran()
```

```
from khaiii import KhaiiiApi
api = KhaiiiApi()
```

Khaiii모듈을 import한다.

```
#입력 문장
input_morph = dict()
for m in komo.morphs(input_origin):
    if(m not in word):
        word.append(m)
        sentence_cnt_of_morph_appear.append(0)
    morph_index = word.index(m)
    input_morph[morph_index] = input_morph.get(morph_index,0) + 1

    for key,val in input_morph.items():
        sentence_cnt_of_morph_appear[key] += 1
```

```
#입력 문장
input_morph = dict()
for morph_list in api.analyze(input_origin):
    print( morph_list ) ##
    for m in morph_list.morphs:
        if(m.lex not in word):
            word.append(m.lex)
            sentence_cnt_of_morph_appear.append(0)
        morph_index = word.index(m.lex)
        input_morph[morph_index] = input_morph.get(morph_index,0) + 1

    for key,val in input_morph.items():
        sentence_cnt_of_morph_appear[key] += 1
```

Komoran은 konlpy의 모듈이기 때문에 morph함수를 사용하여 형태소만 추출이 가능한데 Khaiii는 추가적인 작업이 필요하여 추가하였다.

```
sentence_morph = dict()
morph = komo.morphs(line)
for m in morph:
    if( m not in word ):
        word.append(m)
        sentence_cnt_of_morph_appear.append(0)

    morph_index = word.index(m)
    sentence_morph[morph_index] = sentence_morph.get(morph_index,0) + 1
```

```
sentence_morph = dict()
for morph_list in api.analyze(line):
    for m in morph_list.morphs:
        if( m.lex not in word ):
            word.append(m.lex)
            sentence_cnt_of_morph_appear.append(0)

        morph_index = word.index(m.lex)
        sentence_morph[morph_index] = sentence_morph.get(morph_index,0) + 1
```

동일하게 입력 파일에 대해서도 형태소 추출과정에서 추가적인 작업만 변경해 주었다.

- MeCab

```
from konlpy.tag import Komoran
komo = Komoran()
```

```
import MeCab
mecab = MeCab.Tagger('-d ./mecab-ko-dic-2.1.1-20180720/')
```

Mecab은 dic파일의 위치를 불러와 줘야 한다.

```
#입력 문장
input_morph = dict()
for m in komo.morphs(input_origin):
    if(m not in word):
        word.append(m)
        sentence_cnt_of_morph_appear.append(0)
    morph_index = word.index(m)
    input_morph[morph_index] = input_morph.get(morph_index,0) + 1

    for key,val in input_morph.items():
        sentence_cnt_of_morph_appear[key] += 1
```

```
#입력 문장
input_morph = dict()
morph_list = mecab.parse(input_origin)
morph_list = morph_list[:-5].split('₩n')
morph_list = [morph.split('₩t')[0] morph in morph_list]

for m in morph_list:
    if(m not in word):
        word.append(m)
        sentence_cnt_of_morph_appear.append(0)
    morph_index = word.index(m)
    input_morph[morph_index] = input_morph.get(morph_index,0) + 1

    for key,val in input_morph.items():
        sentence_cnt_of_morph_appear[key] += 1
```

Khaiii와 동일하게 입력 문장에 대해서 형태소 분석 결과에서 형태소만 추출하는 작업만 추가하였다.

```
sentence_morph = dict()
morph = komo.morphs(line)
for m in morph:
    if( m not in word ):
        word.append(m)
        sentence_cnt_of_morph_appear.append(0)

    morph_index = word.index(m)
    sentence_morph[morph_index] = sentence_morph.get(morph_index,0) + 1
```

```
sentence_morph = dict()
morph_list = mecab.parse(line)
morph_list = morph_list[:-5].split('₩n')
morph_list = [morph.split('₩t')[0] for morph in morph_list]

for m in morph_list:

    if( m not in word ):
        word.append(m)
        sentence_cnt_of_morph_appear.append(0)

    morph_index = word.index(m)
    sentence_morph[morph_index] = sentence_morph.get(morph_index,0) + 1
```

동일하게 입력 파일에 대해서도 형태소 추출과정에서 추가적인 작업만 변경해 주었다.

<소스코드 – calc_similarity.py >

```
import math
from ast import literal_eval

f = open('morph_analysis_result.txt')

origin_sentence = f.readline()
input_sentence = literal_eval(f.readline())
n_print = int(f.readline())

total_morph = int(f.readline())
sentence_cnt_of_morph_appear = []
for i in range(total_morph):
        sentence_cnt_of_morph_appear.append(int(f.readline()))

total_lines = int(f.readline())
```

필요한 module을 불러오고 유사도 계산에 필요한 데이터를 파일로부터 불러온다. 유사도 계산을 위한 입력문장(origin_sentence), 그 문장의 형태소 분석 결과(input_sentence), 출력 문장의 수(n_print), 전체 형태소의 수(total_morph), idf계산을 위한 각 형태소의 출현 문장 횟수(sentence_cnt_of_morph_appear), 전체 비교 문장의 수(total_lines)를 불러온다. 각 문장의 형태소 분석 결과(sentence_morph_list)를 저장한다.

```
input_tfidf = dict()
for key, val in input_sentence.items():
        tf = val / sum(list(input_sentence.values()))
        idf = math.log10(total_lines / sentence_cnt_of_morph_appear[key])

        input_tfidf[key] = tf*idf
```

이후 입력 문장에 대해서 tf-idf가중치를 계산한다. tf는 term-frequency로 각 단어(형태소)가 문장 내에서 나타난 빈도를 의미하고, idf는 inverse-document-frequency로 각 단어(형태소)가 전체 문장에서 나타난 document-frequency의 역수를 갖는다. 따라서 각 문장내에서 중요단어를 찾고, 전체 문장에서 빈도가 낮은 단어에는 가중치를 주게 된다.

```
for i in range(total_lines):
        line = f.readline()
        sentence = literal_eval(line)

        #tfidf
        tfidf = dict()
        for key, val in sentence.items():
                tf = val / sum(list(sentence.values())) #문장에서 단어 빈도
                idf = math.log10( total_lines / sentence_cnt_of_morph_appear[key] ) #전
                tfidf[key] = tf*idf

        #유사도
        cosine_sim = 0
        for key, val in tfidf.items():
                cosine_sim += min(val, input_tfidf.get(key,0))

        cosine_sim /= math.sqrt( sum([val*val for key, val in tfidf.items()])
                                * sum([val*val for key,val in input_tfidf.items()]) )

        similarity.append((i, cosine_sim))

similarity.sort(reverse = True, key = lambda s:s[1])

f.close()
```

각 문장에 해당하는 형태소 분석 결과를 가지고 각 문장에 있는 형태소의 tf-idf를 계산하고 계산된 tf-idf와 입력 문장의 tf-idf (input_tfidf)와 유사도를 계산한다. 유사도는 코사인 유사도 계산 방식을 사용하여 계산하였다. 이후, <문장의 index, 유사도>를 similarity배열에 저장하고 정렬한다.

```
f = open('KCC150_K01_utf8.txt')
sentence_list = []
for i in range(total_lines):
        sentence_list.append(f.readline())
f.close()

for i in range(n_print):
        print("{}₩t{}".format(i+1, sentence_list[similarity[i][0]] ) )
```

원문을 출력하기 위해 파일을 읽어와 유사도가 높은 n개에 대해서 출력한다.

<실행결과>

- Komoran

```
morphological analysis time: 7215.3645198345185
1       한편 한국은행 금융통화위원회가 이날 6년 5개월 만에 기준금리를 인상한 데 대해 정부는 최근의 경기 회복세를 감안한 결정으로 향후 경기에 큰 영향을 미치지는 않을 것으로 판단했다.

2       한편, 한국은행은 5일 금융통화위원회를 열어 6월 콜금리를 결정할 예정이지만, 최근의 환율 하락으로 물가상승 요인이 해소됐기 때문에 콜금리를 동결할 것이란 전망이 우세한 편이다.

3       하지만 지난 11일 한국은행 금융통화위원회가 시장의 예상을 깨고 기준금리를 동결한 충격으로 그동안 기준금리 인하를 선반영했던 국채 금리가 일제히 뛰어오르면서 상황이 뒤바뀌었다.

4       4일 열린 금융통화위원회가 콜금리 동결방침을 7개월째 고수한 것은, 아직까지는 통화신용정책의 중심추가 경기부양 쪽으로 좀더 기울어 있음을 내보인 것으로 받아들여진다.

5       일 세 나라가 오는 9일 일본 도쿄에서 대북정책조정감독그룹 회의를 열고 북한과 관련한 전반적인 문제들을 논의할 것이라고 발표했다.

6       씨티은행과 뱅크오브아메리카는 경제성장률 상승으로 통화정책 정상화 압력이 높아지고 있어 금리 인상이 내년 1분기와 하반기 중 두 차례 단행될 것이라고 전망했다.

7       금리동결을 고집스럽게 지켜오던 금통위가 금리인하를 결정한 의미를 여러 가지 국내외 경제상황과 결부시켜 설명할 수 있지만, 한마디로 요약하자면 통화 공급을 늘려 경기를 부양한다는 세계적 흐름에 막차를 탔다는 것이다.

8       전문가들은 한국은행이 지난달 기준금리를 내렸기 때문에 이번 달엔 그 효과를 지켜보기 위해 금리를 동결한 것으로 보고 있습니다.

9       기준금리는 한국은행 금융통화위원회 회의를 통해 결정하는 정책금리이지만, 가산금리는 개별 은행 사정에 따라 위험성과 은행 비용 등을 통합해 자율적으로 결정한다.

10      3월 셋째 주는 지난주 마지막 거래일 증시를 반등시킨 원동력인 한국은행의 깜짝 기준금리 인하에 대한 긍정적인 기대가 전반적으로 확산되며 지수를 견일 할 것으로 예상된다.
```

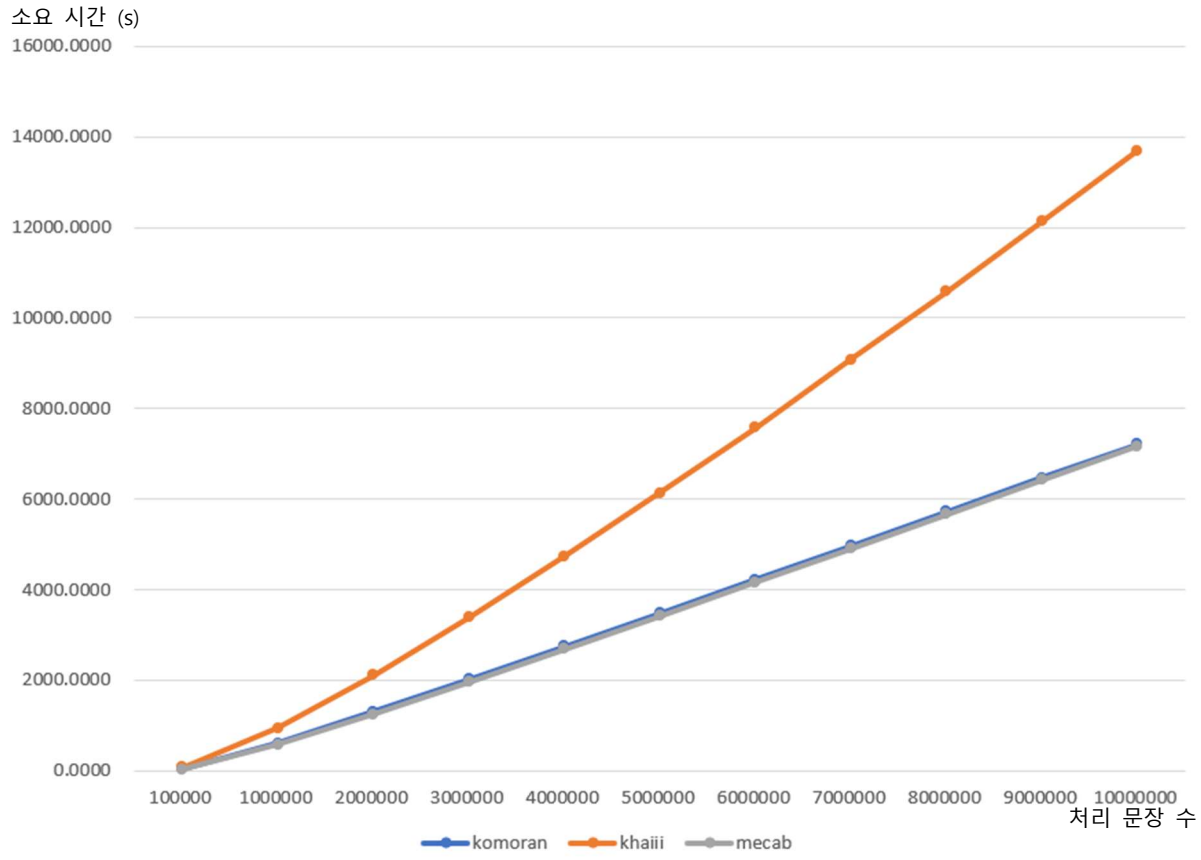
- Khaiii

```
morphological analysis time: 13696.917665243149s
1       한편 한국은행 금융통화위원회가 이날 6년 5개월 만에 기준금리를 인상한 데 대해 정부는 최근의 경기 회복세를 감안한 결정으로 향후 경기에 큰 영향을 미치지는 않을 것으로 판단했다.

2       이번 대회는 지역 내 5개 대학교에서 모두 9개팀이 참가해 국내 및 해외 경제현황을 분석하고 한국은행 기준금리 관련 통화정책 방향을 발표한 뒤 심사위원 질의에 답변하는 방식으로 지난 27일 진행됐다.

3       한편, 한국은행은 5일 금융통화위원회를 열어 6월 콜금리를 결정할 예정이지만, 최근의 환율 하락으로 물가상승 요인이 해소됐기 때문에 콜금리를 동결할 것이란 전망이 우세한 편이다.

4       지난 5월 9일, 세월호 사고 이후 처음으로 기준금리를 결정하는 한국은행 금융통화위원회가 열렸다.

5       하지만 지난 11일 한국은행 금융통화위원회가 시장의 예상을 깨고 기준금리를 동결한 충격으로 그동안 기준금리 인하를 선반영했던 국채 금리가 일제히 뛰어오르면서 상황이 뒤바뀌었다.

6       김중수 한국은행 총재가 지난 2월 기준금리 동결을 결정한 금융통화위원회 전체회의에서 생각에 잠겨 있다.

7       금융개혁 차원에서나 한국은행의 입장에서도 공히 인식해야 할 점은 통화정책 운영체계를 어떻게 하느냐 하는 문제보다 경제운영에 대한 중앙은행의 지도력과 통화정책에 대한 국민들의 신뢰를 확보하는 것이 더욱 중요한 과제라는 점이다.

8       기준금리는 한국은행 금융통화위원회 회의를 통해 결정하는 정책금리이지만, 가산금리는 개별 은행 사정에 따라 위험성과 은행 비용 등을 통합해 자율적으로 결정한다.

9       금리동결을 고집스럽게 지켜오던 금통위가 금리인하를 결정한 의미를 여러 가지 국내외 경제상황과 결부시켜 설명할 수 있지만, 한마디로 요약하자면 통화 공급을 늘려 경기를 부양한다는 세계적 흐름에 막차를 탔다는 것이다.

10      지난 1분기 이상한파로 경제 활동 전반이 위축됐고 의료보험 개혁안의 본격적 시행에도 의료비 지출은 예상을 밑돌면서 경제성장률은 3년 만에 역성장한 바 있다.
```

- MeCab

```
morphological analysis time: 7172.340677976608s
1       한편, 한국은행은 5일 금융통화위원회를 열어 6월 콜금리를 결정할 예정이지만, 최근의 환율 하락으로 물가상승 요인이 해소됐기 때문에 콜금리를 동결할 것이란 전망이 우세한 편이다.

2       한편 한국은행 금융통화위원회가 이날 6년 5개월 만에 기준금리를 인상한 데 대해 정부는 최근의 경기 회복세를 감안한 결정으로 향후 경기에 큰 영향을 미치지는 않을 것으로 판단했다.

3       이번 대회는 지역 내 5개 대학교에서 모두 9개팀이 참가해 국내 및 해외 경제현황을 분석하고 한국은행 기준금리 관련 통화정책 방향을 발표한 뒤 심사위원 질의에 답변하는 방식으로 지난 27일 진행됐다.

4       하지만 지난 11일 한국은행 금융통화위원회가 시장의 예상을 깨고 기준금리를 동결한 충격으로 그동안 기준금리 인하를 선반영했던 국채 금리가 일제히 뛰어오르면서 상황이 뒤바뀌었다.

5       지난 5월 9일, 세월호 사고 이후 처음으로 기준금리를 결정하는 한국은행 금융통화위원회가 열렸다.

6       기준금리는 한국은행 금융통화위원회 회의를 통해 결정하는 정책금리이지만, 가산금리는 개별 은행 사정에 따라 위험성과 은행 비용 등을 통합해 자율적으로 결정한다.

7       김중수 한국은행 총재가 지난 2월 기준금리 동결을 결정한 금융통화위원회 전체회의에서 생각에 잠겨 있다.

8       4일 열린 금융통화위원회가 콜금리 동결방침을 7개월째 고수한 것은, 아직까지는 통화신용정책의 중심추가 경기부양 쪽으로 좀더 기울어 있음을 내보인 것으로 받아들여진다.

9       한국은행도 19일 금융통화위원회에서 기준금리 인상을 주장하는 소수의견이 등장하면서 시장은 연내 금리 인상 가능성을 크게 보며 들썩이고 있다.

10      참고로 한은은 1년에 8번의 통화정책 방향 금통위를 열고 기준금리를 결정하고 있다.
```

<분석시간 그래프>



4. 결론

- 데이터의 특성(띄어쓰기 유무, 오탈자, 사전에 등록되지 않은 단어)이나 개발환경(Linux, Window, Mac, Python, Java)에 따라서 적합한 형태소 분석기를 고려해야 한다.
- 연산 속도가 중요하면 MeCab이나 Komoran이 가장 우수할 것으로 보인다.
- 모바일 환경의 사용량이 많아지고 있는 지금, 자주 발생할 수 있는 자소 분리나 오탈자에 대해서도 어느 정도 분석 품질이 보장되어야 한다면 Komoran을 사용하는 것이 좋아 보인다.
- Khaiii는 아직 일부 케이스에 대한 분석 품질이 좋지 않고, 분석 시간에서 아쉬운 점이 보인다.