

<빅데이터 최신기술 4차과제>

문장 유사도 계산을 3 가지 형태소 분석기에 의한 구현

20153167 김현중

구현 코드 1 (KoLPY의 Hannanum class 이용)

```
import time
start = time.time()
from konlpy.tag import Hannanum
doc = input("문장을 입력해주세요 : ") ## 문장을 입력받는다
n = input("입력문장과 유사한 몇개의 문장을 출력할까요? : ") ## 유사한 문장을 몇개 출력할지 입력 받는다
print(" ")
hannanum = Hannanum() ##형태소 분석을 위하여 konlpy의 Hannanum class를 이용하여 준다
doc_tokenized = hannanum.morphs(doc) ##입력문장을 형태소 단위로 쪼개준다
doc_tokenized_size = len(doc_tokenized) ##입력문장의 형태소 개수

list=[]

with open('KCCq281000.txt', 'r', encoding='utf-8') as input: # 유사도 검사할 말뭉치를 가져옴
    for line in input: ##말뭉치에서 라인 단위로 읽어준다
        file_tokenized = hannanum.morphs(line) ##읽어들인 라인을 형태소 단위로 쪼개준다
        file_tokenized_size = len(file_tokenized) ##읽어들인 라인의 형태소 개수
        intersection_size = 0 ##교집합 형태소 개수를 초기화
        for x in doc_tokenized:
            if x in file_tokenized:
                intersection_size+=1 ##입력문장과 라인문장을 비교하여 겹치는 형태소가 있다면 교집합 형태소 개수 증가시키기
        if len(doc) <= len(line):
            short = doc_tokenized_size
        else:
            short = file_tokenized_size ## 입력문장과 파일 각 문장의(라인단위) 길이를 비교하여 짧은 문장의 형태소 개수를 short에 저장
        similarity = float(intersection_size) / float(short) ## 공통 형태소 개수 / 짧은 문장소 형태소 개수 = 유사도
        list.append([line, similarity * 100]) ##list 배열에 문장과, 그문장의 입력문장에 대한 유사도를 집어넣어준다

sorted_list = sorted(list, key=lambda x: -x[1]) ##list 배열을 유사도 순으로 정렬하여 준다(유사도가 높은순으로, 내림차순)
for i in range(int(n)):
    print(sorted_list[i][0]) ##입력받은 n개의 유사도가 높은 문장을 출력한다.
    print("유사도는 " + str(sorted_list[i][1]) + "% 입니다.") ##입력받은 n개의 유사도가 높은 문장의 유사도를 출력한다.
    print(" ")
    print(" ")

print("소요시간 :", time.time() - start, "초") # 소요시간 출력
```

구현 코드 2 (KoLPY의 Komoran class 이용)

```
import time
start = time.time()
from konlpy.tag import Komoran
doc = input("문장을 입력해주세요 : ") ## 문장을 입력받는다
n = input("입력문장과 유사한 몇개의 문장을 출력할까요? : ") ## 유사한 문장을 몇개 출력할지 입력 받는다
print(" ")
komoran = Komoran() ##형태소 분석을 위하여 konlpy의 Komoran class를 이용하여 준다
doc_tokenized = komoran.morphs(doc) ##입력문장을 형태소 단위로 쪼개준다
doc_tokenized_size = len(doc_tokenized) ##입력문장의 형태소 개수

list=[]

with open('KCCq281000.txt', 'r', encoding='utf-8') as input: # 유사도 검사할 말뭉치를 가져옴
    for line in input: ##말뭉치에서 라인 단위로 읽어준다
        file_tokenized = komoran.morphs(line) ##읽어들인 라인을 형태소 단위로 쪼개준다
        file_tokenized_size = len(file_tokenized) ##읽어들인 라인의 형태소 개수
        intersection_size = 0 ##교집합 형태소 개수를 초기화
        for x in doc_tokenized:
            if x in file_tokenized:
                intersection_size+=1 ##입력문장과 라인문장을 비교하여 겹치는 형태소가 있다면 교집합 형태소 개수 증가시키기
        if len(doc) <= len(line):
            short = doc_tokenized_size
        else:
            short = file_tokenized_size ## 입력문장과 파일 각 문장의(라인단위) 길이를 비교하여 짧은 문장의 형태소 개수를 short에 저장
        similarity = float(intersection_size) / float(short) ## 공통 형태소 개수 / 짧은 문장소 형태소 개수 = 유사도
        list.append([line, similarity * 100]) ##list 배열에 문장과, 그문장의 입력문장에 대한 유사도를 집어넣어준다

sorted_list = sorted(list, key=lambda x: -x[1]) ##list 배열을 유사도 순으로 정렬하여 준다(유사도가 높은순으로, 내림차순)
for i in range(int(n)):
    print(sorted_list[i][0]) ##입력받은 n개의 유사도가 높은 문장을 출력한다.
    print("유사도는 " + str(sorted_list[i][1]) + "% 입니다.") ##입력받은 n개의 유사도가 높은 문장의 유사도를 출력한다.
    print(" ")
    print(" ")

print("소요시간 :", time.time() - start, "초") # 소요시간 출력
```

구현 코드 3 (KoLPY의 Kkma class 이용)

```
import time
start = time.time()
from konlpy.tag import Kkma
doc = input("문장을 입력해주세요 : ") ## 문장을 입력받는다
n = input("입력문장과 유사한 몇개의 문장을 출력할까요? : ") ## 유사한 문장을 몇개 출력할지 입력 받는다
print(" ")
kkma = Kkma() ##형태소 분석을 위하여 konlpy의 Kkma class를 이용하여 준다
doc_tokenized = kkma.morphs(doc) ##입력문장을 형태소 단위로 쪼개준다
doc_tokenized_size = len(doc_tokenized) ##입력문장의 형태소 개수

list=[]

with open('KCCq281000.txt', 'r', encoding='utf-8') as input: # 유사도 검사할 말뭉치를 가져옴
    for line in input: ##말뭉치에서 라인 단위로 읽어준다
        file_tokenized = kkma.morphs(line) ##읽어들인 라인을 형태소 단위로 쪼개준다
        file_tokenized_size = len(file_tokenized) ##읽어들인 라인의 형태소 개수
        intersection_size = 0 ##교집합 형태소 개수를 초기화
        for x in doc_tokenized:
            if x in file_tokenized:
                intersection_size+=1 ##입력문장과 라인문장을 비교하여 겹치는 형태소가 있다면 교집합 형태소 개수 증가시키기
        if len(doc) <= len(line):
            short = doc_tokenized_size
        else:
            short = file_tokenized_size ## 입력문장과 파일 각 문장의(라인단위) 길이를 비교하여 짧은 문장의 형태소 개수를 short에 저장
        similarity = float(intersection_size) / float(short) ## 공통 형태소 개수 / 짧은 문장소 형태소 개수 = 유사도
        list.append([line, similarity * 100]) ##list 배열에 문장과, 그문장의 입력문장에 대한 유사도를 집어넣어준다

sorted_list = sorted(list, key=lambda x: -x[1]) ##list 배열을 유사도 순으로 정렬하여 준다(유사도가 높은순으로, 내림차순)
for i in range(int(n)):
    print(sorted_list[i][0]) ##입력받은 n개의 유사도가 높은 문장을 출력한다.
    print("유사도는 " + str(sorted_list[i][1]) + "% 입니다.") ##입력받은 n개의 유사도가 높은 문장의 유사도를 출력한다.
    print(" ")
    print(" ")

print("소요시간 :", time.time() - start, "초") # 소요시간 출력
```

유사도 계산 STEP

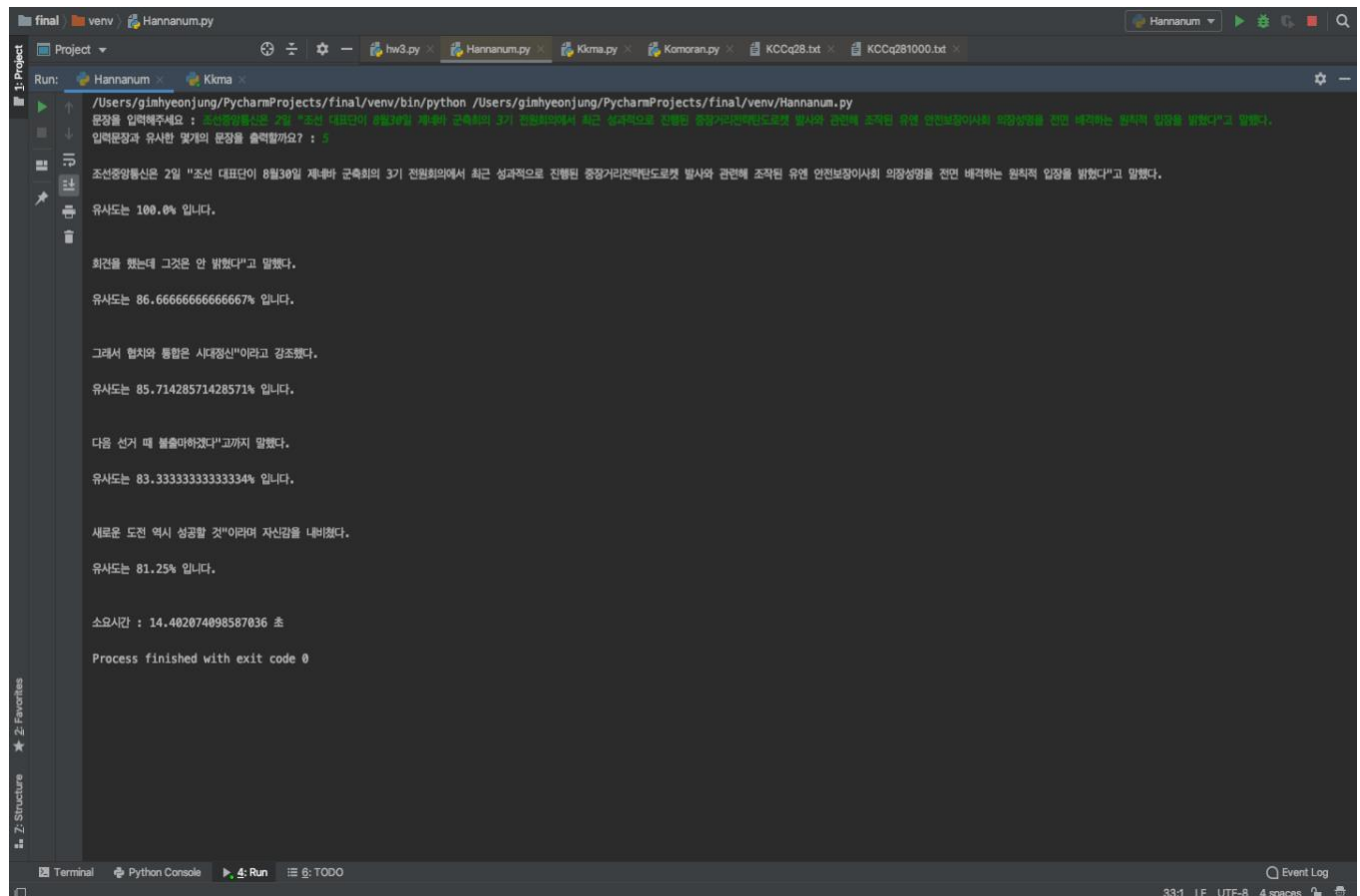
(Hannanum , Komoran, Kkma class의 import와
객체 생성만 다를뿐 전체적인 방법은 같습니다)

1. Konlpy의 class(1.Hannamun 2.Komoran 3.Kkma)를 import한다.
2. 문장을 입력받는다.
3. 유사한 문장을 몇개를 출력할지 입력받는다(n개).
4. 각 class로부터 객체(1.hannanum 2.komoran 3.kkma)를 생성해준다.
5. 각 객체의 morphs method를 이용하여 입력받은 문장을 형태소 단위로 쪼개어준다(tokenizing).
6. 형태소 단위로 쪼개진 입력문장에서 형태소의 개수를 세준다.
7. KCCq281000 파일을 입력으로 가져온다.
(KCCq281000 파일의 상위 1000라인을 추출하여 사용하였습니다)
8. 대용량 파일을 라인단위로 가져온다
9. 라인 단위로 가져온 문장을 각 객체의 morphs 메소드를 이용하여 형태소 단위로 쪼개어준다(tokenizing).
10. 형태소 단위로 쪼개진 문장의 형태소 개수를 세준다.
11. 교집합 형태소의 개수를 0으로 초기화 해준다.
12. 입력문장의 tokenized list와 라인의 tokenized list를 비교해주어 겹치는 형태소가 존재할 경우 카운트를 증가시켜준다.
13. 입력문장과 라인의 길이를 비교하여 짧은 문장의 형태소 개수를 short 변수에 넣어준다.
14. $\text{Intersection_size}(\text{교집합 형태소 개수}) / \text{short}(\text{짧은 문장 형태소 개수})$ 식을 이용하여 두 문장간의 유사도를 similarity 변수에 넣어준다.
15. 각 라인(문장)과 그 문장의 입력문장에 대한 유사도 (similarity*100)를 2차원 배열 list에 넣어준다.

16. 대용량 한국어 텍스트 파일을 모두 읽었을 때
list 배열에는 파일의 라인 순서대로 [문장, 유사도]가 저장이 된다.
17. list 배열을 similarity가 높은 순서대로 정렬한다
18. 출력할 문장과 유사도는 n개이므로 정렬된 배열에서 처음부터
For in range(int(n))
n개의 문장(sorted_list[i][0]),
n개 문장의 유사도(sorted_list[i][1])를 출력하면 된다.
19. 모든 작업을 마쳤을 시 소요시간을 출력한다.

출력 결과

Hannanum class 이용시



```
Run: Hannanum.py
/Users/ginhyeonjung/PycharmProjects/final/venv/bin/python /Users/ginhyeonjung/PycharmProjects/final/venv/Hannanum.py
문장을 입력해주세요 : 조선중앙통신은 2일 "조선 대표단이 8월30일 제-후바 군축회의 3기 전행회의에서 최근 성과적으로 진행된 중장거리전력탄도로켓 발사와 관련해 조차된 유엔 안전보장이사회 의장상명을 전면 배격하는 원칙적 입장을 밝혔다"고 말했다.
입력문장과 유사한 몇개의 문장을 출력할까요? : 5
조선중앙통신은 2일 "조선 대표단이 8월30일 제-후바 군축회의 3기 전행회의에서 최근 성과적으로 진행된 중장거리전력탄도로켓 발사와 관련해 조차된 유엔 안전보장이사회 의장상명을 전면 배격하는 원칙적 입장을 밝혔다"고 말했다.
유사도는 100.0% 입니다.

최근들 했는데 그것은 안 밝혔다"고 말했다.
유사도는 86.66666666666667% 입니다.

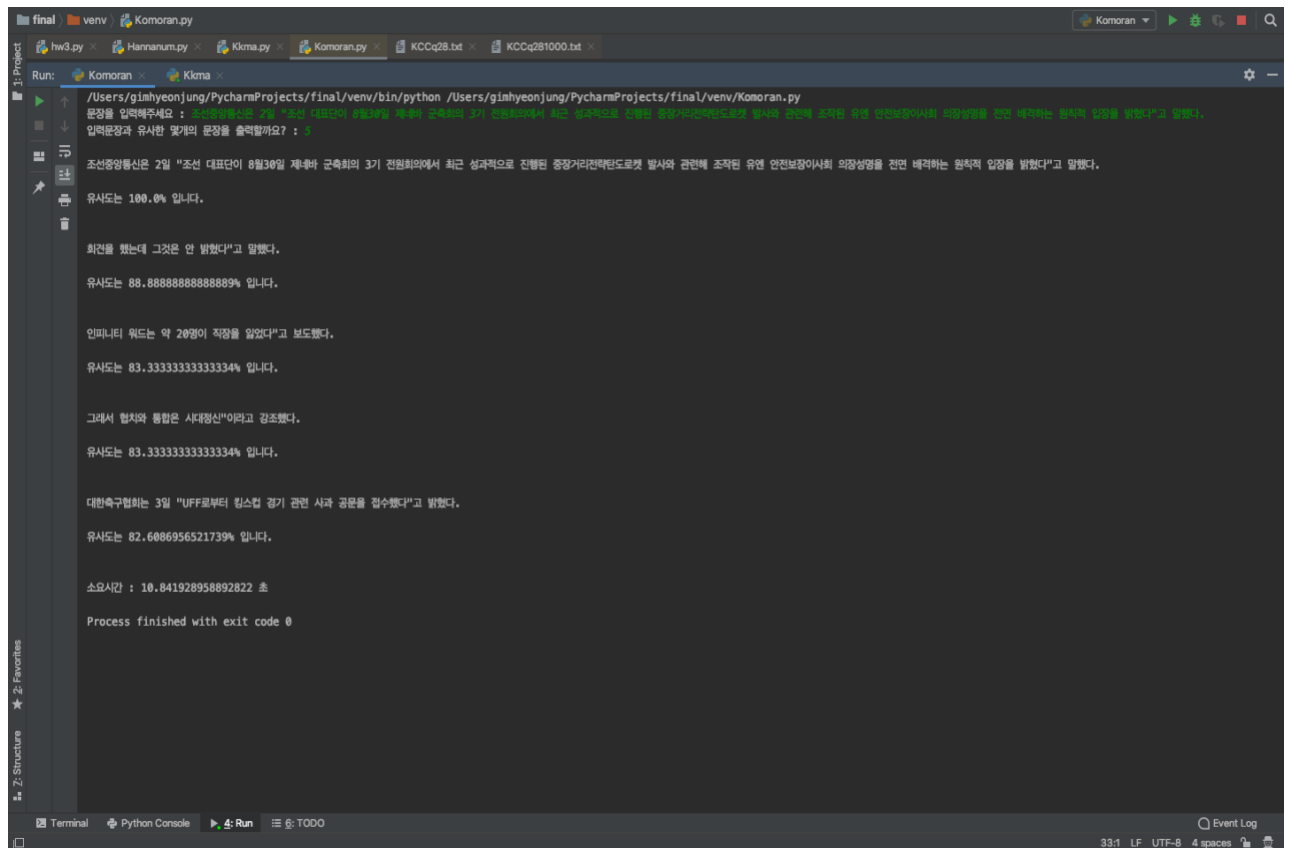
그래서 협치와 통합은 시대정신"이라고 강조했다.
유사도는 85.71428571428571% 입니다.

다음 선거 때 불출마하겠다"고까지 말했다.
유사도는 83.33333333333333% 입니다.

새로운 도전 역시 성공할 것"이라며 자신감을 내비쳤다.
유사도는 81.25% 입니다.

소요시간 : 14.402074098587036 초
Process finished with exit code 0
```

Komoran class 이용시



```
Run: Komoran x Kkma x
/Users/gimhyeonjung/PycharmProjects/final/venv/bin/python /Users/gimhyeonjung/PycharmProjects/final/venv/Komoran.py
문장을 입력해주세요 : 조선중앙통신은 2일 "조선 대표단이 8월30일 제네바 군축회의 3기 전임회의에서 최근 성과적으로 진행된 중장거리전력탄도로켓 발사와 관련해 조직된 유엔 안전보장이사회 의장상영을 전면 배격하는 결의적 입장을 밝혔다"고 말했다.
입력문장과 유사한 것끼리 문장을 출력할까요? : 1

조선중앙통신은 2일 "조선 대표단이 8월30일 제네바 군축회의 3기 전임회의에서 최근 성과적으로 진행된 중장거리전력탄도로켓 발사와 관련해 조직된 유엔 안전보장이사회 의장상영을 전면 배격하는 결의적 입장을 밝혔다"고 말했다.
유사도는 100.0% 입니다.

회전을 했는데 그것은 안 밝혔다"고 말했다.
유사도는 88.88888888888889% 입니다.

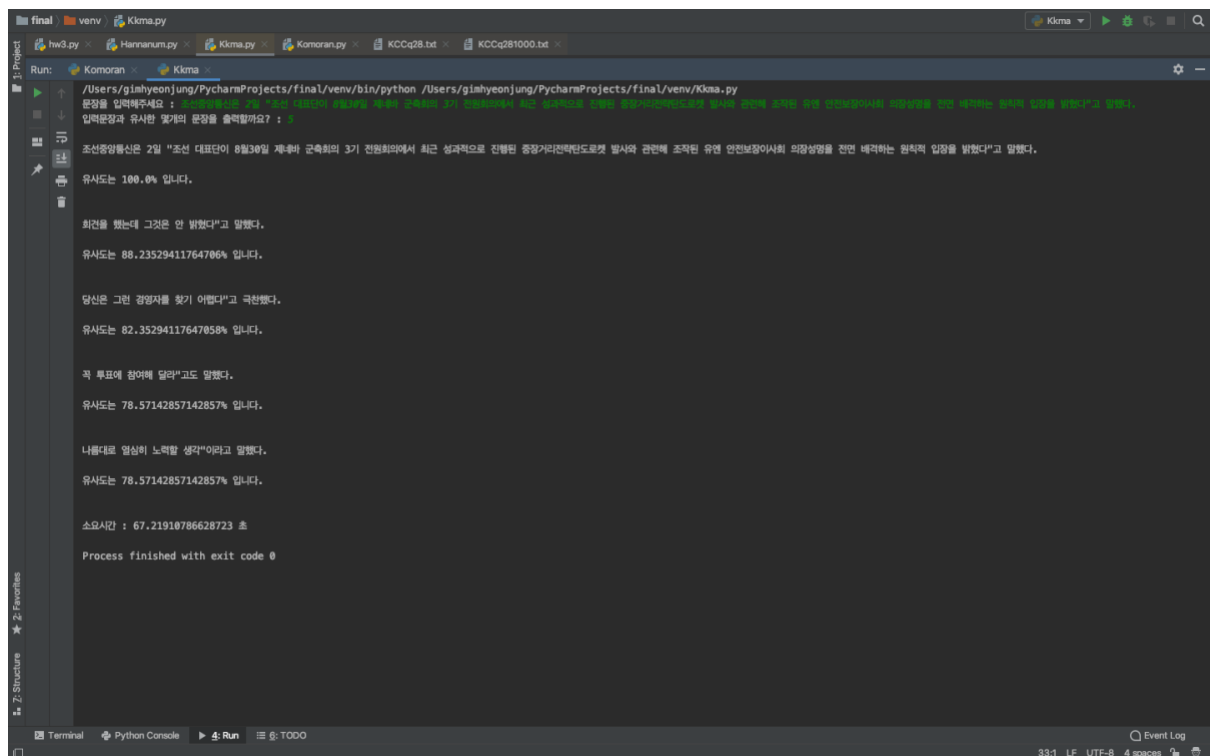
인피니티 워드는 약 20명이 직장을 잃었다"고 보도했다.
유사도는 83.33333333333334% 입니다.

그래서 협치와 통합은 시대정신"이라고 강조했다.
유사도는 83.33333333333334% 입니다.

대한축구협회는 3일 "UFF로부터 킴스킵 경기 관련 사과 공문을 접수했다"고 밝혔다.
유사도는 82.6086956521739% 입니다.

소요시간 : 10.841928958892822 초
Process finished with exit code 0
```

Kkma class 이용시



```
Run: Komoran x Kkma x
/Users/gimhyeonjung/PycharmProjects/final/venv/bin/python /Users/gimhyeonjung/PycharmProjects/final/venv/Kkma.py
문장을 입력해주세요 : 조선중앙통신은 2일 "조선 대표단이 8월30일 제네바 군축회의 3기 전임회의에서 최근 성과적으로 진행된 중장거리전력탄도로켓 발사와 관련해 조직된 유엔 안전보장이사회 의장상영을 전면 배격하는 결의적 입장을 밝혔다"고 말했다.
입력문장과 유사한 것끼리 문장을 출력할까요? : 1

조선중앙통신은 2일 "조선 대표단이 8월30일 제네바 군축회의 3기 전임회의에서 최근 성과적으로 진행된 중장거리전력탄도로켓 발사와 관련해 조직된 유엔 안전보장이사회 의장상영을 전면 배격하는 결의적 입장을 밝혔다"고 말했다.
유사도는 100.0% 입니다.

회전을 했는데 그것은 안 밝혔다"고 말했다.
유사도는 88.23529411764706% 입니다.

당신은 그런 경향자를 찾기 어렵다"고 극찬했다.
유사도는 82.35294117647058% 입니다.

꼭 투표에 참여해 달라"고도 말했다.
유사도는 78.57142857142857% 입니다.

나를대로 열심히 노력할 생각"이라고 말했다.
유사도는 78.57142857142857% 입니다.

소요시간 : 67.21910786628723 초
Process finished with exit code 0
```

결과 분석

1000줄 단위의 한국어 파일에서 입력문장에 대한 유사도를 검색 시

1. Hannamum class 이용시 **14.4초**의 출력시간이 나왔다.
2. Komoran class 이용시 **10.8초**의 출력시간이 나왔다.
3. Kkma class 이용시 **67.2초**의 출력시간이 나왔다.

1) 처리속도

분석 시간에 있어 method 및 구현방법 차이로 인한 시간 차이가 나오지 않도록 class의 import와 객체 생성을 제외한 다른 부분은 통일시켰다.

1000줄 단위의 분석에 있어서도 Komoran class와 Kkma class는 많은 시간 차이가 난다.

대용량 파일에서는 무수히 많은 시간이 차이가 날 것이다.

속도면에서 Komoran class가 월등히 좋다는 것을 알수 있다.

Hannanum class의 경우 Komoran class보다는 처리속도가 느리나 상대적으로 좋은 속도를 보여주고 있다.

2) 정확성

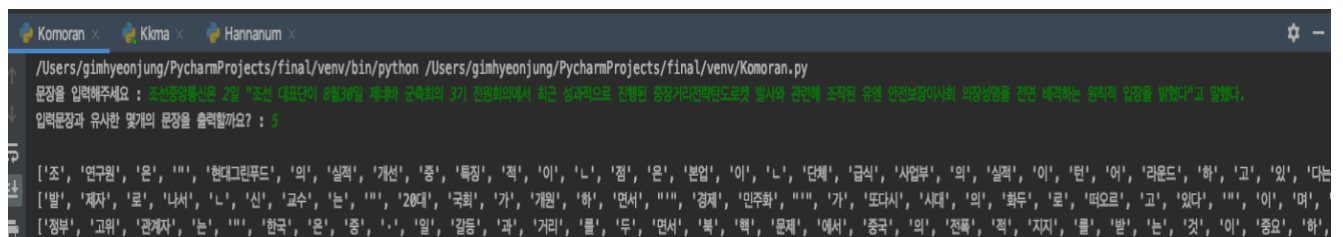
세 가지 class에서 입력 파일의 형태소 분석을 하였을 때 결과이다.

입력 파일의 첫번째 줄의 형태소 분석을 예시로 들어보면

“ 조 연구원은 "현대그린푸드의 실적 개선 중 특징적인 점은 본업인 단체급식 사업부의 실적이 턴어라운드하고 있다는 점"이라며 "올해 1 분기와 2 분기 단체급식 매출은 전년대비 각각 12.5%, 4.9% 역성장했지만, 3 분기에는 매출이 전년대비 5.6% 증가할 것으로 판단한다"고 말했다.

Komoran class의 경우

‘현대그린푸드’ 등 복합명사를 형태소 단위로 분석해 주지 못한다.



```
Komoran x Kkma x Hannanum x
/Users/ginhyeonjung/PycharmProjects/final/venv/bin/python /Users/ginhyeonjung/PycharmProjects/final/venv/Komoran.py
문장을 입력해주세요 : 조연구원은 "현대그린푸드의 실적 개선 중 특징적인 점은 본업인 단체급식 사업부의 실적이 턴어라운드하고 있다는 점"이라며 "올해 1 분기와 2 분기 단체급식 매출은 전년대비 각각 12.5%, 4.9% 역성장했지만, 3 분기에는 매출이 전년대비 5.6% 증가할 것으로 판단한다"고 말했다.
입력문장과 유사한 몇개의 문장을 출력할까요? : 1

['조', '연구원', '은', '"', '현대그린푸드', '의', '실적', '개선', '중', '특징', '적', '이', '나', '점', '은', '본업', '이', '나', '단체', '급식', '사업부', '의', '실적', '이', '턴', '어', '라운드', '하', '고', '있', '다', '는', '점', '이', '라', '며', '다', '"', '올', '해', '1', '분', '기', '와', '2', '분', '기', '단', '체', '급', '식', '매', '출', '은', '전', '년', '대', '비', '각', '각', '1', '2', '5', '%', '4', '9', '%', '역', '성', '장', '했', '지', '만', '3', '분', '기', '에', '는', '매', '출', '이', '전', '년', '대', '비', '5', '6', '%', '증', '가', '하', 'л', '것', '으', '로', '판', '단', '하', '나', '다', '"', '고', '말', '했', '다', '']
```

반면 Kkma class의 경우

‘현대그린푸드’를 ‘현대’, ‘그린’, ‘푸드’로 나누어 복합명사를
형태소 단위로 잘 분석하고 있음을 알 수 있다.

Hannanum class의 경우 Kkma class의 정확성은 아니지만
Komoran class에 비해 비교적 좋은 정확성을 가지고 있다.

이를 통하여

빠른 속도를 원한다면 Komoran class,

속도는 느리지만 정확한 형태소 분석을 토대로 유사도를 계산하고자
하면 Kkma class

속도와 정확성을 평균적으로 둘다 챙기고자 하면

Hannanum class를 사용하는 것이 효율적임을 알 수 있다.