정보검색

**검색엔진 만들기**

컴퓨터학부 2019117865

김나형

1. **DocId, Title, content 파싱**

<https://hiseon.me/python/python-string-parse/> 을 참고하여 라이브러리인 Parse를 설치하여 주어진 full\_corpus.hwp 파일의 100개의 docs를 파싱하기로 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

우선, 파일을 읽어오기 위해 주어진 full\_corpus.hwp를 corpus.txt로 저장한 후, 한 줄씩 읽어와서 doc Id와 title, content를 분리하는 파싱 작업부터 시작하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

하지만 파싱을 하는 도중 위와 같은 에러가 발생하였다. 파싱이 제대로 되지않고 None이 계속 뜨는 일이 발생하였다. 주어진 한글 파일을 그대로 복사 붙여넣기 하였더니 </title> 뒤에 띄어쓰기 등이 있는 경우 주어진 full\_corpus 파일에 오류가 발생한 것이었다. 해당 파일 파싱을 원활히 파싱하기 위해 조금 수정하였다.

<title>5. 하인리히 뵐<title> 등의 타이틀도 <title>5. 하인리히 뵐</title>로 수정하였다. <title>27.야세르 아라파트</title>와 같이 점(.) 뒤 space가 없는 경우도 수정하여 컴퓨터가 파싱을 할 수 있도록 통일시켜주었다.

그 후 위에 다운받은 parse 라이브러리를 이용하여 doc Id, title, content를 파싱하는 작업을 시행하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

출력할 경우 doc Id, title, content 모두 잘 파싱되어 하나의 딕셔너리 배열에 저장되어 출력되는 것을 확인할 수 있다.

1. **형태소 분석**

조사와 같은 frequent terms는 가치가 없기 때문에 term frequency를 고려할 필요가 없기 때문에 명사, 숫자, 동사만을 형태소 분석에서 추출하기로 하였다. <https://mr-doosun.tistory.com/22> 를 참고하여 성능을 비교해본 결과KoNLPy의 Okt 라이브러리 사용하기로 하였다.

<https://konlpy.org/ko/latest/#start> 공식 문서를 따라가며 환경을 먼저 세팅해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

*KoNLPy 라이브러리를 설치하는 모습*

Title과 content 둘 다 검색이 가능하도록 만들어야 한다. 따라서 저장한 title과 content 모두 okt를 이용하여 형태소 분석을 해준다.

1. **Doc의 Tf, idf 계산**

**3-1) tf 계산**

Td-idf를 이용하여 weight를 구해 rank를 출력하도록 할 것이다.

Tf 계산은 term이 doc에 몇 번 나타나느냐가 중요하다. 자주 나타날수록 weight가 커진다. 이를 위해서 우선 doc 들을 돌면서 파싱한 term이 어떤 doc에서 나오는지부터 세야한다. 따라서 2차원 배열을 사용해 어떤 term이 어떤 docID에 나오는지를 찾아 그 값을 세도록 하여 tf를 구할 수 있었다.

Tf의 weight는 횟수에 비례하지 않고 log를 사용하여 계산된다. Wt,d = 1 + log10 tft,d 의 식을 이용하여 tf의 weight를 계산한다. Log 계산을 위해서 math 라이브러리를 import하여 계산에 사용하였다. 이렇게 계산된 값을 리스트에 다시 재저장하여 tf weight 계산값을 쉽게 쓸 수 있도록 하였다.

**3-2) idf 계산**

Idf는 rare term, 그러니까 doc에 term이 적게 나올수록 그 term은 more informative하다는 것을 이용한 weight 계산법이다. Idt,f = log10(N/df,t)의 식을 사용해 계산한다. 따라서 위에서 구한 tf를 이용하고 전체 doc의 개수인 100을 N으로 두고 계산한 값을 구하고 뒤에서 계산하는 과정에서 쓸 수 있도록 voca의 정보가 들어있는 딕셔너리에 append한다.

1. **Query 분석 및 query tf-idf 계산**

모든 term들의 tf와 idf, 그리고 그에 대한 weight를 계산했지만 모든 tf \* idf를 계산하지는 않는다. 왜냐하면 doc에는 수백수천개의 term들이 있는데 이 term들에 대한 tf \* idf를 모두 구하기에는 너무 힘들고 불필요한 일을 하는 것이다. 따라서 query 분석부터 시작한다.

Query 분석은 위에서와 같이 okt를 이용한 형태소 분석부터 query의 tf와 weight를 계산해준다. 그리고 idf와 같은 경우에는 query는 idf를 구할 수 있을 정도로 rare도에 차이가 없다. 따라서 query의 tf, 전체 term의 idf를 이용하여 query term들의 tf-idf를 계산해준다.

1. **Length Normalization**

이제 우리가 원하는 query와 doc의 일치도를 구해 가장 일치도가 높은 것을 rank를 높은 것으로 출력하도록 한다.

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

위의 식을 이용해 구해주기 위해 L2 norm부터 구하기로 하였다.

L2 norm을 구하기 위해 반복문을 돌며 math의 pow를 사용하여 제곱과 루트를 구해준다. Query L2 norm과 Doc L2 norm을 구해 그리고 지금까지 구해둔 값들을 이용해 cos(q, d)를 구해준다. 계산식을 코딩하는데 조금 헷갈림이 있을 뿐만이 아니라 각 doc마다 반복문을 돌리며 cos 값을 구해야했다. 따라서 q는 q대로 구하고 d는 d대로 구한 후 반복문을 사용하여 둘을 곱해주었다. 이렇게 구한 값을 더해서 cos값을 구한다.

1. **Rank 출력 및 성능 테스트**

Sort 함수를 이용하여 내림차순으로 정렬하여 상위 5개를 출력하도록 하였다.

테스트를 위하여 간단한 쿼리 몇개를 이용해 성능을 테스트하였다. 가장 쉬운 1번 doc의 title인 ‘지미 카터’부터 검색을 해보았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

예상대로 1번 doc이 rank 1에 위치하였고 나머지는 0.0으로 전혀 관련성이 없는 상태로 출력되었다.

이번엔 제목이 아닌 content 검색을 하기 위해 그리스를 검색해보았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Doc 28은 그리스, 72는 고대 그리스의 철학자인 아리스토텔레스(, 100은 그리스 신화의 신 아레스, 66은 고대 그리스의 철학자 소크라테스, 66은 그리스 신화의 영웅 오디세우스에 대한 내용이 출력된다.

다음은 좀 더 어려운 쿼리로 성능 테스트를 실시하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

아폴론 남매인 신의 이름은? 을 검색하였을 때 doc ID와 계산된 일치율이 출력되는 것을 확인할 수 있다. Doc 99의 경우 아르테미스에 관한 내용으로 아르테미스는 아폴론과 남매지간이기에 원하는 결과값을 잘 출력한 것을 확인할 수 있다. Doc 42는 제우스, 59는 포세이돈으로 유사한 내용들이 위쪽에 출력됨을 확인할 수 있다.