한글 텍스트 전처리 하기

이상엽

이번 글에서는 영어가 아닌 한글 텍스트 데이터를 전처리하는 것에 대해서 살펴보도록 하겠습니다.

한글의 형태론적 특성

한글 텍스트 분석에 필요한 전처리 과정을 설명하기 이전에 먼저 한글의 기본적인 구조를 살펴보도록 하겠습니다. 기본적인 구조를 알고 있어야 한글 처리를 보다 쉽게 이해할 수 있고, 보다 쉽게 수행할 수 있기 때문입니다.

한글 텍스트의 분석을 이해하기 위해서는 한글의 형태론적 특성을 아는 것이 필요합니다. 유현경 외 (2015)에 따르면, 형태론은 "단어가 어떤 구조로 되어 있는지 단어의 하위 부류들은 어떤 것들이 있는지 등 단어와 관련한 언어학적 사실들에 대해 연구하는 언어학의 하위 분야"로 정의됩니다. 간단하게 얘기하면, 단어의 형태와 구조를 연구하는 학문이라고 생각할 수 있습니다.

한글에서의 하나의 문장은 어절로 구성되어 있고, 어절은 단어들의 조합으로 구성이 되며, 단어들은 형태소의 조합으로 구성이 됩니다. 예를 들어, 다음과 같은 문장이 있다라고 가정을 하겠습니다.

"철수가 밥을 먹다"

위의 문장을 어절로 구분하면, '철수가', '밥을', '먹다'로 구분이 되며, '철수가'라는 어절은 '철수' + '가' 라는 두 개의 단어로 구성이 되어 있고, '밥을'은 '밥'+ '을', '먹다'는 '먹다'라는 단어로 구성되어 있습니다. '철수'나 '가'라는 단어는 하나의 형태소로 구성이 된 단어입니다. 하지만, '먹다'라는 단어는 '먹'이라는 형태소 (어간)과 '다' 라는 형태소 (어미)로 구성되어 있습니다.

단어는 "최소의 자립 형식"으로 정의 되고, 이는 의미적으로나 문법적으로 자립하여 사용될 수 있는 최소 단위를 의미합니다(유현경 외, 2015). 단어는 문법적 성질에 따라 몇 가지로 구분되는데, 이를 품사라고 합니다. 단어의 품사에는 명사, 대명사, 수사, 형용사, 동사, 관형사, 부사, 조사, 감탄사 등의 9 가지가 존재하며, 각 품사는 단어가 문장에서 하는 역할에 따라 체언, 용언, 수식언, 관계언, 독립언으로 구분됩니다. 체언은 문장의 몸과 같은 주어와 목적어의 역할을 하며, 명사, 대명사, 수사 등의 품사가 포함되고, 용언은 문장의 서술어 역할을 하며, 형용사와 동사의 단어가 포함됩니다. 수식언은 체언이나 용언을 수식하는 역할을 하며, 부사와 관형사가 해당됩니다. 관계언에는 말들 간의 관계를 나타내는 조사가 있으며, 감탄사는 독립언에 해당됩니다.

하나의 단어는 한 개 이상의 형태소로 구성됩니다. 형태소는 "의미를 갖는 최소 단위"로 정의되는데, 여기서의 '의미'는 어휘적인 의미뿐 아니라 형식적인 혹은 문법적인 의미도 포함합니다 (유현경 외, 2015). 예를 들어, 어미는 단어는 아니지만, 어절을 생성하는데 있어서 문법적인 의미를 갖기 때문에 형태소로 구분됩니다.

Comment [S11]: 그래서 조사도 하나의 단어가 됩니다.

Comment [S12]: '순 살코기'에서의 순과 같이 체언을 수식하는 단어

Comment [S13]: 어미는 문법적인 의미는 갖지만, 자립하여 사용될 수 없기 때문에 단어로는 간주되지 않습니다.

형태소는 자립성의 여부와 의미의 허실에 따라 그 유형이 달라집니다. 자립 형태소는 다른 형태소와 결합하지 않고도 사용될 수 있는 형태소로, 위의 예에서 '철수' 와 '밥' 등이 포함됩니다. 반대로, 의존 형태소는 다른 형태소와 결합되어 사용되는 형태소를 의미하고, 위의 예에서 '가', '먹' 등이 해당됩니다. 실질적인 의미를 갖느냐에 따라도 구분이 되는데, 실질 형태소는 실질적인 의미를 가진 형태소가 해당됩니다 (예, '철수', '밥', '먹'). 실질 형태소에는 자립 형태소뿐 아니라 의존 형태소(예, '먹')도 포함될 수 있습니다. '먹'과 같은 의존 형태소는 홀로 사용될 수는 없지만, 의미를 지니고 있기 때문입니다. 형식 형태소 (문법 형태소라고도 함)는말과 말 사이의 형식적인 (문법적인) 관계를 표시하는데 사용되는 형태소를 의미합니다 (예, '는', '와', '다').

하나의 단어를 형성하는 형태소는 그 관점에 따라서 불리는 이름이 달라집니다. 단어 형성론 관점에서는 단어를 구성하는 형태소를 어근과 접사로 구분하는 반면, 활용론 관점에서는 어간과 어미로 구분하게 됩니다 (유현경 외, 2015).

한글과 영어의 차이

언어학에서는 한글을 교착어, 그리고 영어를 굴절어라고 합니다. 교착어는 하나의 단어 (혹은 어절)가 하나 이상의 형태소의 조합으로 구성되는 언어를 의미하고, 굴절어는 하나의 단어 (혹은 어절)가 하나의 형태소로 구성되어 있고, 그 형태의 변형에 따라서 다른 기능을 하는 언어를 의미합니다.

즉, 영어는 하나의 문장을 띄어쓰기를 기준으로 분리하면 그 결과는 단어 (혹은 형태소)가 됩니다. 하지만, 한글 문장은 띄어쓰기를 기준으로 어절의 합으로 구성되어 있습니다. 그리고, 하나의 어절은 하나 이상의 단어와 형태소의 조합으로 구성되어 있는 것입니다. 그렇기 때문에 한글 텍스트에서 단어들을 추출해 내는 것이 영어보다 어렵습니다. 더욱이 한글의 단어는 형태소라는 그 보다 작은 단위로 구성되어 있습니다. 한글 같은 경우 텍스트를 형태소로 구분하고 그에 대한 품사를 찾는 것을 형태소 분석이라고 합니다.엄밀하게 따지면 단어와 형태소는 다른 개념이지만, 본 텍스트 분석에서는 편의상 단어와 형태소를 같은 개념이라고 간주하도록 하겠습니다.

수집된 데이터의 전처리 (Korean_preprocessing.ipynb 참조)

한글 텍스트 분석을 위해서는 영어와 마찬가지로 전처리 과정을 거쳐야 합니다. 전처리 과정의 결과물은 보통 최종 분석에 사용하고자 하는 "불용어가 제거된 특정한 품사들의 단어들"이 됩니다.

한글 텍스트도 영어와 비슷한 전처리 과정을 거치기는 하지만 stemming, lemmatization, case conversion 등은 별도로 수행하지 않습니다. 한글과 관련된 주요 전처리 과정은 아래와 같습니다.

① 불필요한 기호 제거하기 (예, !, ., ", ; 등)
② 형태소 분석 [tokenization + POS tagging]
③ 불용어 제거하기

①과③은 영어와 비슷한 방식으로 처리하시면 됩니다.

Comment [S14]: Korean_preprocessing_ver1.py

Comment [S15]: 최종적으로 사용되는 분석 방법이 무엇이냐에 따라서 어떠한 혹은 얼마만큼의 전처리를 해야하는지는 달라집니다.

Comment [S16]: 어간 찾기 (stemming), 원형 찾기 (lemmatization)은 별도로 수행되지 않고, 형태소 분석이라는 과정을 통해서 수행됩니다. 그리고 형태소 분석은 Python 에서 제공되는 형태소 분석기를 사용해서 수행하게 됩니다.

Comment [S17]: 한글은 대소문자를 구분하지 않기 때문에 case conversion 은 수행되지 않습니다.

여기서는 특정 기사[1] 를 예로 사용하여 한글 전처리 과정을 설명하도록 하겠습니다. 해당 기사는 웹 스크래핑 방법을 사용해서 naver_news.txt 파일로 저장해 두었습니다. 파일의 내용을 불러오기 위해 아래와 같은 코드를 사용합니다.

```
with open('naver_news.txt', 'r', encoding='utf8') as f:
    content = f.read()
```

① 불필요한 기호 제거하기 (예, !, ., “, ; 등)

불필요한 기호를 제거하기 위해서는 아래 두가지 방법 중 하나를 사용할 수 있습니다.

- 문자열 함수인 replace() 사용하기
- 정규식 사용하기 (re 모듈의 sub() 함수 사용하기)

Comment [S18]: 정규식 관련 내용은 Ch3 를 참고하세요.

replace() 사용하기

replace() 함수를 사용하여 content 에 있는 기사 내용에서 마침표와 쉼표 등의 불필요한 기호를 없애기 위해서는 아래와 같은 코드를 사용할 수 있습니다.

Comment [S19]: 정규식을 사용할 수도 있습니다. 예) filtered_content = re.sub(r'[^\w\d\s]','',content)
만약, 문장을 구분하는데 사용되는 기호 (., ?, !) 는 삭제하고 싶지 않은 경우에는 filtered_content = re.sub(r'[^\.\?\!\w\d\s]','',content) 를 사용할 수 있습니다.

```
filtered_content = content.replace('.', '').replace(',','').replace("'","").replace('·', ' ').replace('=','')
```

replace() 함수를 사용하는 경우에는 한 번에 하나의 기호만을 삭제할 수 있다는 단점이 있습니다. 한번에 여러개의 기호를 삭제하기 위해서는 re 모듈에서 제공되는 sub() 함수를 사용할 수 있습니다.

```
import re
```

content 에 저장되어 있는 텍스트에서 문자 character 와 숫자 그리고 공백문자를 제외한 나머지 모든 기호를 삭제하기 위해서는 아래와 같이 코딩할 수 있습니다.

```
filtered_content = re.sub(r'[^\w\d\s]','',content)
```

만약, 텍스트를 문장 단위로 분리하여 분석에 사용해야 하는 경우는, 아래와 같이 문장을 구분하는데 사용되는 기호들(., ?, !)은 삭제되면 안됩니다. 이러한 경우에는 아래와 같이 코딩할 수 있습니다.

```
filtered_content = re.sub(r'[^\.\?\!\w\d\s]','',content)
```

② 형태소 분석 [tokenization + PoS tagging]

---

1 http://news.naver.com/main/ranking/read.nhn?mid=etc&sid1=111&rankingType=popular_day&oid=001&aid=0009277690&date=20170519&type=1&rankingSeq=6&rankingSectionId=102

불필요한 기호를 삭제한 후 수행해야 하는 작업은 영어 텍스트에서의 tokenization 과 PoS tagging 입니다. 한글의 경우는 이러한 두 개의 과정을 형태소 분석이라는 방법을 통해서 수행합니다.

형태소 분석이란 텍스트를 어절 단위로 분리하고 각 어절을 구성하는 형태소들을 인식하고, 불규칙 활용이나 축약, 탈락 현상이 일어난 형태소는 원형을 복원하는 과정을 의미합니다 (강승식, 2002). 정확한 정의에는 형태소 분석은 품사 태깅의 내용은 포함하지 않습니다. 하지만, 일반적으로는 형태소 분석이라고 하면 어절을 형태소 단위로 구분하여 원형을 찾고, 각 형태소의 품사를 찾는 과정까지를 의미합니다.

굴절어인 영어와 달리 한글은 교착어입니다. 하나의 어절이 하나 이상의 형태소의 조합으로 이뤄져 있습니다. 보통, 명사, 동사 등의 어간(stem)에 조사, 어미 등의 기능어가 추가되어 어절 형성이 됩니다. 형태소 분석은 일반적으로 형태소 사전과 형태소 변형과 조합에 대한 규칙을 기반으로 수행됩니다. 형태소 사전은 크게 어휘 형태소 사전과 문법 형태소 사전으로 구분됩니다 (강승식, 2002).

형태소 분석과 함께 종종 사용되는 표현이 어휘 분석 (lexical analysis)입니다. 어휘 분석은 어휘 사전에 수록된 형태의 단어를 분리하는 것을 의미합니다 (강승식, 2002). 두 개 이상의 형태소가 독립된 의미를 가지는 하나의 단어를 나타내기 위해 사용되는 경우, 이를 형태소 단위로 분리할 것인지 (형태소 분석), 그냥 단어 그대로 분리할 것인지 (어휘 분석)의 차이라고 볼 수 있습니다. 보통은 형태소 분석과 어휘 분석 비슷한 의미로 사용됩니다 (강승식, 2002). 어휘와 형태소 분석의 차이에 대한 예제 추가할 것 (강승식, 2002 참조).

Python 의 경우, 한글 텍스트의 형태소 분석은 KoNLPy 라는 모듈을 통해서 할 수가 있습니다. Windows OS 의 경우 해당 모듈을 설치하는 방법이 조금 복잡할 수 있습니다. 설치하는 방법은 http://konlpy.org/en/latest/install/ 를 참조하거나 별도로 제공되는 설치 매뉴얼[2]을 참조하면 됩니다. 여기서는 KoNLPy 가 설치되어 있다고 가정을 하고 진행을 하도록 하겠습니다.

KoNLPy 에서는 크게 5 가지의 서로 다른 형태소 분석기(또는 품사 tagger 라고 불림)를 제공합니다. 이에 대한 정보는 http://konlpy.org/en/latest/morph/#pos-tagging-with-konlpy 에서 확인할 수 있습니다. 각 형태소 분석기는 그 장단이 있기 때문에 여러분들 작업 특성에 맞는 것을 선택하여 작업하면 됩니다.[3] 각 형태소 분석기는 별도의 말뭉치 사전과 형태소 사전[4]과 서로 다른 알고리즘을 사용해서 형태소 분석을 합니다. 5 개의 형태소 분석기 중에서 본 글에서는 일반적으로 많이 사용되는 Komoran()와 Twitter() 에 대해서 설명 하겠습니다.

앞의 전처리 과정을 거쳐서 filtered_content 에 저장되어 있는 기사 내용을 가지고 형태소 분석을 해보도록 하겠습니다. 이를 위해서는 먼저 아래 코드를 이용해서 konlpy 를 import 해야 합니다.

```
import konlpy.tag
```

[2] konlpy_installation_manual.pdf 참조

[3] Mecab 은 윈도우에서 지원이 안됩니다.

[4] 말뭉치 사전은 쉽게 말하면 한글 어휘와 어휘 특성의 저장소입니다.

Comment [S110]: 어휘 분석이라는 것도 있습니다. 이는 형태소 사전이 아니라 어휘사전을 사용해서 어휘를 찾아내고, 각 어휘에 품사를 tagging 하는 과정을 말합니다. 어휘는 보통은 형태소보다 큰 단위이며, 보통은 단어를 의미합니다.
송민 (2017)에 따르면 '수행하다'라는 어절의 경우
형태소 분석: '수행' + '하'+ '다'
어휘 분석: '수행하' + '다'

Comment [S111]: 형태소 분석은 주어진 어절에 대한 가능한 모든 형태소를 반환하고 각 형태소의 품사 정보를 제공합니다. 하지만, 이러한 형태소와 품사 후보들 중에서 문장에서 어떠한 형태소 혹은 품사로 사용되었는지를 추가적으로 작업해야 하는데, 이러한 작업을 품사 태깅이라고 합니다 (이중영 외, 1999; 심준혁 외, 1999).

Comment [S112]: 어휘 형태소 사전은 어휘 형태소에 대한 품사 정보, 불규칙 정보, 그리고 타 형태소와의 결합성을 타나내는 자질들의 정보를 담고 있습니다.

Comment [S113]: 조사와 선어말 어미, 어말 어미, 접사에 대한 정보를 수록한 사전입니다.

Comment [S114]: 정의: 일정한 범위 안에서 쓰이는 낱말의 총체

Comment [S115]: Twitter()는 형태소 분석라기 보다는 어휘 분석기에 더 가깝습니다.

위와 같이 import 를 한 다음에는 아래 코드를 사용해서 형태소 분석을 할 수가 있습니다. 아래 코드는 Komoran 에서 제공되는 pos()라는 함수를 사용해서 형태소 분석을 한 경우입니다.

```
komoran = konlpy.tag.Komoran()
komoran_morphs = komoran.pos(filtered_content)
print(komoran_morphs)
```

만약 Twitter 를 이용해서 형태소 분석을 하고자 하는 경우에는 아래와 같이 해주면 됩니다.

Comment [S116]: 역시나 마찬가지로 pos() 함수를 사용합니다.

```
twitter = konlpy.tag.Twitter()
twitter_morphs = twitter.pos(filtered_content)
print(twitter_morphs)
```

이에 대한 결과는 아래와 같습니다.

[('현', 'Noun'), ('제도', 'Noun'), ('상', 'Suffix'), ('추천', 'Noun'), ('된', 'Verb'), ('후보', 'Noun'), ….]

보는 것 처럼 한글의 경우도 (단어, 품사)의 tuple 이 저장된 list 데이터가 반환이 됩니다.

미등록단어 문제

형태소 분석기를 사용하다보면 형태소 분석이 제대로 되지 않는 단어들이 있습니다. 즉, 하나의 독립된 형태소로 구분이 안되거나, 구분이 되더라도 잘못된 품사로 태깅이 되는 것입니다. 이러한 문제는 보통 해당 단어들이 형태소 분석기가 사용하는 말뭉치 사전, 형태소 사전에 등록이 되어 있지 않아서 발생하게 됩니다. 이렇게 사전에 등록되지 않아 발생하는 형태소 분석의 오류를 미등록단어(out of vocabulary) 문제라고 합니다.

예를들어 보겠습니다.

text = '파이콘은 파이썬 콘퍼런스의 약자이다'
print(twitter.pos(text))

이에 해당하는 결과는
[('파', 'Noun'), ('이콘', 'Noun'), ('은', 'Josa'), ('파이썬', 'Noun'), ('콘퍼런스', 'Noun'), ('의', 'Josa'), ('약자', 'Noun'), ('이다', 'Josa')]
와 같습니다. 파이콘이라는 단어는 하나의 고유명사임에도 불구하고, 신조어이기 때문에 Twitter 가 사용하는 사전에 등록되어 있지 않아, 제대로 형태소 분석이 되지 않았습니다.

이러한 미등록단어 문제를 해결하지 않고 텍스트 분석을 하게 되면, 그 분석 결과의 정확도는 낮아질 수 밖에 없습니다. 특히, 키워드 빈도 분석이나, 네트워크 분석 등과 같이 단어를 제대로 분리하는 것이 중요한 분석의 경우는 더욱 그렇습니다. 다른 분석 알고리즘들 (예, LDA 와 같은 토픽 모델링 알고리즘, K-Means 와 같은 군집화

알고리즘 등)도 상대적으로 영향을 덜 받을 수는 있지만, 그래도 정확한 분석을 위해서는 이러한 미등록단어 문제를 되도록이면 최대한 해결하는 것이 좋습니다.

미등록단어 문제 해결

미등록단어 문제를 해결하기 위해서는 다양한 방법이 사용될 수 있습니다. 하지만, 여러분이 기억해야 하는 것은 어떠한 방법을 사용하더라도 미등록단어의 문제를 완벽하게 해결하기는 어렵다는 것입니다. 사용될 수 있는 방법들은 크게 아래와 같습니다.

- 코딩을 통한 후처리

- 형태소 분석기가 사용하는 사전에 해당 단어들 추가하기

① 코딩을 통한 후처리

형태소 분석이 제대로 되지 않는 단어들이 특정한 패턴을 갖고 있다면, 그러한 패턴을 이용해서 그러한 단어들의 품사를 찾는 코드를 직접 작성할 수 있습니다. 하지만, 이러한 방법은 동일한 패턴을 갖는 미등록 단어가 많지 않다는 점과 미등록된 단어들이 많은 경우에는 사용하기 어렵다는 단점이 있습니다.

② 형태소 분석기가 사용하는 사전에 해당 단어들 추가하기

형태소 분석이 잘 안되는 단어들이 무엇인지 아는 경우는, 그러한 단어들을 일시적으로 형태소 분석기가 사용하는 사전에 추가할 수 있습니다.

Twitter 형태소 분석 사전에 새로운 단어 추가하기

먼저 Twitter 가 사용하는 사전에 새로운 단어를 추가하는 방법에 대해서 살펴 보겠습니다. Twitter 형태소 분석기 자체적으로 이러한 추가기능을 제공하고 있지 않아, 사전에 단어를 추가하기 위해서는 ckonlpy[5]라는 모듈을 추가적으로 설치해야 합니다. 명령 프롬프트 창에서 pip install customized_konlpy 를 실행해서 설치할 수 있습니다.

ckonlpy 를 설치한 다음에는 다음과 같이 import 하여 사용할 수 있습니다.[6]

```
from ckonlpy.tag import Twitter
twitter = Twitter()
```

사용의 예를 설명하기 위해서 위에서 살펴본 '파이콘은 파이썬 콘퍼런스의 약자이다' 텍스트를 사용하겠습니다.

```
text = '파이콘은 파이썬 콘퍼런스의 약자이다'
print(twitter.pos(text))
```

[5] 보다 자세한 내용은 https://github.com/lovit/customized_konlpy 참조

[6] ckonlpy 에서 제공되는 Twitter 형태소 분석기는 konlpy 의 원래 Twitter 형태소 분석기를 wrapping 하여 사용하는 것입니다.

위에서 본 것 처럼, 이에 해당하는 결과는 아래와 같습니다.

[('파', 'Noun'), ('이콘', 'Noun'), ('은', 'Josa'), ('파이썬', 'Noun'), ('콘퍼런스', 'Noun'), ('의', 'Josa'), ('약자', 'Noun'), ('이다', 'Josa')]

즉, '파이콘'이라는 단어가 제대로 인식되지 않습니다. 이러한 문제를 해결하기 위해서 Twitter 가 사용하는 사전에 '파이콘'이라는 단어를 명사 단어로 추가해야 합니다. 이는 ckonlpy 에서 제공되는 add_dictionary() 함수를 사용해서 할 수 있습니다 (아래 참조).

```
twitter.add_dictionary('파이콘', 'Noun')
```

add_dictionary() 함수를 통해서 새로운 단어를 등록한 이후에 아래 코드를 다시 실행시켜 보면

```
text = '파이콘은 파이썬 콘퍼런스의 약자이다'
print(twitter.pos(text))
```

아래와 같은 결과가 나옵니다.

[('파이콘', 'Noun'), ('은', 'Josa'), ('파이썬', 'Noun'), ('콘퍼런스', 'Noun'), ('의', 'Josa'), ('약자', 'Noun'), ('이다', 'Josa')]

파이콘이라는 단어가 Noun 으로 제대로 인식된 것을 확인할 수 있습니다.

add_dictionary() 함수는 일시적으로 사전에 새로운 단어를 추가합니다. Jupyter notebook 을 새롭게 실행할 때 마다 단어들을 새롭게 추가 해주어야 합니다.

아래와 같이 여러개의 단어를 한꺼번에 추가할 수도 있습니다.

twitter.add_dictionary(['파이콘', '딥러닝'], 'Noun')

Komoran 사전에 단어 추가하기

Comment [S117]: KoNLPy 0.5 이상 버전에서 지원되는 기능입니다.

Komoran 은 별도의 추가 모듈없이 자체적으로 사전에 단어를 일시적으로 추가하는 기능을 제공합니다.

from konlpy.tag import Komoran

아무런 추가 작업을 하지 않은 경우, 아래 코드의 결과는

komoran.pos('파이콘은 파이썬 컨퍼런스의 약자입니다.')

아래와 같습니다.

[('파', 'NNG'), ('이콘', 'NNP'), ('은', 'JX'), ('파이썬', 'NNP'), ('컨퍼런스', 'NNG'), ('의', 'JKG'), ('약자', 'NNP'), ('이', 'VCP'), ('ㅂ니다', 'EF'), ('.', 'SF')]

Twitter 와 마찬가지로 '파이콘' 이라는 단어가 제대로 분석이 되지 않습니다.

이러한 문제를 해결하기 위해서 '파이콘'이라는 단어를 명사로 Komoran 이 사용하는 사전에 일시적으로 추가할 수 있습니다. 이를 위해서는 아래와 같이 Komoran() 생성자를 호출할 때, 미등록단어의 품사 정보가 들어있는 텍스트 파일 혹은 문자열 데이터를 인자로 제공해야 합니다.

```
komoran = Komoran(userdic='dic.txt')
```

Comment [S118]: dic.txt 파일에 저장된 내용은 아래와 같습니다. 이와 같은 format 을 따라야 합니다.
파이콘\tNNP
코딩\tNNG
데이터분석\tNNP
…

이렇게 한 후 pos()를 통해 형태소 분석을 하면 아래와 같은 결과를 얻을 수 있습니다.

```
[('파이콘', 'NNP'), ('은', 'JX'), ('파이썬', 'NNP'), ('컨퍼런스', 'NNG'), ('의', 'JKG'), ('약자', 'NNP'), ('이', 'VCP'), ('ㅂ니다', 'EF'), ('.', 'SF')]
```

Komoran 의 경우는 nouns()라는 함수를 사용해서 명사의 단어들만 추출할 수 있습니다.

```
komoran.nouns(u'파이콘은 파이썬 컨퍼런스의 약자입니다.')
```

```
['파이콘', '파이썬', '컨퍼런스', '약자']
```

Twitter 의 template 사용하기

만약 아래와 같은 text 가 있다고 가정을 합시다.

```
text = '딥러닝 CNN 알고리즘은 좋습니다.'
```

그리고 기존의 Twitter 분석기에 '딥러닝'이라는 단어와 'CNN'[7] 이라는 단어가 등록되어 있지 않은 경우에 우리는 다음과 같이 두 단어를 추가할 수 있습니다.

```
twitter.add_dictionary(['딥러닝', 'CNN'], 'Noun')
```

추가한 후에 형태소 분석을 해보면 아래와 같은 결과가 나옵니다.

```
twitter.pos(text)
```

```
[('딥', 'Noun'),
 ('러닝', 'Noun'),
 ('CNN', 'Alpha'),
 ('알고리즘', 'Noun'),
 ('은', 'Josa'),
 ('좋', 'Adjective'),
 ('습니다', 'Eomi'),
 ('.', 'Punctuation')]
```

[7] CNN = Convolutional Neural Network (딥러닝 알고리즘의 한 종류)

하지만, 아직도 우리가 원하는 결과가 나오지 않습니다. 왜 그럴까요? text 에 들어있는 문장에는 띄어쓰기가 제대로 되지 않아 어절이 두 개 밖에 있습니다. 하나는 '딥러닝 CNN 알고리즘은' 이고 다른 하나는 '좋습니다.' 입니다. Twitter 형태소 분석기는 어절 단위로 형태소의 품사를 찾습니다. 그런데, 어절의 구조가 이미 정해진 구조 (template 이라고 함)에 맞지 않으면, 형태소 분석을 제대로 하지 못합니다. 첫 번째 어절의 경우는 'Noun' + 'Noun' + 'Noun' + 'Josa'의 형태입니다. 즉, 이러한 형태에 대한 template 이 Twitter 형태소 분석기에 저장되어 있어야지만, 해당 어절을 우리가 원하는 형태로 분석할 수가 있는 것입니다. 하지만, 'Noun' + 'Noun' + 'Noun' + 'Josa'의 template 은 기존 Twitter 형태소 분석기에 저장 되어 있지 않습니다. Twitter 형태소 분석기에 저장되어 있는 templates 의 리스트는 다음과 같이 확인할 수 있습니다.

```
twitter._customized_tagger.templates
```

결과는

```
[('Noun',),
 ('Noun', 'Noun'),
 ('Noun', 'Josa'),
 ('Noun', 'Josa', 'Noun'),
 ('Noun', 'Noun', 'Josa'),
 ('Modifier', 'Noun'),
 ('Modifier', 'Noun', 'Noun'),
 ('Modifier', 'Noun', 'Josa'),
 ('Modifier', 'Noun', 'Noun', 'Josa')]
```

보는 것 처럼 우리가 찾는'Noun' + 'Noun' + 'Noun' + 'Josa' 은 없습니다. 이를 추가하기 위해서 아래 코드를 실행합니다.

```
twitter._customized_tagger.add_a_template(('Noun', 'Noun', 'Noun', 'Josa'))
```

위의 실행한 후에 다음 코드를 다시 실행해 보면

```
text = '딥러닝 CNN 알고리즘은 좋습니다.'
twitter.pos(text)
```

아래와 같이 우리가 원하는 결과가 나옵니다.

```
[('딥러닝', 'Noun'),
 ('CNN', 'Noun'),
 ('알고리즘', 'Noun'),
 ('은', 'Josa'),
 ('좋', 'Adjective'),
 ('습니다', 'Eomi'),
 ('.', 'Punctuation')]
```

문서에서 미등록 명사 단어 추출하기

앞의 방법은 한가지 큰 단점이 존재합니다. 바로, 새로운 단어를 추가하기 위해서는 형태소 분석이 잘 안된 단어가 무엇인지를 알아야 한다는 것입니다. 이러한 단어가 많지 않고, 발견하기 쉽다면 직접 그러한 단어들을 위의 방식으로 추가하는 것이 가능합니다. 하지만, 텍스트의 양이 많은 경우에는 분석이 제대로 되지 않는 단어들이 많아질 것이고, 그러한 단어들을 사람이 일일이 발견한다는 것은 비효율적인 뿐 아니라, 발견하는 것도 쉽지 않습니다. 이러한 경우 우리는 미등록 단어들의 특성을 파악하여 문제를 알고리즘을 통해 해결하는 방법을 사용할 수 있습니다.

여러 텍스트 분석에서 가장 중요하게 사용되는 단어들은 명사입니다. 명사는 실질적인 의미를 담고 있어 문서의 특성을 잘 나타내기 때문에 그렇습니다. 이는 명사 단어들만 제대로 파악을 하더라도 많은 분석의 정확성이 향상될 수 있다는 것을 의미합니다. 그리고, 미등록 명사 단어들을 찾는 것이 다른 품사의 단어들을 찾는 것 보다 상대적으로 용이합니다.

명사들은 어떠한 특성을 갖을까요? 이를 이해하기 위해서 다음 예제 텍스트를 살펴보겠습니다.

> 철수가 친한 친구와 부산으로 여행을 갔다. 철수와 친구는 해운대에 있는 밀면으로 유명한 식당을 찾았다. 철수는 밀면을 주문했고, 친구는 우동을 주문했다. 음식을 먹은 후 그들은 해운대 해변으로 가서 바다를 구경했다.

위의 텍스테에서 사용된 명사 단어들은 '철수', '친구', '부산', '여행', '해운대', '밀면', '식당' 등이 있습니다. 이러한 명사의 주요 특성은 다음과 같습니다.

1) 명사의 출현 특성

① 주로 조사와 함께 사용된다.

명사의 단어들은 주로 조사와 사용된다는 특징이 있습니다. 위의 텍스트에서 사용된 조사로는 '-가', '-와', '-으로', '-는' 등이 있습니다. 즉, 대부분의 명사는 단독으로 사용되기 보다는 명사+조사의 형태로 사용됩니다. 그리고 많은 경우, 명사가 포함된 어절은 해당 명사 단어로 시작을 합니다.

Comment [S119]: 한글에서의 조사 목록은 xxx 에서 찾을 수 있습니다.

② 중요한 명사들은 여러번 사용된다.

많은 텍스트 분석에서, 우리는 모든 명사 단어들을 사용하기 보다는 문서 (혹은 텍스트)의 특성을 잘 나타내는 명사 단어들을 보통 사용합니다. 하나의 문서의 특성을 나타내는 명사 단어들은 보통 해당 문서에서 여러번 사용된다는 특징이 있습니다. 위의 텍스트에서는 '철수', '친구', '해운대' 등이 그러한 단어들입니다.

③ 명사는 단독으로 사용되기도 한다.

명사는 보통 조사와 함께 사용되지만, 조사 없이 단독으로 사용되기도 합니다. 위의 예제에서 '해운대 해변'의 '해운대'는 조사 없이 사용되었습니다. 고유명사 특히 사람의 이름은 단독으로 사용되는 경우가 많은데, 박근혜 대통령, 문재인 대통령, 김무성 원내대표, 이건희 회장, 이재용 부회장, 마광수 교수 등이 그러한 예입니다. 이와 같이 직책을 갖고 있는 사람들의 이름을 직책과 함께 표현할 때 사람들의 이름은 보통 조사없이 사용됩니다.

2) 명사가 포함되지 않는 어절의 특성

어절에서 명사를 추출하는데 있어, 우선적으로 명사가 포함되지 않는 어절을 배제하게 되면 명사 추출 작업을 보다 효율적으로 수행할 수 있다 (이도길 외, 2003). 특히, 어절을 구성하는 단어 중에 형용사, 부사 , 동사 등이 포함되어 있는 어절은 명사 단어를 포함하지 않을 확률이 높다. 그리고, 이러한 어절들은 Komoran 형태소 분석기를 통해서 품사 태깅이 비교적 정확하게 된다는 특징이 있다. 아래는 그러한 어절들의 예를 나타낸다.

'어쩌면', '빨리', '천천히', '느리게', '그리고', '그런데', '그러므로', '가는', '돌아오는'

3) 명사 배제 정보 사용하기

추가적으로 명사가 아닌 단어들을 배제하기 위해서 본 논문에서는 다음과 같은 명사 배제 정보를 사용하였다.

① 이도길 외 (2003)에 따르면, 단어의 받침에 다음과 같은 낱말들이 사용되는 경우, 해당 단어는 명사일 가능성이 매우 낮다고 한다.

'ㄱㅅ','ㅆ', 'ㄶ', 'ㄻ', 'ㄴㅎ','ㄵ', 'ㄹㄱ', 'ㄹㅂ','ㄿ'

② 한 음절의 단어 배제하기

한 음절의 단어는 명사가 아니거나, 의미없는 명사인 경우가 많다. 중요한 명사라고 할지라도 형태소 분석기를 통해서 명사로 간주된 한 음절의 단어는 그 단어 자체만 보고 무엇을 의미하는 단어인지 정확하게 파악할 수 없는 경우가 많다. 따라서 본 연구에서는 한 음절의 단어 혹은 어절은 배제하였다.

이러한 명사 단어의 특성을 사용해서 명사 단어들을 자동으로 추출하는 방법을 살펴보겠습니다.

1. 명사의 사용 특성을 사용

첫번째 방법은 저자가 직접 고안한 명사 추출 알고리즘입니다. 해당 알고리즘은 명사 사용 특성과 Komoran 형태소 분석기를 기반으로 명사 단어들을 추출합니다. 여러가지 형태소 분석기 중에서 Komoran 형태소 분석기를 사용한 이유는 현재 가장 지속적으로 업데이트가 이뤄지고 있는 형태소 분석기로 미등록 단어의 문제가 다른 형태소 분석기 보다 적기 때문입니다.

이를 위해서 kornounextractor 라는 모듈을 아래와 같이 설치합니다.

pip install kornounextractor

Comment [S120]: kornounextractor 는 지속적으로 그 내용이 업데이트 되고 있어서, 여러분이 이 책을 읽고 있을 때는 그 내용이 달라져 있을 수도 있습니다. 따라서, 해당 모듈에 대한 코드와 설명을 제공하는 github 사이트를 참조하기를 권합니다.

주어진 한글 텍스트 혹은 문서에 존재하는 명사를 추출하기 위해서는 kornounextractor 의 extract() 함수를 사용합니다.

해당 모듈을 설치한 다음에, 아래와 같이 명사 추출에 사용되는 extract() 함수를 import 하여 해당 명사 추출기를 사용할 수 있다.

```
from kornounextractor.noun_extractor import extract
```

extract() 함수는 다음과 같은 파라미터를 가지고 정의되어 있다.[8]

```
def extract(text, include_number = False, freq=1.0, threshold=0.3)
```

text: 분석하고자 하는 텍스트 데이터. 텍스트 데이터는 하나의 문서가 될 수도 있고, 복수 문서의 집합이 될 수도 있습니다. 본 함수는 대용량 텍스트에 대해서는 그 처리 속도가 느리므로, 작은 수의 문서들을 대상으로 작업할 것을 권장합니다.

include_number: 숫자를 포함하고 있는 단어의 추출 여부. 기본값은 False (즉, 추출하지 않는다)로 지정되어 있습니다.

freq: 추출하고자 하는 단어들이 text 에서 사용된 최소 빈도. 예를 들어, freq=3 이라고 입력하게 되면 text 에서 최소 3 번 이상 사용된 단어들만 추출하게 됩니다. 기본값은 1.0 으로 지정되어 있습니다.

threshold: 명사 추출에 사용되는 기준값. 본 알고리즘은 명사 단어를 추출할 때, 다음과 같은 값을 사용합니다.

$$\omega_i = \text{Word}_i \text{ 가 조사와 함께 사용된 어절의 수} \Big/ \text{Word}_i \text{ 가 사용된 전체 어절의 수}$$

이는 명사의 사용 특성 중, '조사와 주로 사용된다'라는 특성을 고려한 것입니다. 즉 특정 단어 (Word_i)가 조사와 함께 많이 사용될 수록 Word_i가 명사일 확률 높다라고 본 논문은 가정하는 것이지요. 본 연구에서는$\omega_i > \text{threshold}$ 인 단어들만을 명사 단어 후보로 추출하게 됩니다.

앞에서 살펴본 명사 출현 특성과 배제 정보, 그리고 Komoran 형태소 분석기를 기반으로 하여, 저자가 개발한 알고리즘은 다음의 과정을 거쳐 명사를 추출합니다.

① 불필요한 기호 없애기 등의 기본 전처리 작업
② Komoran 을 이용하여 명사 단어를 포함하지 않는 어절 제외하기
③ 위에서 언급된 명사 출현 특성을 사용해서 명사 단어 후보군 생성하기
④ 위에서 언급된 명사 배제 정보를 이용해 후보 단어들 중에서 명사가 될 수 없는 단어 제거하기

[8] https://github.com/sangyup-lee/korean_noun_extractor

⑤ 후보 단어들 중에서 대명사 제외하기

⑥ 최종 후보 단어들에 대해서 Komoran 을 적용하여 명사가 될수 없는 단어들 제거하기

⑦ 최종 명사 단어 추출

예를 들기 위해, 아래의 텍스트 데이터에서 명사를 추출해 보도록 하겠습니다.

예) '자카르타 아시안게임에서 손흥민은 빠른 드리블과 슈팅으로 한국의 우승을 도왔다. '

위의 텍스트 데이터에 대해서 본 연구에서 제안한 알고리즘의 결과를 순서대로 기술해 보면 아래와 같습니다.

① 먼저 마침표(.)를 제거한다.

② 두번째로, '빠른', '도왔다'와 같은 어절이 Komoran 을 이용해서 제거된다.

③ 명사 출현 특성을 이용하여, '자카르타', '아시안게임', '손흥민', '드리블', '슈팅', '한국', '우승' 이 명사로 추출됩니다.

④ ~ ⑥ 과정에서 제거되는 단어들이 없으므로, 최종 결과물은 아래와 같이 출력됩니다.

['손흥민', '한국', '자카르타', '드리블', '우승', '슈팅', '아시안게임']

extract() 함수 사용 예

extract() 함수의 구체적인 사용 예를 설명하기 위해 아래와 같은 신문기사가 있다라고 가정하겠습니다.

> 토트넘이 긴장해야 할 지도 모른다. 손흥민의 바이에른 뮌헨 이적설이 끊이질 않는다.
>
> 영국 유력지 '가디언'은 22 일(한국시간) "마우리시오 포체티노 감독에게 좋지 않은 소식이다. 내년 1 월 뮌헨이 손흥민 영입 추진하고 있다. 손흥민은 분데스리가 챔피언 뮌헨으로 떠날 수도 있다"라고 설명했다.
>
> 손흥민의 이적설은 이탈리아 현지에서 흘러 나왔다. 이탈리아 일간지 '칼치오메르카토'의 최초 보도 이후 영국과 일부 독일 언론에서 재해석해 이적설을 전했다. 당시 언론들은 손흥민이 최근 재계약을 체결했다는 점을 근거로 가능성을 낮게 점쳤다.
>
> '가디언'도 "손흥민이 이적설을 민감하게 반응하진 않을 것이다. 뮌헨 이적설을 토트넘과 연봉 협상에 이용할 수도 있다"라고 전했다. 그러나 '가디언'의 보도로 뮌헨 이적설이 재점화 된 점을 고려하면, 토트넘 입장에서 흘려 들을 만한 이야기는 아니다.
>
> 손흥민은 2015 년 토트넘 입단 전까지 독일 분데스리가에서 활약했다. 언어와 리그 적응에 문제가 없단 점과, 뮌헨이 유럽축구연맹(UEFA) 챔피언스리그 우승에 도전하는 점을 돌아보면 매력적인 팀이다. 2018 자카르타-팔렘방 아시안게임으로 더 이상 병역 문제도 없다. 축구공은 둥글기에 어떤 일도 일어날 수 있다.
>
> 그러나 내년 1 월 이적에는 회의적이다 . 손흥민은 잉글랜드 프리미어리그 적응 이후 2 시즌 동안 두 자리 득점에 성공했다. 리그와 챔피언스리그가 본격적으로 접어드는 시점에 토트넘이 손흥민이 내줄 가능성은 낮다.

이 신문기사의 내용이 example.txt 라는 텍스트 파일에 저장되어 있다고 하는 경우, 우리는 아래와 같은 코드를 통해서 해당 텍스트 내용을 Python 으로 불러 올 수 있습니다.

```
with open('example.txt', 'r', encoding='utf8') as f:
    news_article = f.read()
```

news_article 변수에 저장되어 있는 해당 기사의 내용을 가지고 명사 추출을 위한 extract()함수를 아래와 같이 호출할 수 있습니다.

```
extract(text)
```

위 함수 호출은 아래와 같이 61 개의 명사 후보 단어들을 추출합니다.

['UEFA', '가능', '가디언', '감독', '고려', '근거', '긴장', '내년', '도전', '독일', '득점', '리그', '마우리시오', '매력', '문제', '뮌헨', '바이에른', '반응', '병역', '보도', '분데스리가', '소식', '손흥민', '시점', '아시안게임', '언론', '언어', '연봉', '영국', '영입', '우승', '유럽축구연맹', '유력지', '이야기', '이용', '이적설', '이탈리아', '일간지', '일부', '입단', '입장', '잉글랜드', '자리', '자카르타팔렘방', '재계약', '재점화', '재해석', '적응', '챔피언스리그', '최근', '최초', '추진', '축구공', '칼치오메르카토', '토트넘', '포체티노', '프리미어리그', '한국시간', '현지', '협상', '회의']

만약, 더 자주 출현하는 명사 단어들만을 추출하고자 한다면, freq 의 값을 증가시키면 됩니다. 예를 들어,

```
extract(text, freq=2.0)
```

라고 입력하게 되면, 아래와 같이 17 개의 명사 후보 단어들이 추출됩니다.

['가능', '가디언', '내년', '독일', '리그', '문제', '뮌헨', '보도', '분데스리가', '손흥민', '언론', '영국', '이적설', '이탈리아', '적응', '챔피언스리그', '토트넘']

기본 형태소 분석기의 결과와 비교

이번에는 Twitter 를 가지고 명사 단어들을 추출해 보겠습니다.

```
print(set(twitter.nouns(text)))
```

['1 월', '2015 년', '2018', '22 일', '2 시', '가능', '가능성', '가디언', '감독', '것', '게임', '고려', '근거', '긴장', '내년', '당시', '더', '도', '도전', '독일', '동안', '두', '득점', '리그', '마', '만', '매력', '메르', '문제', '뮌헨', '바이에른', '반응', '병역', '보도', '보도로', '본격', '분데스리가', '설명', '성은', '소식', '손흥민', '수', '수도', '시간', '시오', '시점', '아시안', '언론', '언어', '연맹', '연봉', '영국', '우리', '우승', '유럽', '유력', '은', '의', '이상', '이야기', '이용', '이적', '이적설', '이탈리아', '이후', '일간', '일도', '일부', '입', '입단', '입장', '잉글랜드', '자리', '자카르타', '재', '재계약', '재해', '적응', '전', '점', '점화', '지도', '챔피언', '챔피언스리그', '체결', '최근', '최초', '추진', '축구', '축구공', '카토', '칼치', '토트넘', '티노', '팀', '팔렘방', '포체', '프리미어리그', '하진', '한국', '해', '현지', '협상', '활약', '회의']

Komoran 을 이용하여 추출된 명사 단어들은 아래와 같습니다.

['1 월', '가디언', '감독', '것', '고려', '근거', '긴장', '내년', '년', '당시', '도전', '독일', '동안', '득점', '르', '리그', '마우리시오 포체티노', '만', '매력', '문제', '뮌헨', '바이에른', '반응', '병역', '보도', '분데스리가', '설', '설명', '성공', '소식', '손흥민', '수', '시간', '시점', '시즌', '아시안게임', '언론', '언어', '연맹', '연봉', '영국', '영입', '오메', '우승', '유럽',

'유력지', '이상', '이야기', '이용', '이적', '이탈리아', '이후', '일', '일간지', '일부', '입단', '입장', '있다', '잉글랜드', '자리', '자카르타', '재계약', '적응', '전', '점', '점화', '지도', '챔피언', '챔피언스리그', '체결', '최근', '최초', '추진', '축구', '축구공', '치', '카토', '칼', '토트넘', '팀', '팔렘방', '프리미어리그', '하진', '한국', '해석', '현지', '협상', '활약', '회의']

위의 결과에서 보이는 것 처럼 본 논문에서 제시된 명사 추출기의 경우 고유명사, 외래어와 같은 미등록 단어를 더 잘 찾는 것으로 나타났습니다 (예, 마우리시오, 포체티노, UEFA 등). 하지만, '가능성' ' 당시' 등과의 일반 명사는 기존의 형태소 분석기가 더 잘 찾는 것으로 나왔습니다. 많은 실험 결과, 이러한 특성은 다른 문서에 대해서 비슷하게 작용하는 것으로 나타났습니다.

soynlp 사용하기[9]

두번째 방법은 soynlp 라는 모듈을 사용하는 것입니다.

명령 프롬프트 윈도우에서 다음 명령어 수행합니다.

```
pip install soynlp
```

명사 추출하기

soynlp 에는 명사 추출에 사용될 수 있는 여러개의 클래스가 제공됩니다. 그 중에서 여기서는 LRNounExtractor_v2 에 대해서 알아보도록 하겠습니다.

LRNounExtractor_v2 는 cohesion 이라는 방법과 branch entropy 라는 방법을 혼합하여 사용한 알고리즘입니다. cohesion 이라는 방법과 branch entropy 에 대한 구체적인 설명은 이 책의 범위를 벗어남으로 추가적인 설명을 원하는 독자는 https://github.com/lovit/soynlp 를 참조하기 바랍니다.

다음과 같이 LRNounExtractor_v2 를 사용할 수 있습니다.

```
from soynlp.noun import LRNounExtractor_v2
noun_extractor = LRNounExtractor_v2(verbose=True)
```

train_extract() 함수를 사용해서 명사 단어들을 추출해 낼 수 있습니다. train_extract() 함수는 텍스트를 구성하고 있는 문장들로 이루어진 리스트 데이터를 인자로 받습니다. 아래와 같이 코딩할 수 있습니다.

```
nouns = noun_extractor.train_extract(sents)
```

위의 기사를 가지고 직접한번 해보겠습니다. 아쉽게도 train_extract()는 전처리 기능을 자동으로 제공하지 않기 때문에, sents 형태의 데이터를 준비하기 위해서는 사용자가 직접 불필요한 기호 제거와 문장 형태로 분리를 해주어야 합니다. 이는 다음과 같은 코드를 사용할 수 있습니다.

Comment [S121]: soynlp 는 지속적으로 그 내용이 업데이트 되고 있어서, 여러분이 이 책을 읽고 있을 때는 그 내용이 달라져 있을 수도 있습니다. 따라서, 해당 모듈에 대한 코드와 설명을 제공하는 https://github.com/lovit/soynlp 을 참조하기를 권합니다.

Comment [S122]: nouns 는 {단어:namedtuple} 형식

Comment [S123]: list of strings
예) ['문장 1', '문장 2']
LRNounExtractor 는 대용량의 text 데이터를 대상으로 작성되었기 때문에 sents 에 저장된 문장의 수가 많은 경우에 보다 적합하게 실행이 됩니다.

[9] https://github.com/lovit/soynlp 참조

```
import re

text1 = re.sub(r'[^\.\?\!\s\w\d]', ' ', text)

sents = re.split(r'[\.\?\!]\s+',text1)
```

위의 sents 에 대해서 nouns = noun_extractor.train_extract(sents) 를 실행하면 아래의 결과를 얻을 수 있습니다.

```
nouns

{'가능성': NounScore(frequency=2, score=1.0),
 '리그': NounScore(frequency=2, score=1.0),
 '문제': NounScore(frequency=2, score=1.0),
 '뮌헨': NounScore(frequency=6, score=1.0),
 '보도': NounScore(frequency=2, score=1.0),
 '분데스리': NounScore(frequency=2, score=1.0),
 '손흥민': NounScore(frequency=9, score=1.0),
 '언론': NounScore(frequency=1, score=0.5),
 '영국': NounScore(frequency=2, score=1.0),
 '이적': NounScore(frequency=1, score=0.5),
 '이적설': NounScore(frequency=6, score=1.0),
 '적응': NounScore(frequency=2, score=1.0),
 '전': NounScore(frequency=3, score=1.0),
 '점': NounScore(frequency=5, score=1.0),
 '챔피언스리그': NounScore(frequency=2, score=1.0),
 '토트넘': NounScore(frequency=5, score=1.0)}
```

위에서 볼 수 있는 것 처럼, 명사 추출의 결과는 사전 데이터입니다. 각 명사 단어에 대해서 추가적인 정보들이 value 로 저장되어 있습니다.

명사 단어들만 아래와 같이 출력해 보겠습니다.

```
print(sorted(list(nouns.keys())))
```

['가능성', '리그', '문제', '뮌헨', '보도', '분데스리', '손흥민', '언론', '영국', '이적', '이적설', '적응', '전', '점', '챔피언스리그', '토트넘']

명사 추출기와 형태소 분석기를 같이 사용하는 방법

위의 방법을 사용하더라도 문서에서 사용된 명사 단어 모두를 정확하게 추출하는 것은 어렵습니다. 명사 단어들을 사용해서 텍스트를 분석하고자 하는 경우에는 위에서 제시된 명사 추출기만을 사용하기 보다는 앞에서 언급된 것

Comment [S124]: text 에는 원본 기사 내용이 저장되어 있습니다.

Comment [S125]: 이러한 추가적인 정보 (즉, frequency 와 score)를 이용해서 추가적으로 단어를 다시 한번 선택할 수 있습니다.

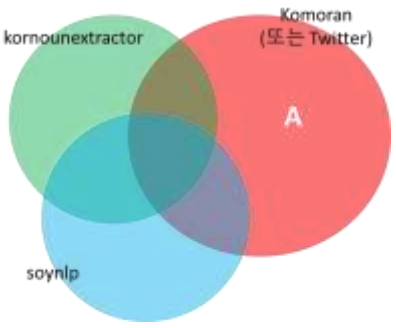
처럼 기존의 다른 형태소 분석기 등을 같이 사용하는 것이 좋습니다. 아래 방법을 사용해서 명사 추출기를 통해 추출이 된 명사를 형태소 분석기들이 사용하는 사전에 추가할 수 있습니다.

① ckonlpy Twitter 의 add_dictionary() 사용해서 명사 단어 추가하기

② Komoran() 에서 사용할 수 있는 dic.txt 파일에 해당 단어들 추가하기

여전히 문제점 존재

명사 추출기를 사용해서 명사를 등록하더라도 여전히 문제가 발생합니다. 이를 위해 아래 그림을 보도록 하겠습니다.



하지만, 그렇게 하면 명사로 태깅이 안되던 단어들은 태깅이 되지만, 태깅이 잘못되었던 애들은 그대로 잘 안된채로 남아 있게 됩니다.

위의 그림에서 A 부분에 있는 단어들은 kornounextractor 또는 soynlp 에서는 명사로 구분되지 않았지만, konlpy 자체 형태소 분석기인 Komoran (또는 Twitter)에 의해서 명사로 구분된 단어들 입니다. 우리는 이러한 단어들도 추가적으로 명사의 단어들로 사용해야 합니다. 하지만, A 의 영역에는 명사로 태깅이 잘 되지 않은 단어들 (혹은 명사인지 아닌지 구분이 잘 안되는 단어들)이 존재합니다. 예를 들어, '설', '만', '치' 등의 한 음절 단어들은 어떠한 단어들을 의미하는지 (즉, 명사인지 구분이) 명확하지 않다. 따라서 이러한 단어들은 명사로 구분하지 않아야 합니다. 다음과 같은 코드를 사용해서 한 음절의 단어들은 제거할 수 있습니다.

```
komo_nouns = komoran.nouns(text)
for word in set(komo_nouns):
  if len(word) == 1:
    while word in komo_nouns:
      komo_nouns.remove(word)
```

이러한 한 음절의 단어들 이외에도 명사로 잘못 태깅된 단어들 (예, '오메', '있다')이 존재할 수 있는데, 이러한 단어들은 최종분석에서 제거하고 사용해야 합니다.

특정 품사의 단어들 선택하기

이렇게 각 단어에 대해서 품사 태깅을 한 다음에 우리가 해야하는 작업은 영어와 마찬가지로 특정 품사의 단어들만 선택하는 경우입니다. 한글의 경우에는 형태소 분석기 마다 품사의 이름이 조금씩 차이가 있습니다. 각 형태소 분석기에서 사용되는 품사의 이름에 대한 정보는 https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0 에서 볼 수 있습니다.

본 글에서는 Twitter() 분석기를 통해서 나온 결과를 이후 분석에 이용하도록 하겠습니다. 분석을 위해서 명사만을 선택하고자 하는 경우 아래와 같이 할 수 있습니다.

```
Noun_words = []
for word, pos in twitter_nouns:
    if pos == 'Noun':
        Noun_words.append(word)
```

위의 코드를 실행하면 Noun_words 라는 list 변수에 명사 단어들만 저장되게 됩니다.

명사만을 추출하고자 하는 경우에는 Komoran, Twitter 에서 제공되는 nouns() 함수를 사용하면, 명사 추출을 더 쉽게 작업할 수 있습니다. 아래와 같이 사용합니다.

komoran.nouns(text)

twitter.nouns(text)

③ 불용어 제거

불용어는 분석에 있어 별 의미를 갖지 않는 단어들을 말합니다. 위의 예에서는 '당시', '최근' 등이 불용어로 간주될 수 있습니다. 이러한 단어들은 분석의 목적과 상관없는 정보들이기 때문에 이러한 단어들이 분석에 들어가면 결과가 정확하지 않을 수도 있습니다. 따라서, 가능한 분석 전에 제거를 해주는 것이 좋습니다. 한글 같은 경우에는 영어와 달리 기본적으로 제공되는 불용어 사전이 별도로 없습니다. 사용자가 직접 불용어 사전을 만들어야 합니다. 불용어 사전은 텍스트 파일로 만들수도 있고, 임시로 사용할 수 있는 리스트 변수의 형태로 만들 수도 있습니다. 텍스트 파일을 사용하면, 체계적으로 관리하는 것이 가능함으로 텍스트 파일로 불용어 사전을 구축하는 것을 권장합니다.

여기서는 간단하게 아래와 같이 리스트 형태의 불용어 사전을 만들도록 하겠습니다.

stopwords = ['연합뉴스', '기자', '최근', '당시']

이 변수를 사용해서 아래와 같이 해당 단어들을 text 에서 제거할 수가 있습니다.

```
unique_Noun_words = set(Noun_words)
for word in unique_Noun_words:
   if word in stopwords:
      while word in Noun_words: Noun_words.remove(word)
```

이렇게 하면 Noun_words 에 불용어가 제거된 명사단어들만 저장되게 됩니다.

영어의 경우와 마찬가지로 필요하다면 이러한 전처리 과정을 반복하게 됩니다.

한글 음절의 초·중·종성 분리하기

한글 텍스트 분석을 하는 경우, 하나의 음절을 음소의 단위로 구분해야 하는 경우도 있습니다. 한글 음절의초·중·종성 분리하기 위해서 사용할 수 있는 모듈에는 여러개가 존재합니다. 비슷한 기능을 제공함으로 여기서는 hgtk 모듈을 사용하겠습니다.

아래 코드를 사용하여 설치합니다.

pip install hgtk

아래와 같이 사용할 수 있습니다.

import hgtk

한음절

hgtk.letter.decompose('값')

('ㄱ', 'ㅏ', 'ㅄ')

두음절 이상

hgtk.text.decompose('손흥민')

'ㅅㅗㄴᴥㅎㅡㅇᴥㅁㅣㄴᴥ'