Text preprocessing (텍스트 전처리)

이상엽

텍스트 전처리란?

수집한 텍스트 데이터 (e.g., 문서의 집합)를 최종 분석에 적합한 형태로 만드는 과정으로, 전처리의 결과물은 우리가 최종 분석에 사용하고자 하는 텍스트 정보만을 추출, 저장한 것입니다. 많은 경우에 불용어가 제거된 특정 품사 (예, 명사)의 단어들만을 최종 분석에 사용하게 됩니다.

주요 과정

전처리 단계에서는 주로 다음과 같은 작업들을 수행합니다.

① 불필요한 기호 / 표현 없애기(예, !, ., “, ; 등)
② 대소문자 변환 (Case conversion, 소문자 ↔ 대문자) (영어의 경우에만 해당)
③ 단어 (혹은 Token) 단위로 짤라주기 (Tokenization)
④ 단어의 품사 찾기 (Part of Speech tagging)
④ 원하는 품사의 단어들만 선택 [1]
⑤ 단어의 원형(혹은 어근) 찾기(Lemmatization / Stemming)
⑥ 불용어 (Stopwords) 제거

필요에 따라서는 위 과정의 순서가 바뀔수도 있고, 같은 과정을 한번이상 수행할 수도 있습니다.

각 과정에 대해서 좀 더 자세하게 살펴보도록 하겠습니다.

1) 불필요한 기호 / 표현 없애기(예, !, ., “, ; 등)

웹으로부터 수집한 텍스트에는 최종 분석에 필요하지 않은 다양한 형태의 marks 와 symbols 들이 포함되어 있습니다. 텍스트 전처리의 가장 기본적인 작업은 이러한 불필요한 mark 와 symbol 들을 없애주는 것입니다. 이러한 작업을 하기 전에 결정해야 하는 중요한 것 중하나가 ‘어떠한 mark 나 symbol 들을 이 단계에서 제거할 것인가?’ 하는 것입니다. 처음에는 불필요하다고 생각되었던 기호들도 나중에는 필요하게 될 수도 있습니다. 대표적인 예가 문장의 끝을 나타내는 기호들 (예, . (마침표), ? (물음표), ! (느낌표) 등) 입니다. 특히

[1] 많은 경우 명사의 단어들만을 사용해서 분석을 하게 됩니다.

텍스트 분석이 문장 단위로 이루어지는 경우 (예, semantic network analysis 등)에는 이 단계에서 문장의 끝을 나타내는 기호들을 없애지 않는 것이 중요합니다.

불필요한 기호를 제거하기 위해서 사용할 수 있는 가장 간단한 방법은 Python 에서 제공되는 string 함수 중 하나인 replace()를 사용하는 것입니다. 다음과 같이 코드를 작성할 수 있습니다.

```
text = "Today is 'Thursday,' it is going to be a nice day."
cleaned_text = text.replace("'",'').replace(',','').replace('.', '')
```

위의 코드는 text 변수에 담긴 기사 내용에서 작은 따옴표 ('), 마침표(.), 쉼표(,)를 제거하는 코드입니다. 만약 추가로 제거하고 싶은 문자가 있다면 replace() 함수를 추가적으로 사용하면 됩니다.

불필요한 기호를 없애는 좀 더 효율적인 방법은 정규식을 사용하는 것입니다. 다음과 같은 코드를 사용할 수 있습니다.

```
import re
text1 = "Today is 'Thursday,' it is going to be a nice day. I am happy! @@ Are you?"
cleaned_text = re.sub(r'[^\.\?\!\w\d\s]','',text1)
cleaned_text
```

주피터 노트북에서 위의 코드를 실행하면 아래와 같은 결과가 나옵니다.

'Today is Thursday it is going to be a nice day. I am happy! Are you?'

2) 대소문자 변환 (Case conversion, 소문자 ↔ 대문자) (영어의 경우에만 해당)

대소문자의 변환은 영어 텍스트에만 해당 합니다. 영어 같은 경우는 같은 단어라도 단어가 사용되는 위치에 따라서 그 형태가 다를 수 있습니다. 이러한 경우, 컴퓨터는 형태가 다른 두 단어를 서로 다른 단어로 인식을 하게 됩니다. 예를 들어, this 라는 단어는 문장 중간에 사용될 때는 this 로 사용되는 반면, 문장 처음에서 사용되게 되면 This 로 사용됩니다. 그러면 컴퓨터는 this 와 This 를 서로 다른 단어로 인식하게 됩니다. 이러한 문제점을 방지하기 위해서 영어 텍스트 분석의 경우에는 전처리 과정에서 많은 경우에 전제 텍스트 데이터를 소문자나 대문자로 변환을 해 줍니다.

Python 에서는 string 함수인 lower() (대문자 -> 소문자) 혹은 upper() (소문자 -> 대문자)를 사용해서 case conversion 을 수행하게 됩니다. 아래와 같이 할 수 있습니다 .

```
cleaned_text = cleaned_text.lower()
```

Comment [S11]: Regular expression 에 대한 tutorial 을 참고하시기 바랍니다.

Comment [S12]: 대체된 결과가 cleaned_text 라는 새로운 변수에 저장됩니다.

Comment [S13]: re.sub() 함수는 3 개의 파라미터를 입력 받습니다. 첫번째 파라미터는 대체하고자 하는 문자(열)이고, 두번째 파라미터는 대체하는데 사용하고자 하는 문자(열), 그리고 세번째 파라미터는 대체하고자 하는 문자(열)이 저장된 텍스트 데이터 입니다.

Comment [S14]: 첫번째 파라미터는 대체하고자 하는 기존에 있는 문자(열)이 들어가는데 정규식에서는 문자열의 패턴을 입력할 수 있습니다. [^\.\?\!\w\d\s] 가 의미하는 것은 [] 괄호 안에 있는 ^를 제외한 다른 문자가 아닌 모든 문자를 의미합니다. ^는 부정 (negation) 기능을 제공하는 메타문자 입니다. ., ?, ! 등은 정규식에서 특수한 기능을 하는 메타문자로 사용되기 때문에 해당 기호들을 의미하기 위해서는 앞에 \를 붙여 주어야 합니다. 즉, [^\.\?\!\w\d\s]는 ., ?, ! 등이 아닌 기호와 문자 (\w), 숫자 (\d), 공백문자 (\s) 가 아닌 모든 character 를 의미합니다 (예, ‘, “, @, # 등)

영어 데이터에 대해서 case conversion 을 할 때, 중요하게 고려해야 하는 것이 그 순서입니다. 좀 더 구체적으로 말하면, 불필요한 기호나 표현을 제거한 다음에 하는 것이 좋은지 아니면 품사 tagging 을 한다음에 하는 것이 좋은지를 결정해야 합니다. 왜냐하면, 영어 같은 경우는 고유 명사인 경우에 단어 전체가 대문자로 되어 있던지 아니면 첫 글자가 대문자로 되어 있습니다. 그리고 이러한 정보는 해당 단어의 품사를 찾을 때 사용됩니다. 즉, 품사 tagger 가 특정한 단어의 품사를 찾을 때 첫글자가 대문자 이거나 단어에 사용된 모든 문자가 대문자인 경우에는 해당 단어를 (고유) 명사로 tagging 할 확률이 높습니다. 하지만, 이 단어를 품사 tagging 을 하기 전에 소문자로 변환해 버리면, 해당 단어를 (고유) 명사로 tagging 할 확률이 낮아집니다. 특히나, 해당 단어에 대한 다른 단어가 존재하는 경우에는 더욱 그렇습니다. 예를 들어서 성 (last name)이 White 라는 사람이 있고, 그 사람에 대한 기사 데이터가 있다라고 가정을 합시다. 이러한 경우에 White 라는 단어의 품사는 고유 명사입니다. 그리고 소문자로 변환을 하지 않으면 해당 단어는 (고유) 명사로 tagging 될 확률이 높습니다. 하지만, tagging 을 하기 전에 텍스트를 모두 소문자로 변화하게 되면 White 는 white 로 변환되게 됩니다. white 라는 단어가는 "흰, 흰색의" 라는 뜻을 갖는 형용사입니다. 즉 White 가 white 로 변환이 된 경우에는 white 로 인식되어 형용사로 tagging 이 될 것입니다. 이는 우리가 원하는 결과가 아니고, 이러한 품사 tagging 결과를 가지고 최종 분석에 사용한다면 그 분석 결과의 신뢰도는 낮아질 것입니다.

그러면 여러분들이 생각하기에는 품사 tagging 을 한 다음에 case conversion 을 하면 되지 않느냐라고 생각할 것입니다. 하지만, 반대로 생각해 보면 명사가 아닌 영어 단어들 같은 경우에 문장의 첫 단어로 사용되게 되면 첫 글자가 대문자로 표기 됩니다. 그러면 이러한 단어는 고유명사가 아님에도 불구하고, 품사 tagging 시 고유명사로 tagging 이 될 확률이 있습니다.

이러한 문제에 대한 하나의 해결방안은 고유명사에 대한 사전을 사용자가 만드는 것입니다. 사전을 만들고, 사전에 존재하는 고유명사를 별도로 처리하는 것입니다. 하지만, 텍스트 데이터에 존재하는 모든 고유 명사를 미리 아는 것이 어렵다는 문제가 있습니다.

3) 단어 (혹은 Token) 단위로 잘라주기 (Tokenization)

하나의 문서를 구성하고 있는 단어들이 무엇인지 파악하기 위해서는 문서 혹은 문장을 단어 (token) 단위로 분리해서 표현할 필요가 있습니다. 이러한 과정을 tokenization 이라고 합니다. 보통 텍스트 분석의 기본 단위는 단어이기 때문에 우리가 갖고 있는 텍스트 데이터를 단어로 분리하여 저장한 다음, 그러한 단어 단위로 이루어진 텍스트를 가지고 분석을 하게 됩니다. 하나의 토큰(token)은 뜻을 갖고 사용될 수 있는 가장 작은 글의 단위라고 정의될 수 있으며 (Sarkar, 2016, p.108), 보통은 하나의 단어라고 생각하면 됩니다.

Comment [S15]: /토크나이제이션/ 이라고 발음합니다.

Comment [S16]: 한글은 단어가 아니라 형태소가 의미를 갖는 작은 단위로 정의됩니다.

가장 기본적인 tokenization 방법은 영어 같은 경우에는 띄어쓰기를 기준으로 문장을 쪼개 주는 것입니다. 영어의 경우에는 하나의 문장이 띄어쓰기를 기준으로 단어들의 조합으로 구성되어 있어 띄어쓰기를 가지고 tokenization 을 해도 단어 단위의 데이터를 추출할 수 있습니다. 이를 위해서 영어 같은 경우에는 다음과 같이 split() 이라는 string 함수를 사용할 수 있습니다.

```
tokens = cleaned_text.split()
```

한글의 경우에는 문장의 구성단위가 단어가 아니라 어절입니다. 하나의 어절은 여러 단어의 조합으로 이루어져 있습니다 (예, 명사+조사).[2] 한글의 tokenization 은 형태소 분석이라는 방법을 통해서 이루어 지는데 이는 별도의 section 에서 좀 더 구체적으로 설명하도록 하겠습니다.

Comment [S17]: nltk 모듈에서 제공되는 word_tokenize()라는 함수도 비슷한 기능을 제공합니다.

Comment [S18]: 한글에서 형태소는 뜻을 가진 작은 단위로 정의됩니다.

4) 단어의 원형(혹은 어근) 찾기(Lemmatization / Stemming)

Tokenization 을 수행하고 난 이후에 우리는 단어 단위의 데이터를 갖게 됩니다. 그 다음으로 해야 하는 작업은 각 단어의 원형 (혹은 어근)을 찾는 것입니다. 하나의 단어 (명사, 형용사, 동사 등)는 그 시제와 인칭, 혹은 단·복수에 따라 그 형태가 달라집니다. 예를 들어, 영어 단어인 eat 같은 경우에는 원형은 eat 이지만, 3 인칭인 경우에는 eats 로 과거인 경우에는 ate 로 과거 분사인 경우에는 eaten 으로 사용되는 것 처럼요. 한글의 '먹다'라는 단어도 시제에 따라 먹었다, 먹는다, 먹을 것이다 등의 다양한 형태로 사용될 수 있습니다. case conversion 을 설명하면서 언급한 것 처럼, 같은 뜻을 갖는 단어라고 할지라도 그 형태가 다른 경우에는 컴퓨터는 서로 다른 단어로 인식하게 됩니다. 보통의 텍스트 분석에서는 같은 단어가 어떻게 다른 형태로 사용되었느냐 보다는 어떤 단어가 사용되었느냐가 더 중요하기 때문에, 같은 단어들 같은 경우에는 별도의 과정을 거쳐서 동일한 형태로 변환을 한 후에 텍스트 분석을 진행하는 것이 더 바람직합니다. 이러한 목적에 사용될 수 있는 방법이 여러가지 있을 수 있는데, 대표적인 방법이 해당 단어의 원형 (lemma)이나 어근 (stem)의 형태로 단어를 변환하는 것입니다. 전자를 lemmatization (원형찾기), 후자를 stemming (어근찾기)이라고 합니다.[3] 영어의 경우에는 tokenization, lemmatization (혹은 stemming), 그리고 아래에서 다루는 PoS tagging (품사표기)의 과정을 별도로 수행해야 하지만, 한글의 경우에는 형태소 분석에서 3 개의 과정이 한번에 이뤄지게 됩니다. 영어의 경우에는 Python 에서 제공되는 nltk 라는 모듈을 사용해서 해당 작업을 수행할 수 있습니다.

Comment [S19]: 보통은 둘 중에 한가지 방법만을 사용하면 됩니다. 그럼 우리는 둘중 어떤 방법을 사용할 것인지를 결정해야 합니다. 선택되어져야 하는 방법은 분석의 목적이나 분석에 사용되는 텍스트 데이터의 특성에 따라서 달라집니다. 하지만, 저는 lemmatization 방법을 많은 경우에 추천합니다. 주된 이유는 stemming 은 많은 경우에 어근을 제외한 단어의 나머지 부분을 잘라내기 때문에 stemming 의 결과로 남아 있는 단어가 의미하는 것이 무엇인지 모르는 경우가 많습니다. 예를 들어 saw 라는 단어가 있다라고 가정하겠습니다. stemming 방법을 적용하면 s 라는 하나의 문자만 return 되는 반면에 lemmatization 방법을 사용하면 문맥에 따라서 see 또는 saw (명사, 톱)이라는 단어가 return 됩니다.

[2] https://blog.naver.com/unjangkorean/220814019022 참조

[3] 둘의 차이에 대한 보다 자세한 정보는 https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html 를 참조하세요.

5) 단어의 품사 찾기 (Part of Speech tagging)

하나의 단어는 저 마다의 품사가 있습니다. 명사, 형용사, 동사 등이 그러한 품사의 예입니다. 텍스트 분석에서는 텍스트에서 사용된 각 단어의 품사를 아는 것이 중요한데 이는 많은 경우에 최종 분석에는 특정 품사 (예, 명사, 형용사, 동사 등)의 단어들만을 사용하기 때문입니다. 영어 텍스트의 분석은 텍스트 데이터를 단어 단위로 tokenization 한 이후에 각 단어의 품사를 찾는 과정을 별도로 수행하지만, 위에서 언급한 것 처럼 한글 텍스트 경우에는 형태소 분석을 통해서 각 단어 (즉, 형태소)의 품사를 찾게 된다.

Comment [S110]: 특정한 품사의 단어들만 사용하는 이유는 그러한 품사들이 텍스트의 주된 특성과 의미를 포함하고 있기 때문입니다. 예를 들어, 한글에서는 조사 단어들 (예, 이, 가, 으로, 을 등)은 별 뜻을 갖고 있지 않습니다.

Comment [S111]: 이러한 역할을 하는 툴을 형태소 분석기라고 합니다. Komoran, KKma, Twitter 등의 다양한 형태소 분석기가 있습니다.

Comment [S112]: 한글 분석에 사용되는 형태소 분석기에서 다루는 기본 단위는 형태소 입니다. 하지만, 형태소와 단어가 큰 차이가 없기 때문에 보통은 단어라고 표현하기도 합니다.

Python 에서의 영어와 한글의 품사 tagging

영어의 경우 Python 에서는 nltk 모듈에서 제공하는 pos_tag()[4]라는 함수를 사용해서 단어의 품사를 찾습니다.[5] nltk 에서도 여러 종류의 pos tagger 를 제공하는데 기본적으로는 Maxent treebank pos tagging model 방법을 사용합니다. 단어들의 품사를 찾는 것도 일종의 classification (분류) 작업입니다. 따라서 많은 pos tagger 들이 지도학습 방법 (supervised machine learning) 을 사용해서 tagging 을 합니다.

한글의 경우는 Python 에서 제공되는 KoNLPy[6] 라는 모듈을 사용해서 단어에 대한 (정확하게는 형태소) 품사를 찾습니다. 좀 더 구체적으로 말하자면, KoNLPy 에서 제공되는 여러 종류의 형태소 분석기를 사용해서 단어의 품사를 찾게 됩니다. 자주 사용되는 형태소 분석기로는 Twitter, Komoran 등이 있습니다.[7] 한글 형태소 분석기는 많은 경우 고유한 사전을 가지고 단어의 품사를 찾습니다.

영어 분석에 사용되는 nltk 모듈에서 제공되는 pos tagger 들이나 한글 분석에 사용되는 KoNLPy 에서 제공되는 형태소 분석기 모두 완벽하지는 않습니다. Tagging 에 사용되는 학습 데이터나 사전이 완벽할 수 없기 때문에 그렇습니다. 학습 데이터 같은 경우는 해당 데이터에서 사용되는 단어들이나 표현이 최신 것이 아니거나 그리고 분석 대상이 되는 데이터와 다른 분야의 데이터라면 분석에 사용되는 단어들의 품사를 정확하게 찾을 수 없습니다. 사전을 사용하는 경우에는 사전에 포함되어 있는 단어들이 최신 단어들이 아닌 경우에 그러한 사전을 사용해서는 단어의 품사를 제대로 찾을 수가 없는 것입니다.

Comment [S113]: 특히, 분석하고자 하는 텍스트 데이터에 최신 용어 혹은 인터넷 용어 등이 많이 포함되어 있다면 더욱 그렇습니다.
이러한 문제를 해결하기 위해서는 다음과 같은 방법을 사용할 수 있습니다.
1) 품사 tagging 에 사용되는 사전의 update (즉, tagging 이 잘 안되는 단어 추가 등)
2) coding 을 통해서 해당 문제 해결 (하지만 이 방법은 tagging 이 잘 안되는 단어가 많은 경우에는 노력이 많이 필요하다는 단점이 있습니다.)
3) 다른 개발자가 이미 개발해 놓은 모듈 혹은 함수 사용하기 (이를 위해서는 구글 혹은 github 을 잘 찾아봐야 합니다.)

원하는 품사의 단어들만 선택

품사 태깅을 통해서 각 단어의 품사를 찾은 이후에 수행해야하는 과정은 분석에 필요한 품사의 단어들을 선택해 저장하는 것입니다. 많은 경우에 명사, 형용사, 동사 등의 주요 품사만을 선택해서 최종 분석을 하게

[4] nltk 의 pos_tag() 함수의 작동원리에 대한 좀 더 자세한 내용은 http://textminingonline.com/dive-into-nltk-part-iii-part-of-speech-tagging-and-pos-tagger 을 참조하세요.
[5] 보다 자세한 내용은 http://www.nltk.org/book/ch05.html 참조하세요.
[6] http://konlpy.org/en/v0.4.4/
[7] http://konlpy.org/en/v0.4.4/api/konlpy.tag/

됩니다. 이유는 이러한 주요 품사의 단어들이 해당 문서의 특성을 더 잘 나타내기 때문에, 해당 품사의 단어들만을 사용하게 되면 우리가 원하는 결과를 얻는데 더 효과적일 수 있기 때문입니다. 뿐만 아니라, 분석해야하는 단어의 수를 줄이므로써 분석을 좀 더 빠르게 진행할 수 있다는 장점도 있습니다.

6) 불용어 (Stopwords) 제거

불용어라 함은 텍스트 분석에서 의미가 없는 단어를 의미합니다. 예를 들어 영어의 경우는 a, the 와 같은 관사나 this, that 과 같은 지시 대명사가 여기에 포함됩니다. 그 외에도 문맥이나 연구 목적에 따라서 별로 중요하지 않은 단어들이 불용어로 간주될 수 있습니다. 예를 들어, 기사를 분석하는 경우에는 신문사의 이름이나 기자의 이름 혹은 '기자', '앵커', '뉴스' 등의 단어들은 불용어로 간주될 수 있습니다. Python 으로 텍스트 분석을 하는 경우, 영어 같은 경우는 nltk 에서 기본적인 불용어 사전을 제공하지만, 한글에 대한 불용어 사전은 제공되지 않습니다. nltk 에서 제공되는 불용어 사전도 포함된 단어의 수가 괴장히 제한적입니다. 그렇기 때문에 많은 경우에 불용어를 제거하기 위해서는 사용자가 별도의 불용어 사전을 만들어 사용하게 됩니다. 불용어 사전은 간단하게는 다음과 같은 list 변수로 만들 수도 있고, 지속적인 사용과 업데이트를 위해서 파일로 만들 수도 있습니다.

```
stopwords_list = ['기자','신문','앵커']
```

● 한글과 영어 텍스트 전처리의 비교

위에서 설명한 6 가지 전처리 단계들에 대해서 한글과 영어 텍스트를 비교하면 아래와 같습니다. 영어 텍스트는 보통 nltk 라는 모듈을 통해서 각 단계를 별도로 수행하지만, 한글은 KoNLPy 에서 제공되는 형태소 분석기를 통해서 tokenization, lemmatization, 품사 찾기 (pos tagging) 작업을 한번에 수행합니다.

Comment [S114]: 각 형태소 분석기에서 제공되는 pos()라는 함수를 사용하면 됩니다.

Steps	영어	한글
Text cleaning (remove symbols/marks)	O	O
Case conversion	O	X
Tokenization	O	O
Removing stopwords (불용어)	O	O
Lemmatization (or Stemming)	O	O
POS tagging	O	O

한글과 영어 텍스트 전처리의 좀 더 구체적인 내용은 별도 튜토리얼을 참고하기 바랍니다.