English Text preprocessing

이상엽

이 문서에서는 Python 을 이용해서 영어 텍스트를 어떻게 전처리할 수 있는지에 대해서 살펴보도록 하겠습니다. Python 에서는 기본적으로 nltk 모듈을 사용해서 영어 텍스트를 전처리 합니다. nltk 모듈을 설치 후[1] 다음과 같은 명령어를 실행시켜야지만 nltk 를 사용하는데 필요한 모든 데이터를 다운로드 받을 수 있습니다.

```
import nltk
nltk.download('all')
```

본 문서는별도로 제공되는 파이썬 코드 파일인 "English_preprocessing_ver1.py" 에 있는 내용을 토대로 전처리 과정을 설명합니다. 본 코드에서는 Web scraping 과정을 통해서 pickle 형태로 저장된 New York Times 의 기사에 대해서 전처리를 수행합니다.[2] 해당 pickle 파일(En_text.p)은 dictionary 형태이기 때문에 해당 기사의 내용을 아래와 같이 Python file 로 읽어옵니다.

```
import pickle
article_dict = pickle.load(open('En_text.p', 'rb'))
title = article_dict[1]['title'] # 기사의 제목을 가져옵니다.
content = article_dict[1]['content'] # 기사 내용을 가지고 옵니다.
```

우리는 여기서 content 에 담긴 해당 기사 텍스트에 대한 전처리를 수행하도록 합니다. 다른 문서[3]에서 설명한 것 처럼 다음과 같은 과정을 거쳐서 영어 텍스트의 전처리를 수행합니다. [4]

- 불필요한 기호 / 표현 없애기(예, !, ., ", ; 등)
- 대소문자 변환 (Case conversion, 소문자 ↔ 대문자) (영어의 경우에만 해당)
- 단어 (혹은 Token) 단위로 잘라주기 (Tokenization)
- 단어의 품사 찾기 (Part of Speech tagging)
- 원하는 품사의 단어들만 선택
- 단어의 원형(혹은 어근) 찾기(Lemmatization / Stemming)
- 불용어 (Stopwords) 제거

[1] http://www.nltk.org/install.html 참조

[2] 기사의 내용은 https://www.nytimes.com/2017/06/12/well/live/having-friends-is-good-for-you.html 참조

[3] 2. Text preprocessing_overall_ver1.0.pdf

[4] 필요하다면 이러한 과정의 순서가 바뀔수도 있고, 같은 과정을 한번 이상 수행할 수도 있다.

① 불필요한 기호 / 표현 없애기

기본적으로 string 함수 중 하나인 replace()를 여러번 사용해서 불필요한 기호와 표현을 제거합니다. 만약, 문장 단위로 분석을 하게 된다면 문장을 구분하는데 사용되는 기호는 이 과정에서 제거하지 말아야 합니다.

```
filtered_content = content.replace('!', '').replace(',','').replace('.','').replace('“','')
filtered_content = filtered_content.replace('”','').replace('\n','').replace('’','')
filtered_content = filtered_content.replace('The New York Times', '')
```

② 대소문자 변환 (Case conversion, 소문자 ↔ 대문자)

여기서는 아래와 같이 lower() 함수를 이용해서 대문자를 소문자로 변환합니다.

```
lower_case_content = filtered_content.lower()
```

③ 단어 (혹은 Token) 단위로 잘라주기 (Tokenization)

Tokenization 을 위해서는 nltk 의 word_tokenize()라는 함수를 사용합니다 word_tokenize()는 파라미터로 입력받은 text 를 word 단위로 tokenize 합니다.[5] 결과를 확인해 보면 string 함수인 split()의 결과와 거의 유사한 것을 알 수 있습니다.

```
import nltk
word_tokens = nltk.word_tokenize(filtered_content)
```

④단어의 품사 찾기 (Part of Speech tagging)

tokenization 의 결과로 얻어진 단어에 대한 품사를 찾기 (PoS tagging) 위해서는 nltk 에서 제공되는 pos_tag() 라는 함수를 사용합니다. 해당 함수의 파라미터로는 tokenized 된 결과, 즉 단어의 리스트(word_tokens)를 입력합니다.

```
tokens_pos = nltk.pos_tag(word_tokens)
print(tokens_pos)
```

pos_tag() 함수의 결과를 출력해 보면, [('hurray', 'NN'), ('for', 'IN'), ('the', 'DT'), ('hotblack', 'NN'), ('coffee', 'NN'), …] 의 내용이 화면에 출력되는 것을 확인할 수 있습니다. 보는 것 처럼 tokens_pos 는 리스트 형태의 데이터로 각

Comment [S11]: 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 한 다음에 하는 것이 좋은지를 결정해야 합니다.
2. Text preprocessing_overall_ver1.0.pdf 파일에서 언급한 것 처럼, 고유명사 같은 경우는 소문자로 변경시 제대로 품사 tagging 이 안될 수 있기 때문입니다. 반대로 고유명사가 아닌 단어들이 문장의 첫단어로 사용되어 첫 character 가 대문자로 사용되는 경우는 고유명사로 tagging 이 될 수도 있습니다.

[5] sentence 단위로 tokenize 하기 위해서는 sentence_tokenize() 함수를 사용합니다.

element 가 (단어, 품사)로 구성된 하나의 tuple 입니다. nltk 에서 제공되는 각 PoS tag 의 의미는 https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html 에서 확인할 수 있습니다.

⑤ 원하는 품사의 단어들만 선택

다양한 품사들 중에서 우리는 최종 분석에 사용될 품사의 단어들만을 선택하게 됩니다. 주요 의미를 포함하고 있는 명사, 형용사 등의 품사의 단어들이 주로 선택이 됩니다. 여기서는 명사를 선택하는 경우를 설명하겠습니다. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html 에서 명사에 해당하는 tag 를 살펴보면, NN (단수 명사 혹은 셀수 없는 명사), NNS (복수 명사), NNP (단수 고유명사), NNPS (복수 고유명사)의 4 개가 있는 것을 확인할 수 있습니다. 공통적으로 모두 NN 이라는 characters 가 들어갑니다. 이러한 특성을 이용해서 명사 단어들만 아래와 같이 추출할 수 있습니다.

```
NN_words = []
for word, pos in tokens_pos:
   if 'NN' in pos:
      NN_words.append(word)
```

이렇게 하면 NN_words 라는 리스트 변수에 명사 단어들이 저장되게 됩니다.

⑥ 단어의 원형(혹은 어근) 찾기(Lemmatization / Stemming)

다음 단계는 추출된 명사 단어들의 원형을 찾는 것(Lemmatization)입니다. 명사 단어들 같은 경우에는 해당 단어가 단수냐 복수냐에 따라서 그 형태가 달라집니다. 그리고, 다른 형태의 단어는 비록 동일한 단어라고 할지라도 컴퓨터가 다르게 인식합니다. 이러한 문제점을 방지하기 위해서 우리는 단어의 원형을 사용합니다. Lemmatization 을 위해서는 nltk 에서 제공되는 WordNetLemmatizer() 함수를 사용합니다. 이 함수는 WordNet 이라는 단어 사전을 사용해서 단어들의 원형을 찾습니다.

```
wlem = nltk.WordNetLemmatizer()
lemmatized_words = []
for word in NN_words:
   new_word = wlem.lemmatize(word)
   lemmatized_words.append(new_word)
```

lemmatize() 함수는 원래 두개의 파라미터를 받습니다. 원형을 찾고자 하는 단어가 첫번째 파라미터이고 그 단어의 품사가 두번째 파라미터입니다. 두번째 파라미터가 제공되지 않으면 기본값으로 명사 품사를 사용하게 됩니다. 이번 예에서는 명사의 단어들만을 가지고 lemmatization 을 하기 때문에 두번째 품사를

제공할 필요가 없지만, 다른 품사 단어들의 원형을 찾기 위해서는 해당 단어의 품사를 두번째 파라미터로 제공해야 합니다.[6]

⑦ 불용어 (Stopwords) 제거

마지막으로 해야 하는 작업은 선택되어진 명사단어들 중에서 불용어 (즉, 최종 분석에 있어서 별 의미가 없는 단어들)를 제거하는 것입니다. 영어 같은 경우에는 nltk 에서 기본적으로 불용어 사전을 제공합니다. 그래서 영어에서 불용어를 제거하는 경우에는 1 차적으로 nltk 에서 제공되는 불용어 사전을 사용하고, 추가적으로 사용자가 별도의 불용어 사전을 생성하여 사용하게 됩니다.

nltk 에서 제공되는 불용어 사전은 아래와 같이 사용할 수 있습니다.

```
from nltk.corpus import stopwords

stopwords_list = stopwords.words('english')
```

stopwords_list 에 포함된 단어들 (불용어)을 우리가 갖고 있는 단어리스트에서 아래와 같이 제거할 수 있습니다.

```
unique_NN_words = set(lemmatized_words)
final_NN_words = lemmatized_words

for word in unique_NN_words:
   if word in stopwords_list:
      while word in final_NN_words: final_NN_words.remove(word)
```

nltk 에서 제공되는 불용어 사전을 가지고 불용어를 1 차적으로 제거한 다음에도 아직 별 의미없는 단어들이 우리의 데이터에 존재할 것입니다. 이러한 추가적인 불용어를 제거하기 위해서는 사용자가 직접 자신만의 불용어 사전을 만들어야 합니다. 이러한 불용어 사전은 간단하게는 아래와 같이 list 변수를 가지고 만들 수도 있고, 파일로 만들어서 반복해서 사용할 수도 있을 것입니다.

```
customized_stopwords = ['be', 'today', 'yesterday', "it's", "don't"]
```

이와 같은 사용자 정의 사전을 가지고 추가적으로 아래와 같이 불용어를 제거합니다.

[6] http://textminingonline.com/dive-into-nltk-part-iv-stemming-and-lemmatization 를 참조하세요.

```
unique_NN_words1 = set(final_NN_words)

for word in unique_NN_words1:
   if word in customized_stopwords:
      while word in final_NN_words: final_NN_words.remove(word)
```

이렇게 하면 특정 문서에 대해서 전처리 과정이 끝난 결과로 추출된 단어들이 final_NN_words 라는 변수에 저장이 되게 됩니다. 우리는 이 변수의 저장된 단어들을 가지고 추가적인 작업을 하게 됩니다.