8. 텍스트 분석

한승훈

목차

8.1 텍스트 분석 이해

8.1 텍스트 분석의 이해

- NLP? Text Analytics?
  - NLP(Natural Language Processing, 자연어 처리)란 텍스트에서 의미있는 정보를 분석, 추출하고 이해하는 일련의 기술 집합
  - 텍스트 분석(Text Analytics)이란 text mining이라고도 불리며 비정형 텍스트에서 의미있는 정보를 추출하는 것
  - NLP와 Text Analytics 사이에 큰 차이는 없음

8.1 텍스트 분석의 이해

- 텍스트 분석의 영역

  - 텍스트 분류 : 문서가 특정 분류 또는 카테고리에 속하는 것을 예측하는 기법
  - 감성 분석 : 텍스트에 나타나는 주관적인 요소를 분석하는 기법
  - 텍스트 요약 : 텍스트 내에서 중요한 주제나 중심 사상을 추출하는 기법
  - 텍스트 군집화와 유사도 측정 : 비슷한 유형의 문서에 대해 군집화를 수행하는 기법

8.1 텍스트 분석의 이해

- 텍스트 분석 수행 프로세스

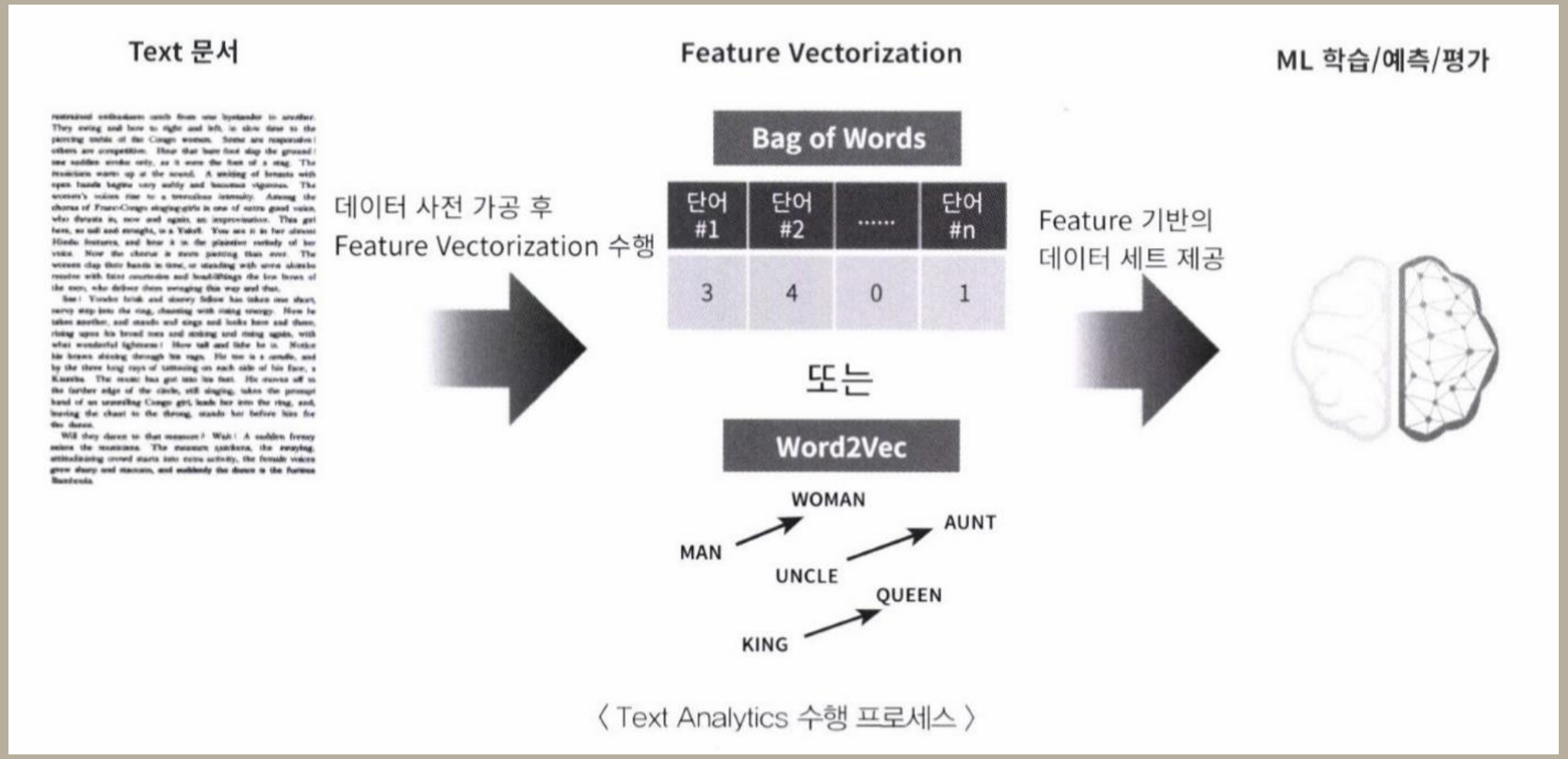
1. 텍스트 사전 준비작업(텍스트 전처리)
   - 텍스트를 피처로 만들기 전에 사전에 클렌징, 대/소문자 변경, 특수문자 삭제 등의 클렌징 작업과 텍스트 정규화 작업을 수행

2. 피처 벡터화/추출
   - 사전 준비 작업으로 가공된 텍스트에서 피처를 추출하고 벡터 값 할당 대표적인 방법으로 BOW와 Word2Vec이 있고, BOW는 대표적으로 Count 기반과 TF-IDF 기반 벡터화가 있음

3. ML 모델 수립 및 학습/예측/평가
   - 피처 벡터화된 데이터 세트에 ML 모델을 적용해 학습/예측 및 평가를 수행

8.1 텍스트 분석의 이해

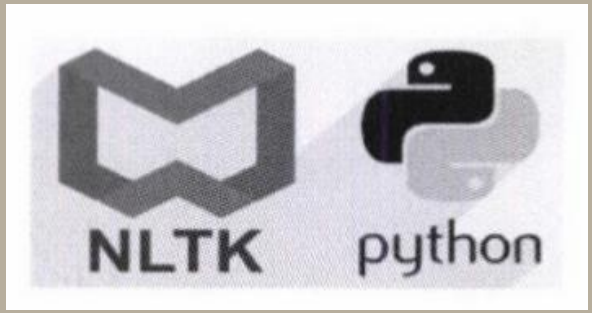
- 텍스트 분석 수행 프로세스 도식화



〈 Text Analytics 수행 프로세스 〉

8.1 텍스트 분석의 이해

- 파이썬 기반의 NLP, 텍스트 분석 패키지



• NLTK : 파이썬의 가장 대표적인 NLP 패키지
방대한 데이터 세트와 서브 모듈을 가지고 있으며 NLP의 거의 모든 영역을 커버, 수행 속도에 아쉬움이 있어 실제 대량의 데이터 기반에서는 제대로 사용되지 못함

• gensim : 토픽 모델링 분야에서 가장 두각을 나타내는 패키지, 오래전부터 토픽 모델링을 쉽게 구현하는 기능을 제공해왔으며 다양한 신기능도 제공, SpaCy와 함께 가장 많이 사용되는 NLP 패키지

• SpaCy : 뛰어난 수행 성능으로 최근 가장 주목을 받는 NLP 패키지, 많은 NLP 애플리케이션에서 SpaCy를 사용하는 사례가 늘고 있음

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

- 텍스트 정규화
  - 텍스트 자체를 바로 피처로 만들 수 없음, 이를 위해 사전에 텍스트를 가공하는 준비 작업이 필요함
    - 클렌징(Cleansing)
    - 토큰화(Tokenizing)
    - 필터링/스톱 워드 제거/철자 수정
    - Stemming
    - Lemmatization

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

- 텍스트 정규화
  - 텍스트 자체를 바로 피처로 만들 수 없음, 이를 위해 사전에 텍스트를 가공하는 준비 작업이 필요함
    - 클렌징(Cleansing)
    - 토큰화(Tokenizing)
    - 필터링/스톱 워드 제거/철자 수정
    - Stemming
    - Lemmatization

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

- 텍스트 정규화
  - 클렌징(Cleansing)
    - 텍스트에서 분석에 방해가 되는 문자, 기호 등을 사전에 제거하는 작업
      ex) HTML, XML, 태그, 특정 기호

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

- 텍스트 정규화
  - 토큰화(Tokenization)
    - 토큰화의 유형은 문서에서 문장을 분리하는 문장 토큰화와 문장에서 단어를 분리하는 단어 토큰화로 나눌 수 있음

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

• 텍스트 정규화

– 텍스트 토큰화(Tokenization)

• 문장 토큰화 : 문장의 마침표(.), 개행문자(\n) 등 문장의 마지막을 뜻하는 기호에 따라 분리하는 것이 일반적

```
from nltk import sent_tokenize
text_sample = 'The Matrix is everywhere its all around us, here even in this room. \
               You can see it out your window or on your television. \
               You feel it when you go to work, or go to church or pay your taxes.'
sentences = sent_tokenize(text=text_sample)
print(type(sentences), len(sentences))
print(sentences)
```

• NTLK 문장 토큰화 예제

【Output】

```
<class 'list'> 3
['The Matrix is everywhere its all around us, here even in this room.', 'You can see it out your window or on
your television.', 'You feel it when you go to work, or go to church or pay your taxes.']
```

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

• 텍스트 정규화

– 텍스트 토큰화(Tokenization)

• 단어 토큰화 : 기본적으로 공백( ), 콤마(,), 마침표(.), 개행문자 등으로 단어를 분리
정규 표현식을 이용해 다양한 유형으로 토큰화를 수행 가능

```
from nltk import word_tokenize

sentence = "The Matrix is everywhere its all around us, here even in this room."
words = word_tokenize(sentence)
print(type(words), len(words))
print(words)
```

【Output】

```
<class 'list'> 15
['The', 'Matrix', 'is', 'everywhere', 'its', 'all', 'around', 'us', ',', 'here', 'even', 'in', 'this', 'room', '.']
```

• NTLK 단어 토큰화 예제

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

• 텍스트 정규화

– 텍스트 토큰화(Tokenization)

• 앞의 문장 토큰화와 단어 토큰화를 조합한 코드

```
from nltk import word_tokenize, sent_tokenize

# 여러 개의 문장으로 된 입력 데이터를 문장별로 단어 토큰화하게 만드는 함수 생성
def tokenize_text(text):

    # 문장별로 분리 토큰
    sentences = sent_tokenize(text)
    # 분리된 문장별 단어 토큰화
    word_tokens = [word_tokenize(sentence) for sentence in sentences]
    return word_tokens

# 여러 문장에 대해 문장별 단어 토큰화 수행.
word_tokens = tokenize_text(text_sample)
print(type(word_tokens), len(word_tokens))
print(word_tokens)
```

【Output】

```
<class 'list'> 3
[['The', 'Matrix', 'is', 'everywhere', 'its', 'all', 'around', 'us', ',', 'here', 'even', 'in',
'this', 'room', '.'], ['You', 'can', 'see', 'it', 'out', 'your', 'window', 'or', 'on', 'your',
'television', '.'], ['You', 'feel', 'it', 'when', 'you', 'go', 'to', 'work', ',', 'or', 'go', 'to',
'church', 'or', 'pay', 'your', 'taxes', '.']]
```

• 문장을 단어별로 하나씩 토큰화 하는 경우 문맥적인 의미는 무시됨 -> 이에 대한 해결 방법으로 'n-gram'이 있음
n-gram은 연속된 n개의 단어를 하나의 토큰화 단위로 분리하는 것

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

• 텍스트 정규화

– 스톱 워드(Stop word) 제거

• 스톱 워드란 분석에 큰 의미가 없는 단어를 지칭 (ex. is, the, a, will)
이러한 단어들의 특성 상 빈번하게 텍스트에 나타나므로 이것들을 사전에 제거하지 않으면
중요 단어로 인식할 가능성이 존재, 따라서 제거해주는 전처리 과정이 필요

```
import nltk

stopwords = nltk.corpus.stopwords.words('english')
all_tokens = []
# 위 예제에서 3개의 문장별로 얻은 word_tokens list에 대해 스톱 워드를 제거하는 반복문
for sentence in word_tokens:
    filtered_words=[]
    # 개별 문장별로 토큰화된 문장 list에 대해 스톱 워드를 제거하는 반복문
    for word in sentence:
        # 소문자로 모두 변환합니다.
        word = word.lower()
        # 토큰화된 개별 단어가 스톱 워드의 단어에 포함되지 않으면 word_tokens에 추가
        if word not in stopwords:
            filtered_words.append(word)
    all_tokens.append(filtered_words)

print(all_tokens)
```

【Output】

```
[['matrix', 'everywhere', 'around', 'us', ',', ',', 'even', 'room', '.'], ['see', 'window', 'television',
'.'], ['feel', 'go', 'work', ',', ',', 'go', 'church', 'pay', 'taxes', '.']]
```

• 앞의 문장 별 단어 토큰화를 수행
한 예제의 스톱 워드를 제거한 예제

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

- 텍스트 정규화
  - Stemming과 Lemmatization
    - Stemming과 Lemmatization은 기본적으로 단어의 원형을 찾는 것이지만 Lemmatization이 Stemming보다 정교하며 의미론적인 기반에서 단어의 원형을 찾음
    - Stemming은 원형 단어로 변환 시 일반적인 방법을 적용하거나 더 단순화된 방법을 적용해 원래 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향이 있음
    - Lemmatization은 이에 반해 품사와 같은 문법적인 요소와 더 의미적인 부분을 감안해 정확한 철자로 된 어근을 찾아줌, 따라서 Stemming보다 변환에 더 오랜 시간이 소요됨

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

- 텍스트 정규화
  - Stemming

```
from nltk.stem import LancasterStemmer
stemmer = LancasterStemmer()

print(stemmer.stem('working'), stemmer.stem('works'), stemmer.stem('worked'))
print(stemmer.stem('amusing'), stemmer.stem('amuses'), stemmer.stem('amused'))
print(stemmer.stem('happier'), stemmer.stem('happiest'))
print(stemmer.stem('fancier'), stemmer.stem('fanciest'))
```

【Output】

```
work work work
amus amus amus
happy happiest
fant fanciest
```

• Stemming을 수행한 결과 단어 원형이 훼손된 경우를 찾아볼 수 있음

8.2 텍스트 사전 준비 작업(텍스트 전처리) – 텍스트 정규화

- 텍스트 정규화
  - Lemmatization

```
from nltk.stem import WordNetLemmatizer

lemma = WordNetLemmatizer()
print(lemma.lemmatize('amusing', 'v'), lemma.lemmatize('amuses', 'v'), lemma.lemmatize('amused',
'v'))
print(lemma.lemmatize('happier', 'a'), lemma.lemmatize('happiest', 'a'))
print(lemma.lemmatize('fancier', 'a'), lemma.lemmatize('fanciest', 'a'))
```

【Output】

```
amuse amuse amuse
happy happy
fancy fancy
```

• Lemmatization을 수행한 결과를 보면 단어 원형이 훼손 없이 추출됨을 알 수 있음