

Revenue growth divisions.

TYU division

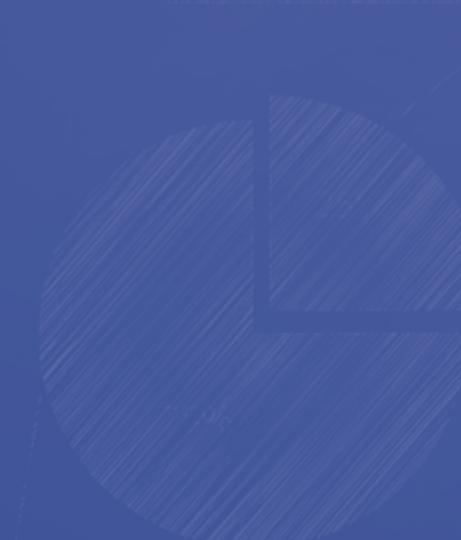
FRT division

Projected sales of main products in 2013

## CHAPTER 2 텍스트 전처리 실습



Share of market activity



Changes in  
product and  
uncertain  
trends in v  
eriments.

Passive market share



**MLP Lab**  
- Prof. 임경태  
- T.A. 김상민

# Contents

2.1 토큰화

---

2.2 정제와 정규화

---

2.3 어간 추출과 표제어 추출

---

2.4 불용어

---

2.5 정규 표현식

---



## 텍스트 전처리

- 풀고자 하는 문제의 용도에 맞게 텍스트를 사전에 처리하는 작업
  - 요리를 할 때 재료를 손질하지 않으면, 요리가 엉망이 되는 것과 같이 텍스트에 제대로 전처리를 하지 않으면 자연어처리 기법들이 제대로 동작하지 않음
- 방법으로 토큰화, 정제, 정규화 등이 있음

## 2.1 토큰화

- 주어진 코퍼스(corpus)에서 토큰(token)이라 불리는 단위로 나누는 작업
  - 상황에 따라 다르지만, 보통 의미있는 단위로 토큰을 정의
- 자연어 처리에서 가장 일반적으로 전처리하는 방법중 하나로, 많은 모델에서 이처럼 토큰화된 데이터를 필요로 함
- 단어 사전을 만들때에도 토큰화가 필요
  - 코퍼스에서 나오는 데이터를 중복없이 만들어서 모델에서 활용하는데 이때 토큰으로 사전을 만듦
  - 이렇게 만든 사전을 통해 BoW, TF-IDF와 같은 분석 방법을 이용할 수 있음

## 2.1 토큰화

- 단어 토큰화(Word Tokenization)

- 토큰의 기준을 단어(word)로 하는 경우로, 가장 일반적인 토큰화 방법
  - ‘단어’는 단어 단위 외에도 단어구, 의미를 갖는 문자열로도 간주되기도 함

ex) Time is an illusion. Lunchtime double so!

→ “Time”, “is”, “an”, “illusion”, “Lunchtime”, “double”, “so”

- 이 예제에서 토큰화 작업은 구두점을 지운 뒤에 띄어쓰기를 기준으로 잘라냈지만,

보통 토큰화 작업은 구두점이나 특수문자를 전부 제거하는 정제 작업만으로 해결되지 않음

구두점이나 특수문자를 전부 제거하면 토큰이 의미를 잃거나, 띄어쓰기만으로 단어 토큰을 구분하기 어려운 경우도 있음

## 2.1 토큰화

- 토큰화 중 생기는 선택의 순간
  - 예상치 못하게 토큰화의 기준을 생각해봐야 하는 경우가 발생
    - 데이터를 어떤 용도로 사용할 것인지에 따라서 용도에 영향이 없는 기준으로 정함

ex) Don't be fooled by the dark sounding name,

Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop.

다음 문장에서 아포스트로피가 들어간 Don't와 Jone's는 토큰화에서 다양한 선택지가 있음

→ "Don't", "Don t", "Dont", "Do n't", "Jone's", "Jone s", "Jone", "Jones"

- 이중 사용자가 원하는 결과가 나오도록 토큰화 도구를 직접 설계할 수도 있지만,

기준에 공개된 도구들을 사용했을 때의 결과가 사용자의 목적과 일치한다면 해당 도구를 사용할 수 있음

## 2.1 토큰화

- 토큰화에서 고려해야 할 사항
  - 구두점이나 특수 문자를 단순 제외해서는 안된다.
    - 마침표(.) 같은 경우는 문장의 경계를 알 수 있는데 도움이 되므로 마침표(.)를 제외하지 않을 수 있음
    - 단어 자체에 구두점이나 특수 문자를 갖고 있는 경우
      - ex) m.p.h, Ph.d, AT&T, \$45.55 (가격), 01/02/06 (날짜), 123,456,789 등
  - 줄임말과 단어 내에 띄어쓰기가 있는 경우
    - we're은 we are의 줄임말로 re와 같이 단어가 줄임말로 쓰일 때 생기는 형태를 접어(clitic)라고 하고, 토큰화할 때 접어를 분리할지 고려해야 함
    - New York이나, rock 'n' roll처럼 하나의 단어 중간에 띄어쓰기가 있지만 하나의 토큰으로 봐야하는 경우도 있음

## 2.1 토큰화

- 문장 토큰화(Sentence Tokenization)
  - 토큰의 단위가 문장 - 문장 분류(sentence segmentation)라고도 부름
  - 코퍼스가 정제되지 않은 상대라면, 문장 단위로 구분되어 있지 않아서 문장 토큰화가 필요할 수 있음
  - 마침표(.)는 문장의 끝이 아니라도 등장할 수 있기 때문에 마침표를 기준으로 문장을 자를 수 없음

ex1) IP 192.168.56.31 서버에 들어가서 로그 파일 저장해서 aaa@gmail.com로 결과 좀 보내줘. 그 후 점심 먹으러 가자.

ex2) Since I'm actively looking for Ph.D. students, I get the same question a dozen times every year.

→ [“IP 192.168.56.31 서버에 들어가서 로그 파일 저장해서 aaa@gmail.com로 결과 좀 보내줘.”, “그 후 점심 먹으러 가자.”]

[“Since I'm actively looking for Ph.D. students, I get the same question a dozen times every year.”]

## 2.1 토큰화

- 한국어 토큰화의 어려움
  - 영어는 New York과 같은 합성어나 he's와 같이 줄임말에 대한 예외처리만 한다면, 띠어쓰기를 기준으로 토큰화를 수행해도 단어 토큰화가 잘 됨
  - 한국어에는 조사라는것이 존재하고, ‘그가’, ‘그에게’, ‘그를’, ‘그와’, ‘그는’과 같이 다양한 조사가 붙음
  - 같은 단어임에도 서로 다른 조사가 붙어서 다른 단어로 인식되면 자연어 처리가 힘들고 번거로워짐  
→ 대부분의 한국어 NLP에서 조사는 분리할 필요가 있음
  - 한국어 토큰화에서는 형태소(morpheme)란 개념을 반드시 이해해야 함

## 2.1 토큰화

- 한국어 토큰화의 어려움
  - 형태소(morpheme)
    - 뜻을 가진 가장 작은 말의 단위
    - 자립 형태소 : 접사, 어미, 조사와 상관없이 자립하여 사용할 수 있는 형태소. 그 자체로 단어가 됨
    - 의존 형태소 : 다른 형태소와 결합하여 사용되는 형태소. 접사, 어미, 조사, 어간을 말함

ex) 에디가 책을 읽었다

자립 형태소 : 에디, 책

의존 형태소 : -가, -을, 읽-, -었, -다

## 2.1 토큰화

- 한국어 토큰화의 어려움
  - 한국어 띄어쓰기가 영어보다 잘 지켜지지 않음
    - 한국어의 경우 띄어쓰기가 지켜지지 않아도 글을 쉽게 이해할 수 있는 언어

ex1) 제가 이렇게 띄어쓰기를 전혀 하지 않고 글을 썼다고 하더라도 글을 이해할 수 있습니다.

ex2) To be or not to be that is the question

## 2.1 토큰화

- 품사 태깅(Part-of-speech tagging)
  - 단어는 표기는 같지만 품사에 따라서 단어의 의미가 달라지기도 함 ex) 'fly' - 날다, 파리
  - 한국어에서도 '못'은 망치를 사용해서 목재 따위를 고정하는 물건과, 못먹는다, 못달린다와 같이 동작 동사를 할 수 없다는 의미로 쓰임
  - 단어의 의미를 제대로 파악하기 위해 토큰화 과정에서 각 단어가 어떤 품사로 쓰였는지 구분하는 작업

## 2.1 토큰화

- 표준 토큰화 예제

- 표준으로 쓰이고 있는 토큰화 방법 중 하나인 Penn Treebank Tokenization의 규칙은 다음과 같음

규칙 1. 하이푼으로 구성된 단어는 하나로 유지한다.

규칙 2. doesn't와 같이 아포스트로피로 ‘접어’가 함께하는 단어는 분리해준다.

```
from nltk.tokenize import TreebankWordTokenizer  
  
tokenizer = TreebankWordTokenizer()  
  
text = "Starting a home-based restaurant may be an ideal. it doesn't have a food chain or restaurant of th  
eir own."  
print('트리뱅크 워드토크나이저 :',tokenizer.tokenize(text))
```

```
트리뱅크 워드토크나이저 : ['Starting', 'a', 'home-based', 'restaurant', 'may', 'be', 'an', 'ideal.', 'it', 'doe  
s', "n't", 'have', 'a', 'food', 'chain', 'or', 'restaurant', 'of', 'their', 'own', '.']
```

- 결과를 보면, 규칙1과 2에 따라서 home-based는 하나의 토큰으로 취급하고, doesn't는 does와 n't로 분리

## 2.2 정제와 정규화

- 정제(cleaning) : 갖고 있는 코퍼스로부터 노이즈 데이터를 제거
  - 불필요한 단어의 제거
  - 자연어가 아니면서 아무 의미도 갖지 않는 글자들 뿐만 아니라 분석하고자 하는 목적에 맞지 않는 불필요 단어들을 노이즈 데이터라고 하기도 함
  - 등장 빈도가 적은 단어
- ex) 100,000개의 메일 데이터에서 총 합 5번 밖에 등장하지 않은 단어가 있다면 이 단어는 분류에 거의 도움이 되지 않음
- 길이가 짧은 단어
  - 영어에서는 길이가 짧은 단어를 삭제하는 것만으로도 크게 의미가 없는 단어들을 제거하는 효과를 볼 수 있음
  - 하지만 한국어는 한자어가 많고, 한 글자만으로도 의미를 가진 경우가 많음

## 2.2 정제와 정규화

- 정제(cleaning) : 갖고 있는 코퍼스로부터 노이즈 데이터를 제거
  - 정규 표현식(Regular Expression)
    - 코퍼스에서 노이즈 데이터의 특징을 잡아낼 수 있다면, 정규 표현식을 통해 제거할 수 있는 경우가 많음
    - HTML문서의 HTML 태그, 뉴스 기사의 게재 시간처럼 코퍼스 내에 계속 등장하는 글자들을 규칙에 기반하여 한번에 제거하는 방식으로 사용 가능

## 2.2 정제와 정규화

- 정규화(normalization) : 표현 방법이 다른 단어들을 통합시켜서 같은 단어로 만듦
  - 자연어 처리에서 전처리, 더 정확하는 정규화의 지향점은 코퍼스로부터 복잡성을 줄이는 것
  - 규칙에 기반한 표기가 다른 단어들의 통합
    - USA와 US, uh-huh와 uhhuh는 형태는 다르지만 같은 의미를 갖고 있으므로 하나의 단어로 정규화해볼 수 있음
    - 어간 추출(stemming)과 표제어 추출(lemmatization) 등이 있음
  - 대, 소문자 통합
    - 영어권 언어에서 단어의 개수를 줄일 수 있는 또 다른 정규화 방법
      - ex) 검색 엔진에서 ferrari 만 쳐도 a Ferrari car의 결과를 볼 수 있음
    - 대문자와 소문자를 무작정 통합해서는 안됨
      - ex) US(미국) - us(우리), General Motors(회사이름), Bush(사람이름)등은 대문자로 유지해야 함

## 2.3 어간 추출과 표제어 추출

- 표제어 추출(Lemmatization)
  - 표제어(Lemma) : 단어의 기본형으로 사전에 등재된 단어
  - 단어들이 다른 형태를 가지더라도, 그 뿐만 아니라 단어를 찾아가서 단어의 개수를 줄일 수 있는지 판단  
ex) am, are, is는 서로 다른 스펠링이지만 그 뿐만 아니라 단어인 be는 이 단어들의 표제어라고 함
- 어간 추출(Stemming)
  - 어간(Stem) : 문법에서 어형 변화의 기초가 되는 부분, 실질적인 뜻을 가지고 있으며 형태가 변하지 않음
  - 정해진 규칙으로 단어의 어미를 자르기 때문에 추출의 결과는 사전에 존재하지 않을 수 있음

## 2.3 어간 추출과 표제어 추출

- 한국어에서의 어간 추출
  - 한국어는 다음과 같이 5언 9품사의 구조를 가짐
  - 용언에 해당되는 '동사'와 '형용사'는 어간과 어미의 결합으로 구성
  - 활용(conjugation)
    - 용언의 어간(stem)이 어미(ending)를 가지는 일
    - 어간이 어미를 취할 때, 어간의 모습이 일정하다면 규칙 활용, 어간이나 어미의 모습이 변하면 불규칙 활용
    - 규칙 활용 ex) 잡(어간) + 다(어미)
    - 불규칙 활용 ex) 들-, 둡- 등이 들/들-, 둡/도록- 와 같이 어간의 형식이 달라지거나  
오르 + 아/어 = 올라, 푸르 + 아/어 = 푸르러 와 같이 일반적인 어미가 아닌 특수한 어미를 취하는 경우
    - 불규칙활용의 경우 어간의 모습이 바뀌었으므로 단순한 분리만으로 어간 추출이 되지 않고 좀 더 복잡한 규칙을 필요로 함

언	품사
체언	명사, 대명사, 수사
수식언	관형사, 부사
관계언	조사
독립언	감탄사
용언	동사, 형용사

## 2.3 어간 추출과 표제어 추출

- 표제어 추출과 어간 추출 실습

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']

print('표제어 추출 전 :', words)
print('표제어 추출 후 :', [lemmatizer.lemmatize(word) for word in words])
```

표제어 추출 전 : ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']  
 표제어 추출 후 : ['policy', 'doing', 'organization', 'have', 'going', 'love', 'life', 'fly', 'dy', 'watched', 'ha', 'starting']

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

stemmer = PorterStemmer()

sentence = "This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights and soundings--with the single exception of the red crosses and the written note s."
tokenized_sentence = word_tokenize(sentence)

print('어간 추출 전 :', tokenized_sentence)
print('어간 추출 후 :', [stemmer.stem(word) for word in tokenized_sentence])
```

어간 추출 전 : ['This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', "'s", 'chest', ',', 'but', 'an', 'accurate', 'copy', ',', 'complete', 'in', 'all', 'things', '--', 'names', 'and', 'heights', 'and', 'soundings', '--', 'with', 'the', 'single', 'exception', 'of', 'the', 'red', 'crosses', 'and', 'the', 'written', 'notes', '.']  
 어간 추출 후 : ['thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone', "'s", 'chest', ',', 'but', 'an', 'accur', 'copi', ',', 'complet', 'in', 'all', 'thing', '--', 'name', 'and', 'height', 'and', 'sound', '--', 'with', 'the', 'singl', 'except', 'of', 'the', 'red', 'cross', 'and', 'the', 'written', 'note', '.']

## 2.4 불용어

- I, my, me, over, 조사, 접미사와 같이 문장에서 자주 등장하지만  
실제 의미에는 거의 기여하지 않는 단어를 불용어(stopword)라고 함
- 데이터에서 유의미한 단어 토큰을 선별하기 위해서는 큰 의미가 없는 단어 토큰을 제거하는 작업이 필요
  - NLTK에서는 위와 같은 1000여개 이상의 영어 단어를 불용어로 패키지 내에서 미리 정의하고 있음
  - 또한 개발자가 직접 불용어를 정의할 수도 있음
- 한국어에서 불용어를 제거하는 방법으로 간단하게는 토큰화 후에 조사, 접속사 등을 제거하는 방법이 있음

## 2.4 불용어

- NLTK를 통해서 불용어 제거하기

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from konlpy.tag import Okt
```

```
example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example)

result = []
for word in word_tokens:
    if word not in stop_words:
        result.append(word)

print('불용어 제거 전 :', word_tokens)
print('불용어 제거 후 :', result)
```

```
불용어 제거 전 : ['Family', 'is', 'not', 'an', 'important', 'thing',
'.', 'It', "'s", 'everything', '.']
불용어 제거 후 : ['Family', 'important', 'thing', '.', 'It', "'s", 'ev
erything', '.']
```

## 2.4 불용어

- NLTK를 통해서 불용어 제거하기

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from konlpy.tag import Okt
```

```
example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example)

result = []
for word in word_tokens:
    if word not in stop_words:
        result.append(word)

print('불용어 제거 전 :', word_tokens)
print('불용어 제거 후 :', result)
```

```
불용어 제거 전 : ['Family', 'is', 'not', 'an', 'important', 'thing',
'.', 'It', "'s", 'everything', '.']
불용어 제거 후 : ['Family', 'important', 'thing', '.', 'It', "'s", 'ev
erything', '.']
```

## 2.5 정규 표현식

- 정규 표현식(Regular Expression)
  - 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어
- 정규 표현식을 위해 사용되는 문법 중 특수 문자들은 다음과 같음
  - . : 한 개의 임의의 문자를 나타냄 (줄바꿈 문자인 \n은 제외)
  - ? : 앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있음(문자가 0개 또는 1개)
  - \* : 앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있음(문자가 0개 이상)
  - + : 앞의 문자가 최소 한 개 이상 존재
  - ^ : 뒤의 문자열로 문자열이 시작
  - \$ : 앞의 문자열로 문자열이 끝남
  - {숫자} : 숫자만큼 반복
  - {숫자1, 숫자2} : 숫자1 이상 숫자2 이하만큼 반복
  - {숫자, } : 숫자 이상만큼 반복
  - [ ] : 대괄호 안의 문자들 중 한 개의 문자와 매치, [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재한다면 매치를 의미.  
[a-zA-Z]와 같이 범위를 지정할 수도 있음. [a-zA-Z]는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미
  - [^문자] : 해당 문자를 제외한 문자를 매치
  - | : A|B와 같이 쓰이며 A또는 B의 의미를 가짐

## 2.5 정규 표현식

- 정규 표현식 모듈에서 지원하는 함수

\\\ : 역슬래쉬 문자 자체를 의미.

\\d : 모든 숫자를 의미. [0-9]와 동일

\\D : 숫자를 제외한 모든 문자를 의미. [^0-9]와 동일

\\s : 공백을 의미. [\t\n\r\f\v]와 동일

\\S : 공백을 제외한 문자를 의미. [^\t\n\r\f\v]와 동일

\\w : 문자 또는 숫자를 의미. [a-zA-Z0-9]와 동일

\\W : 문자 또는 숫자가 아닌 문자를 의미. [^a-zA-Z0-9]와 동일

## 2.5 정규 표현식

- 정규 표현식 문법에 자주 쓰이는 문자 규칙

`re.compile()` : 정규 표현식을 컴파일하는 함수로 파이썬에게 전해주는 역할

`re.search()` : 문자열 전체에 대해서 정규 표현식과 매치되는지를 검색

`re.match()` : 문자열의 처음이 정규 표현식과 매치되는지를 검색

`re.split()` : 정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴

`re.findall()` : 문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴, 매치되는 문자열이 없다면 빈 리스트가 리턴됨

`re.finditer()` : 문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴

`re.sub()` : 문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체

## 2.5 정규 표현식

- 정규 표현식을 이용한 토큰화

```
from nltk.tokenize import RegexpTokenizer

text = "Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a
pastry shop"

tokenizer1 = RegexpTokenizer("[\w]+")
tokenizer2 = RegexpTokenizer("\s+", gaps=True)

print(tokenizer1.tokenize(text))
print(tokenizer2.tokenize(text))
```

```
['Don', 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'Mr', 'Jone', 's', 'Orphanage', 'i
s', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
["Don't", "be", "fooled", "by", "the", "dark", "sounding", "name,", "Mr.", "Jone's", "Orphanage", "is", "a
s", "cheery", "as", "cheery", "goes", "for", "a", "pastry", "shop"]
```

# Practice!

- 학생 실습 코딩

- 실습출처
- <https://wikidocs.net/book/2155>
- <https://en.wikipedia.org/wiki/SpaCy>
- <https://ungodly-hour.tistory.com/37>

**감사합니다.**