

Text Mining

Bok, Jong Soon
javaexpert@nate.com
<https://github.com/swacademy>

Text Mining



Text Mining개요

- Text Data 접근(Text-as-Data Approach)
 - Digital화된 Text를 수치형 Data로 간주하여 Algorithm을 적용해서 분석/요약하는 통계기법
- 위계적 질서를 갖는 Text를 Row와 Column의 Matrix 형태를 갖는 Data로 개념화.
 - 문자(Letter) → 형태소(Morpheme) → 단어(World) → 문장(Sentence) → 단락(Paragraph) → 문서(Document) → 말뭉치(Corpus)
- Text Data를 이해하고 모형화하는 작업
- Token을 근거로 문서에 나타난 주제(Topic)이나 감정(Sentiment)를 추정.

Text Mining개요(Cont.)

■ Data Mining

- 대규모 Big Data(정형화된 Data)에서 가치 있는/의미 있는 정보를 추출하는 기술
- 사람이 예측할 수 없는 의미 있는 경향과 규칙까지 발견하기 위해서 대량의 Big Data로부터 자동화 또는 반자동화 도구를 활용해 탐색하고 분석하는 과정
- 고급 통계 분석과 Modeling 기법을 적용하여 Data 안의 Pattern과 관계를 찾아내는 과정.

Text Mining개요(Cont.)

■ Text Mining

- 대규모의 Text 문서에서 의미 있는 추출하는 기술
- 비정형 Data(글자/텍스트)를 정형화 및 특징을 추출하는 과정이 필요.
- Text 덩어리 안에서 단어들을 분해해 단어의 출현빈도나 단어들 간의 관계성을 파악하여 의미 있는 정보를 추출해내는 기술
- 정보 검색, Data Mining, Machine Learning, 통계학, Computer 언어학(Computational linguistics) 및 Computer 공학 등이 결합된 분야
- 특히 분석 대상이 형태가 일정하지 않고 다루기 힘든 비정형 데이터이므로 인간의 언어를 Computer가 인식해 처리하는 자연어 처리(NLP) 방법과 관련이 깊음.

Text Mining개요(Cont.)

- 전통적 내용분석 방식이 유지되기 어려운 이유
 - Social Media, Online 공간에서 기존과는 다른 새로운 Text가 넘쳐나고 있다.
 - 내용분석을 실시할 coder들을 고용하고 교육시키기 위한 비용과 시간이 더욱 많이 필요.
 - Coder들은 인간이기 때문에 검사-재검사 신뢰도(Test-Retest Reliability) 낮다.
 - 방대한 Text를 분석하는 데 엄청난 시간과 비용이 소요.

Text Mining개요(Cont.)

■ "It is good" vs "It is bad"

	bad	good	is	it	Sentiment
"It is good"	0	1	1	1	?
"It is bad"	1	0	1	1	?

■ Text-as-Data

- Data Matrix
- Numerical Array : 분석 단위 X 단어
- Document-Term Matrix(DTM) : 문서 X 단어 행렬

Text Mining개요(Cont.)

- Text Data / DTM(문서 X 단어 행렬, Document-Term Matrix)를 분석하는 방법
 - Dictionary-based Approach
 - 사전(事前, A Priori)에 규정된 단어의 의미를 기반으로 Text의 의미 추정.
 - Computer Algorithm을 이용한 Text Mining
 - 감정분석(Sentiment Analysis)
 - Machine Learning을 이용한 기법
 - 단어의 의미는 기계학습 사후(事後, A Post Hoc)에 추정.

Text Mining개요(Cont.)

■ 지도 기계학습(Supervised Machine Learning)

- 기계학습을 위해 사용되는 Text Data 중 일부에서 Text의 의미가 알려져 있는 경우
- 훈련 Data는 예측변수와 결과변수로 구성
- 예측변수와 단어들의 속성(예:빈도 ; Frequency 등)과 이 예측변수를 통해 예측되는 결과변수인 Text Data의 분석 단위에 나타난 의미(예:Topic이나 Sentiment 등)으로 구성.
- 예측변수와 결과변수의 관계를 최적으로 설명할 수 있는 함수 추출.
- 문서에 표출된 감정이 어떤지에 대한 인간의 판단과 문서에 등장하는 단어들의 관계를 지도학습을 통해 추정한 후, 새로운 문서에 지도학습으로 추정된 문서분류모형을 적용
- Logistic Regression, Naïve Bayes Classification, SVM, Boosting, Deep Learning

Text Mining개요(Cont.)

- 비지도 기계학습(Unsupervised Machine Learning)
 - Text Data의 의미가 전혀 알려져 있지 않고 기계학습을 통해 Text Data의 의미를 추정하는 경우.
 - PCA, Cluster Analysis
 - 연구자가 분석 단계마다 자신의 주관적 판단의 개입이 필요.

기존 Data Analysis vs Big Data Analysis

기존 Data Analysis	Big Data Analysis
<ul style="list-style-type: none">• Data Mining• Machine Learning	<ul style="list-style-type: none">• Text Mining• Sentiment Analysis• Social Network Analysis• Text Clustering

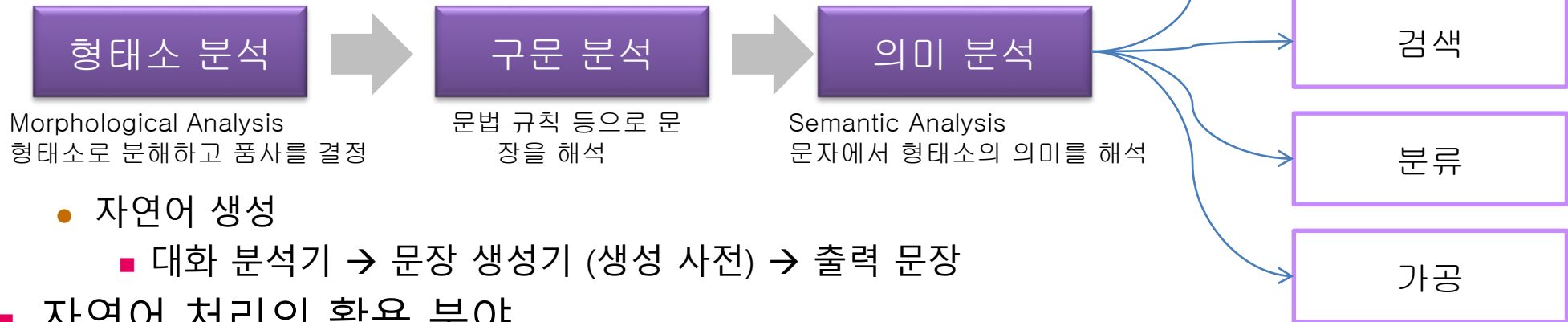
자연어 처리

■ 자연어 처리 (Natural Language Processing, NLP)

- 자연어 : 사람들이 일상적으로 사용하는 언어
- 인공어 : Artificial Language, 사람들이 필요에 의해서 만든 언어
에스페란토, C언어, Java 등

■ 자연어 처리 분야

- 자연어 이해 : 형태소 분석 → 의미 분석 → 대화 분석



- 자연어 생성

- 대화 분석기 → 문장 생성기 (생성 사전) → 출력 문장

■ 자연어 처리의 활용 분야

- 맞춤법 검사, 번역기, 검색 엔진, 키워드 분석 등

■ 문서 (Document) → 문단 (Paragraph) → 문장 (Sentence) → 어절 (Word phrase) → 형태소 (Morpheme) → 음절 (Syllable) → 음소 (Phoneme)

자연어 처리 용어

용어	상세
형태소(Morpheme)	의미를 가진 최소 단위입니다.
용언	꾸미는 말(동사, 형용사)입니다. 용언은 어근 + 어미로 구성됩니다.
어근(Stem)	용언이 활용할 때, 원칙적으로 모양이 변하지 않는 부분입니다.
어미	용언이 활용할 때, 변하는 부분으로 문법적 기능 수행합니다. 어미에는 연결 어미, 선어말 어미 + 종결 어미가 있습니다.
자모	문자 체계의 한 요소(자음, 모음)입니다.
품사	명사, 대명사, 수사, 동사, 형용사, 관형사, 부사, 감탄사, 조사가 있습니다.
어절 분류	명사 + 주격 조사, 명사 + 목적격 조사, 명사 + 관형격 조사, 동사 + 연결 어미 또는 동사 + 선어말 어미 + 종결 어미 등으로 분류합니다.
불용어(Stopword)	검색 등에서 의미가 없어 무시되도록 설정된 단어들입니다.
n-gram	문자의 빈도와 문자간 관계를 의미합니다. "안녕하세요"를 2-gram으로 나누면, "안녕", "녕하", "하세", "세요"로 나눌 수 있습니다.

자연어 처리 학습 주제

- 텍스트 전처리 : 토큰화, 정제, 형태소 분석, 불용어 처리.
Label Encoding
- 개수 기반 단어 표현 : 문장 내에서 단어들의 빈도수를 측정해서 이를 기반으로 데이터를 분석할 수 있는 형태로 만드는 것
- 문서 유사도(Document Similarity) : 단어들을 수치화 한 후 이를 기반으로 단어들 사이의 거리를 계산해서 문서 간의 단어들의 차이를 계산하는 것
- Topic Modeling : 텍스트 본문의 숨겨진 의미 구조를 발견하기 위해 사용되는 Text Mining 기법
- 연관 분석(Association) : 문서 내의 단어들을 이용해서 연관 분석을 실시

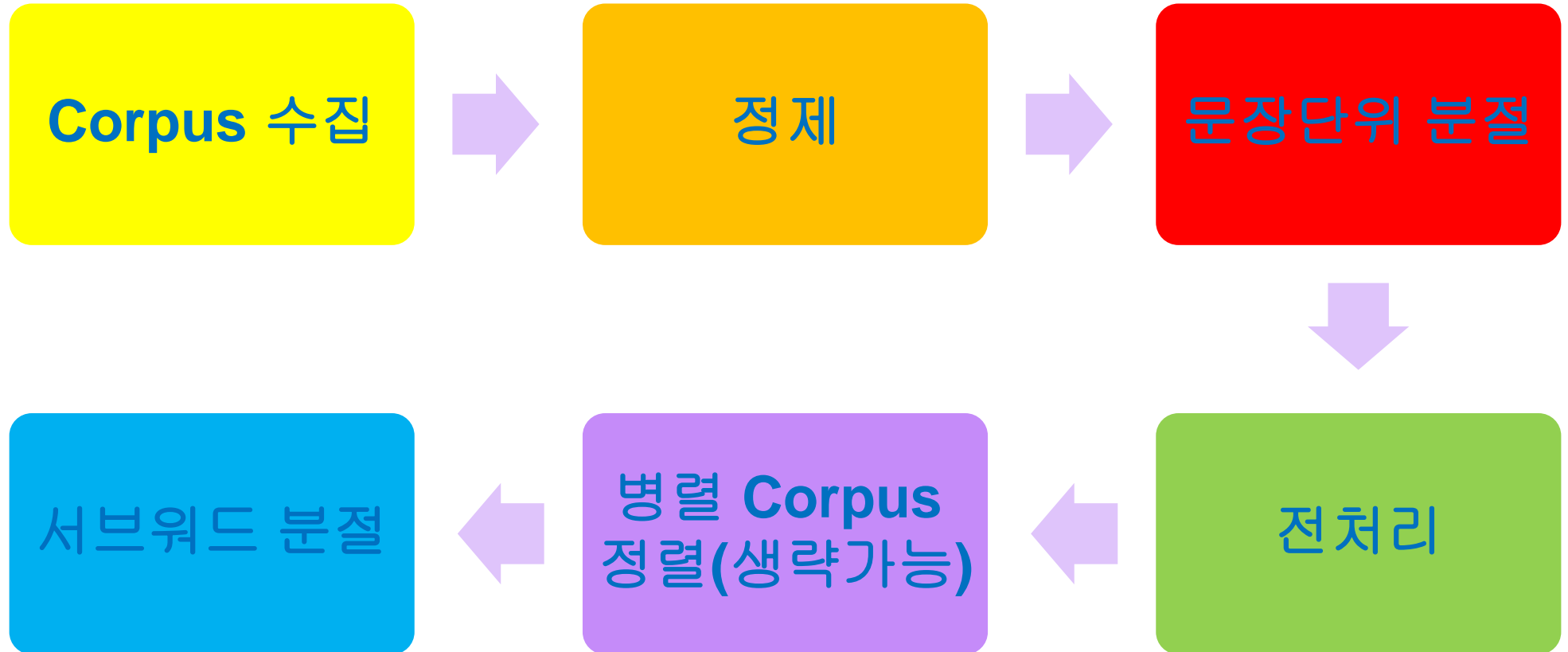
자연어 처리 학습 주제 (Cont.)

- Deep Learning을 이용한 자연어 처리 : RNN, LSTM 등의 인공 신경망 Algorithm을 이용해서 Deep Learning으로 자연어 처리
- Word Embedding : Word2vec 패키지를 이용해서 단어를 벡터로 표현하는 방법으로 희소 표현에서 밀집 표현으로 변환하는 것
- 텍스트 분류(Text Classification) : 텍스트를 입력으로 받아, 텍스트가 어떤 종류의 범주(Class)에 속하는지를 구분하는 작업
- Tagging : 각 단어가 어떤 유형에 속해 있는지를 알아내는 것
- 번역(Translation) : 챗봇(Chatbot) 또는 기계 번역(Machine Translation)

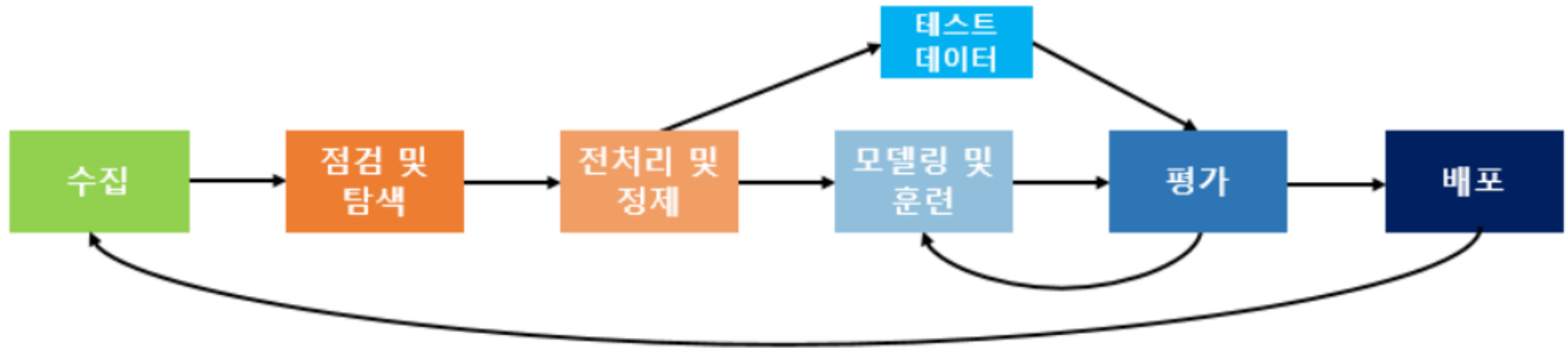
자연어(Natural Language Processing, NLP)

- 사람이 사용하는 언어를 Computer로 처리하는 각종 방법
- 크게 2가지로 나눈다
 - 통계적 자연어 처리
 - 언어학 자연어 처리

자연어(Natural Language Processing, NLP) Workflow



Machine Learning Workflow



Corpus 수집

- 공개된 Data의 Crawling or 구매
- txt, csv, xml, html, Database ...
- urllib.request, requests, BeautifulSoup, lxml, Selenium, Scrapy ...
- 저작권 같은 법적인 문제 조심
- 불필요한 Traffic으로 Web Server 부담 가중

점검 및 탐색

- 탐색적 데이터 분석(Exploratory Data Analysis, EDA) 단계
- 수집된 Data를 점검하고 탐색하는 단계
- 대소문자 통일
- 정규 표현식을 이용한 정제
- Noise Data, Data의 구조 파악 등.

문장 단위 분절

- 한 개의 line에는 한 문장만 있어야
- 단순한 마침표 기준으로 처리시 문제점 발생(예:U.S.)
- Algorithm 필요 or NLTK(>=3.2.5)

자연어처리는 인공지능의 한 줄기입니다. seq2seq의 등장 이후로 Deep Learning을 활용한 자연어처리는 새로운 전기를 맞이하게 되었습니다. 문장을 받아 단순히 수치로 나타내던 시절을 넘어, 원하는 대로 문장을 만들어낼 수 있게 된 것입니다.

자연어처리는 인공지능의 한 줄기입니다. seq2seq의 등장 이후로 \n Deep Learning을 활용한 자연어처리는 새로운 전기를 맞이하게 \n 되었습니다. 문장을 받아 단순히 수치로 나타내던 시절을 넘어, 원하는 \n 대로 문장을 만들어낼 수 있게 된 것입니다.

전처리 및 정제(Preprocessing and Cleaning)

- Tokenizing
- Cleaning
- Normalization
- Stemming
- Lemmatizing
- Stopword Extraction
- Splitting Data
- Integer Encoding & One-Hot Encoding
- Subword Segmentation

전처리 및 정제(Preprocessing and Cleaning) (Cont.)

언어	프로그램명	제작 언어	특징
한국어	Mecab	C++	일본어 Mecab을 wrappin했으며, 속도가 가장 빠름. Windows에서 사용 불가, 설치 다소 까다로움.
한국어	KoNLPy	Python Wrapping	PIP를 통해 설치 가능. 사용이 쉬우나 속도가 다소 느림.
일본어	Mecab	C++	속도가 빠름.
중국어	Stanford Parser	Java	미국 Standford에서 개발
중국어	PKU Parser	Java	북경대에서 개발. 위와 거의 성능 차이 없음.
중국어	Jieba	Python	가장 최근에 개발. Python으로 개발되어 시스템 구성에 용이.

- KorQuAD(<https://korquad.github.io/>)
- Khaiii(<https://tech.kakao.com/2018/12/13/khaiii/>)

Subword 분절

- BPE(Byte Pair Encoding) Algorithm 사용
- *단어는 의미를 가진 더 작은 서브워드들의 조합으로 이루어진다*는 가정으로 사용
- 어휘 수를 줄이고, 효과적으로 희소성 줄어듦.

언어	단어	조합
영어	concentrate	con(=together) + centr(=center) + ate(=make)
한국어	집중(集中)	集(모을 집) + 中(가운데 중)

Subword 분절 (Cont.)

언어	단어	조합
영어	seq2seq streams	se + q + 2 + se + q stream + s
한국어	러닝 신경망 자연어	러 + 닝 신경 + 망 자연 + 어

■ Open Source

- Sennrich

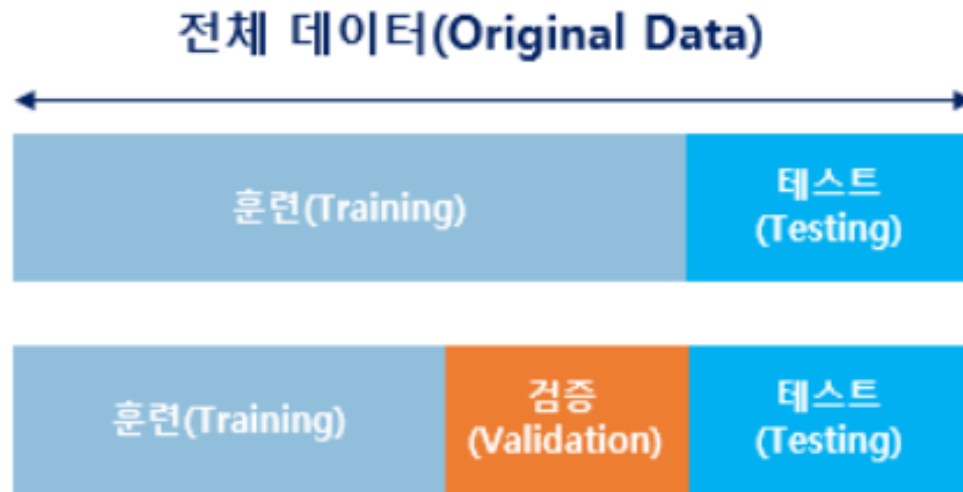
- <https://github.com/rsennrich/subword-nmt>

- Google의 SentencePiece

- <https://github.com/google/sentencepiece>

Modeling and Training

- 적절한 Machine Learning Algorithm 선택
- 전처리가 완료된 Data(학습지)를 가지고 기계에게 학습
- Test Data(수능시험) 준비
- 검증(Validation) Data(모의고사)



평가(Evaluation)와 배포(Deployment)

- 훈련 후, 학습된 Data를 Test Data로 평가
- 다양한 Machine Learning Algorithm에 따라 다양한 평가방법 사용
- 학습이 성공적이면 완성된 Model 배포

NLTK 자연어 처리 Package

NLTK(Natural Language Toolkit) Package

- 교육용으로 개발된 자연어 처리와 문서 분석용 Python Package
- <https://www.nltk.org>
- NLTK 패키지가 제공하는 주요 기능
 - 말뭉치(corpus)
 - 토큰 생성(tokenizing)
 - 형태소 분석(morphological analysis)
 - 품사 태깅(POS tagging)
- Cross-Platform Open Source

NLTK(Natural Language Toolkit) Package (Cont.)

언어 처리 분야	NLTK Module	기능
Corpora 접근	corpus	corpora와 lexicon에 대한 표준화된 interface
문자열 처리	tokenize, stem	Tokenizing, 어간 추출(Stemming)
Collocation 발견	collocations	t-test, chi-squared, point-wise
품사 Tagging	tag	n-gram, backoff, Brill, HMM, TnT
기계 학습	classify, cluster, tbl	Decision Making Tree, Naïve Bayes, k-means 등
Chunking	chunk	정규식, n-gram, named-entity
Parsing	parse, ccg	Chart, 특징 기반, 단일화, 확률론적 의존성
구문 해석	sem, inference	Lambda 미적분학, 1차 논리, 모델 검사
평가 지표	metrics	정밀도(precision), 재현율(recall), 계수 일치도 (Agreement coefficients)
확률 및 추정	probability	빈도 분포, 확률 분포
응용프로그램	app, chat	용어 색인기(Concordancer), Parser, WorldNet Browser, Chatbot
언어 현장 조사	toolbox	SIL, Toolbox 형식으로 데이터 조작

말뭉치(Corpus)

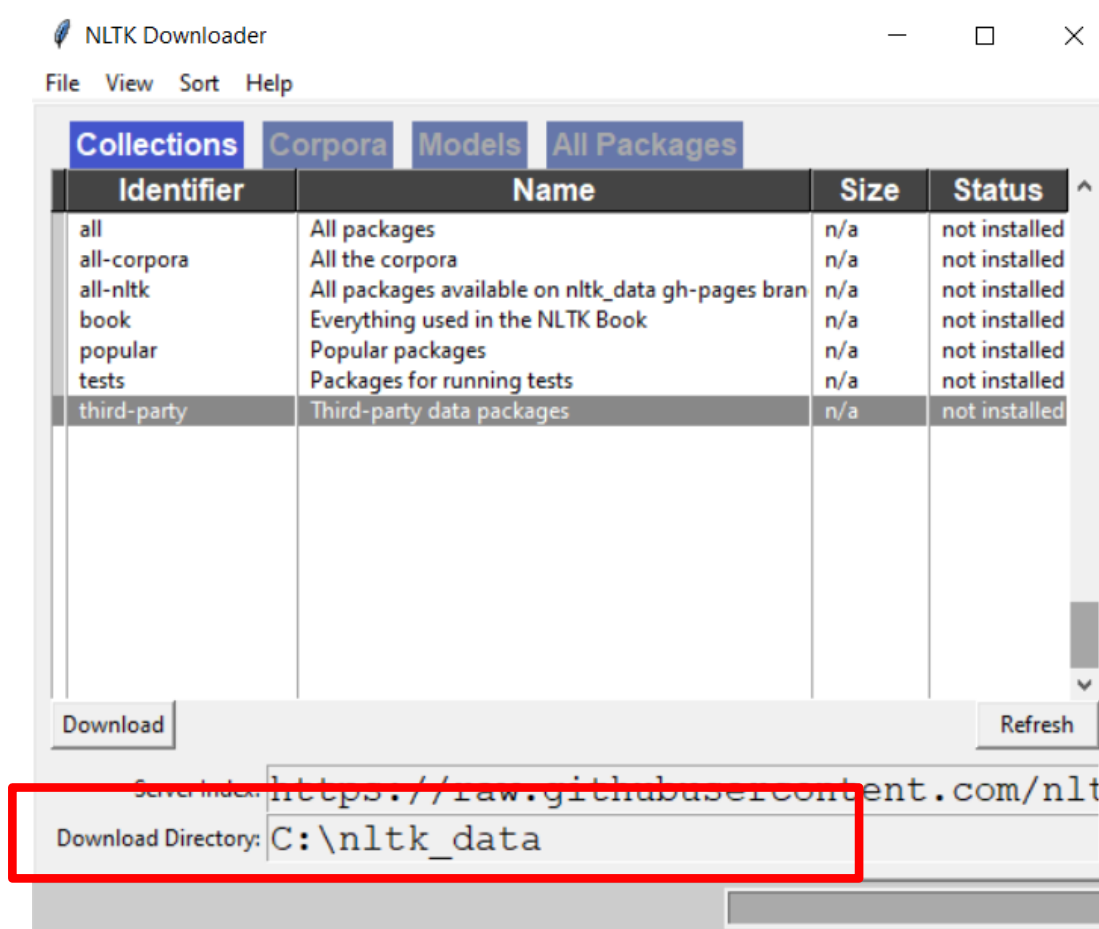
- 연구 대상 분야의 언어 현실을 총체적으로 보여 주는 자료의 집합
- Computer를 이용해 자연어 분석 작업을 할 수 있도록 만든 문서 집합
- Machine Learning을 수행하기 위한 훈련 Data
- NLTK에서는
 - 단순히 소설, 신문 등의 문서
 - 품사, 형태소 등의 보조적 의미를 추가
 - 쉬운 분석을 위해 구조적인 형태로 정리
- NLTK에서 corpus module로 학습용 말뭉치 제공
- <http://www.nltk.org/data.html>
- **`nltk.download()`** 함수로 download 해야 함.

말뭉치(Corpus) (Cont.)

```
import nltk
```

```
nltk.download()
```

showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml



Penn Treebank Data

- NLTK를 설명할 목적으로 사용
- Tokenizing, Tagging, Chunking, Parsing 등에 사용
- Treebank
 - 구문 주석 말뭉치
 - 구조 분석을 좀 더 정교하게 만든 Tree Structure의 집합
- 400만 어절 규모의 Treebank 중 가장 유명한 것
- 주로 WallStreet Journal의 문장들로 구성.
- `nltk.download('treebank')`

```
[nltk_data] Downloading package treebank to C:\nltk_data...  
[nltk_data] Package treebank is already up-to-date!
```

Penn Treebank Data (Cont.)

NLTK Downloader

File View Sort Help

Collections Corpora Models All Packages

Identifier	Name	Size	Status
shakespeare	Shakespeare XML Corpus Sample	464.3 KB	not installed
sinica_treebank	Sinica Treebank Corpus Sample	878.2 KB	not installed
smultron	SMULTRON Corpus Sample	162.3 KB	not installed
state_union	C-Span State of the Union Address Corpus	789.8 KB	not installed
stopwords	Stopwords Corpus	22.6 KB	not installed
subjectivity	Subjectivity Dataset v1.0	509.4 KB	not installed
swadesh	Swadesh Wordlists	22.3 KB	not installed
switchboard	Switchboard Corpus Sample	772.6 KB	not installed
timit	TIMIT Corpus Sample	21.2 MB	not installed
toolbox	Toolbox Sample Files	244.7 KB	not installed
treebank	Penn Treebank Sample	1.7 MB	installed
twitter_samples	Twitter Samples	15.3 MB	not installed
udhr	Universal Declaration of Human Rights Corpus	1.1 MB	not installed
udhr2	Universal Declaration of Human Rights Corpus (Ur	1.6 MB	not installed
unicode_samples	Unicode Samples	1.2 KB	not installed
universal_treebanks_v	Universal Treebanks Version 2.0	24.7 MB	not installed

Download Refresh

Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

Download Directory: C:\nltk_data

Finished installing treebank

Penn Treebank Data (Cont.)

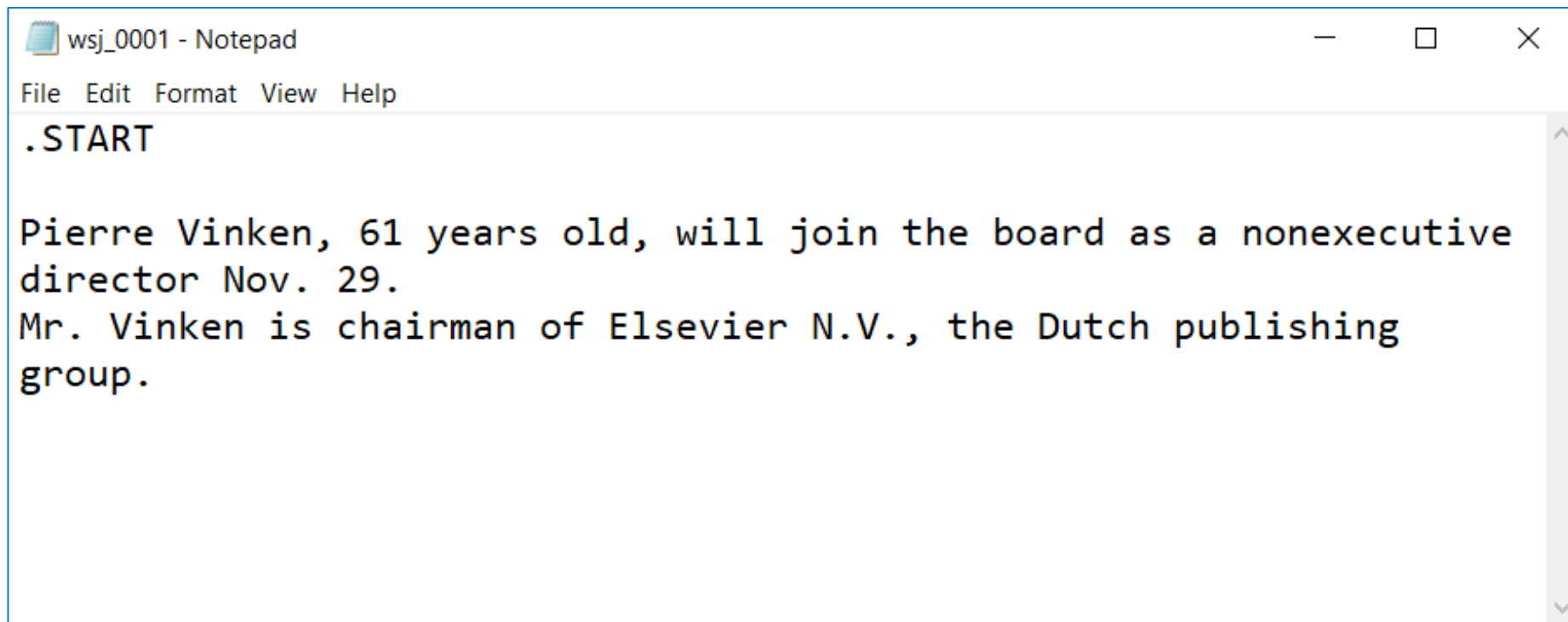
■ File들의 id 확인

```
from nltk.corpus import treebank  
print(treebank.fields())
```

```
['wsj_0001.mrg', 'wsj_0002.mrg', 'wsj_0003.mrg', 'wsj_0004.mrg', 'wsj_0005.mrg', 'wsj_0006.mrg', 'wsj_0007.mrg', 'wsj_0008.mrg', 'wsj_0009.mrg', 'wsj_0010.mrg', 'wsj_0011.mrg', 'wsj_0012.mrg', 'wsj_0013.mrg', 'wsj_0014.mrg', 'wsj_0015.mrg', 'wsj_0016.mrg', 'wsj_0017.mrg', 'wsj_0018.mrg', 'wsj_0019.mrg', 'wsj_0020.mrg', 'wsj_0021.mrg', 'wsj_0022.mrg', 'wsj_0023.mrg', 'wsj_0024.mrg', 'wsj_0025.mrg', 'wsj_0026.mrg', 'wsj_0027.mrg', 'wsj_0028.mrg', 'wsj_0029.mrg', 'wsj_0030.mrg', 'wsj_0031.mrg', 'wsj_0032.mrg', 'wsj_0033.mrg', 'wsj_0034.mrg', 'wsj_0035.mrg', 'wsj_0036.mrg', 'wsj_0037.mrg', 'wsj_0038.mrg', 'wsj_0039.mrg', 'wsj_0040.mrg', 'wsj_0041.mrg', 'wsj_0042.mrg', 'wsj_0043.mrg', 'wsj_0044.mrg', 'wsj_0045.mrg', 'wsj_0046.mrg', 'wsj_0047.mrg', 'wsj_0048.mrg', 'wsj_0049.mrg', 'wsj_0050.mrg', 'wsj_0051.mrg', 'wsj_0052.mrg', 'wsj_0053.mrg', 'wsj_0054.mrg', 'wsj_0055.mrg', 'wsj_0056.mrg', 'wsj_0057.mrg', 'wsj_0058.mrg', 'wsj_0059.mrg', 'wsj_0060.mrg', 'wsj_0061.mrg', 'wsj_0062.mrg', 'wsj_0063.mrg', 'wsj_0064.mrg', 'wsj_0065.mrg', 'wsj_0066.mrg', 'wsj_0067.mrg', 'wsj_0068.mrg', 'wsj_0069.mrg', 'wsj_0070.mrg', 'wsj_0071.mrg', 'wsj_0072.mrg', 'wsj_0073.mrg', 'wsj_0074.mrg', 'wsj_0075.mrg', 'wsj_0076.mrg', 'wsj_0077.mrg', 'wsj_0078.mrg', 'wsj_0079.mrg', 'wsj_0080.mrg', 'wsj_0081.mrg', 'wsj_0082.mrg', 'wsj_0083.mrg', 'wsj_0084.mrg', 'wsj_0085.mrg', 'wsj_0086.mrg', 'wsj_0087.mrg', 'wsj_0088.mrg', 'wsj_0089.mrg', 'wsj_0090.mrg', 'wsj_0091.mrg', 'wsj_0092.mrg', 'wsj_0093.mrg', 'wsj_0094.mrg', 'wsj_0095.mrg', 'wsj_0096.mrg', 'wsj_0097.mrg', 'wsj_0098.mrg', 'wsj_0099.mrg', 'wsj_0100.mrg', 'wsj_0101.mrg', 'wsj_0102.mrg', 'wsj_0103.mrg', 'wsj_0104.mrg', 'wsj_0105.mrg', 'wsj_0106.mrg', 'wsj_0107.mrg', 'wsj_0108.mrg', 'wsj_0109.mrg', 'wsj_0110.mrg', 'wsj_0111.mrg', 'wsj_0112.mrg', 'wsj_0113.mrg', 'wsj_0114.mrg', 'wsj_0115.mrg', 'wsj_0116.mrg', 'wsj_0117.mrg', 'wsj_0118.mrg', 'wsj_0119.mrg', 'wsj_0120.mrg', 'wsj_0121.mrg', 'wsj_0122.mrg', 'wsj_0123.mrg', 'wsj_0124.mrg', 'wsj_0125.mrg', 'wsj_0126.mrg', 'wsj_0127.mrg', 'wsj_0128.mrg', 'wsj_0129.mrg', 'wsj_0130.mrg', 'wsj_0131.mrg', 'wsj_0132.mrg', 'wsj_0133.mrg', 'wsj_0134.mrg', 'wsj_0135.mrg', 'wsj_0136.mrg', 'wsj_0137.mrg', 'wsj_0138.mrg', 'wsj_0139.mrg', 'wsj_0140.mrg', 'wsj_0141.mrg', 'wsj_0142.mrg', 'wsj_0143.mrg', 'wsj_0144.mrg', 'wsj_0145.mrg', 'wsj_0146.mrg', 'wsj_0147.mrg', 'wsj_0148.mrg', 'wsj_0149.mrg', 'wsj_0150.mrg', 'wsj_0151.mrg', 'wsj_0152.mrg', 'wsj_0153.mrg', 'wsj_0154.mrg', 'wsj_0155.mrg', 'wsj_0156.mrg', 'wsj_0157.mrg', 'wsj_0158.mrg', 'wsj_0159.mrg', 'wsj_0160.mrg', 'wsj_0161.mrg', 'wsj_0162.mrg', 'wsj_0163.mrg', 'wsj_0164.mrg', 'wsj_0165.mrg', 'wsj_0166.mrg', 'wsj_0167.mrg', 'wsj_0168.mrg', 'wsj_0169.mrg', 'wsj_0170.mrg', 'wsj_0171.mrg', 'wsj_0172.mrg', 'wsj_0173.mrg', 'wsj_0174.mrg', 'wsj_0175.mrg', 'wsj_0176.mrg', 'wsj_0177.mrg', 'wsj_0178.mrg', 'wsj_0179.mrg', 'wsj_0180.mrg', 'wsj_0181.mrg', 'wsj_0182.mrg', 'wsj_0183.mrg', 'wsj_0184.mrg', 'wsj_0185.mrg', 'wsj_0186.mrg', 'wsj_0187.mrg', 'wsj_0188.mrg', 'wsj_0189.mrg', 'wsj_0190.mrg', 'wsj_0191.mrg', 'wsj_0192.mrg', 'wsj_0193.mrg', 'wsj_0194.mrg', 'wsj_0195.mrg', 'wsj_0196.mrg', 'wsj_0197.mrg', 'wsj_0198.mrg', 'wsj_0199.mrg']
```

Penn Treebank Data (Cont.)

C: \ nltk_data \ corpora \ treebank \ raw \ wjs_0001



```
wsj_0001 - Notepad
File Edit Format View Help
.START

Pierre Vinken, 61 years old, will join the board as a nonexecutive
director Nov. 29.
Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing
group.
```

Penn Treebank Data (Cont.)

- Tokenizing
- **sents()** 이용
- 문장의 token化한 결과

```
1 treebank.sents('wsj_0001.mrg')
```

```
[[ 'Pierre', 'Vinken', ',', ',', '61', 'years', 'old', ',', ',', 'will', 'join', 'the', 'board', 'as', 'a', 'nonexecutive', 'director', 'No', 'v.', '29', '.' ], [ 'Mr.', 'Vinken', 'is', 'chairman', 'of', 'Elsevier', 'N.V.', ',', ',', 'the', 'Dutch', 'publishing', 'group', '.' ] ]
```

Penn Treebank Data (Cont.)

- 원문 보기
- `join()`

```
1 wsj_0001 = treebank.sents('wsj_0001.mrg')
2 for line in wsj_0001:
3     print(' '.join(line))
```

Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group .

Penn Treebank Data (Cont.)

- 품사 Tagging
- **tagged_words()**

```
1 treebank.tagged_words('wsj_0001.mrg')
```

```
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ...]
```

Penn Treebank Data (Cont.)

- Data parsing 및 각 단어들의 상세 tagging
- `parsed_sents()`

```
1 treebank.parsed_sents('wsj_0001.mrg')[0]
```

The Ghostscript executable isn't found.

See <http://web.mit.edu/ghostscript/www/Install.htm>

If you're using a Mac, you can try installing

<https://docs.brew.sh/Installation> then `brew install ghostscript`

Tree('S', [Tree('NP-SBJ', [Tree('NP', [Tree('NNP', ['Pierre']), Tree('NNP', ['Vinken'])]), Tree(',', ['']), Tree('ADJP', [Tree('NP', [Tree('CD', ['61']), Tree('NNS', ['years'])]), Tree('JJ', ['old'])]), Tree(',', ['']), Tree('VP', [Tree('MD', ['will']), Tree('VP', [Tree('VB', ['join']), Tree('NP', [Tree('DT', ['the']), Tree('NN', ['board'])]), Tree('PP-CLR', [Tree('IN', ['as']), Tree('NP', [Tree('DT', ['a']), Tree('JJ', ['nonexecutive']), Tree('NN', ['director'])])]), Tree('NP-TMP', [Tree('NNP', ['Nov.']), Tree('CD', ['29'])])])])])]), Tree('.', [''])])])])

book Data

- `nltk.download('book')`
- NLTK 책에서 사용하는 모든 Data downloads.

```
import nltk
```

```
nltk.download('book', quiet=True)
```

```
from nltk.book import *
```

```
1 import nltk
2 nltk.download('book', quiet=True)
```

True

```
1 from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

book Data (Cont.)

- 허먼 멜빌(Herman Melville, 1819.8.1 ~ 1891.9.28, 미국)의 소설 Moby Dick

```
1 type(text1)
```

```
nltk.text.Text
```

```
1 text1
```

```
<Text: Moby Dick by Herman Melville 1851>
```

Gutenberg Data

- nltk의 corpus module 중 parsing이 안된 원문 데이터
- Gutenberg 말뭉치는 저작권이 말소된 문학작품.

```
1 nltk.download('gutenberg')
```

```
[nltk_data] Downloading package gutenberg to C:\nltk_data..  
[nltk_data] Package gutenberg is already up-to-date!
```

```
True
```

```
1 nltk.corpus.gutenberg.fileids()
```

```
['austen-emma.txt',  
'austen-persuasion.txt',  
'austen-sense.txt',  
'bible-kjv.txt',  
'blake-poems.txt',  
'bryant-stories.txt',  
'burgess-busterbrown.txt',  
'carroll-alice.txt',  
'chesterton-ball.txt',  
'chesterton-brown.txt',  
'chesterton-thursday.txt',  
'edgeworth-parents.txt',  
'melville-moby_dick.txt',  
'milton-paradise.txt',  
'shakespeare-caesar.txt',  
'shakespeare-hamlet.txt',  
'shakespeare-macbeth.txt',  
'whitman-leaves.txt']
```

Gutenberg Data (Cont.)

- 제인 오스틴(Jane Austen, 1775.12.16 ~ 1817.7.18, 영국의 소설가)의 소설 엠마(austen_emma.txt)

```
1 emma = nltk.corpus.gutenberg.raw('austen-emma.txt')
2 emma[:289]
```

```
'[Emma by Jane Austen 1816]#n#nVOLUME I#n#nCHAPTER I#n#n#nEmma Woodhouse, handsome, clever, and rich, with a comfortable home#nand
happy disposition, seemed to unite some of the best blessings#nof existence; and had lived nearly twenty-one years in the world#nwi
th very little to distress or vex her.'
```

한글 형태소 분석



형태소

■ 형태소

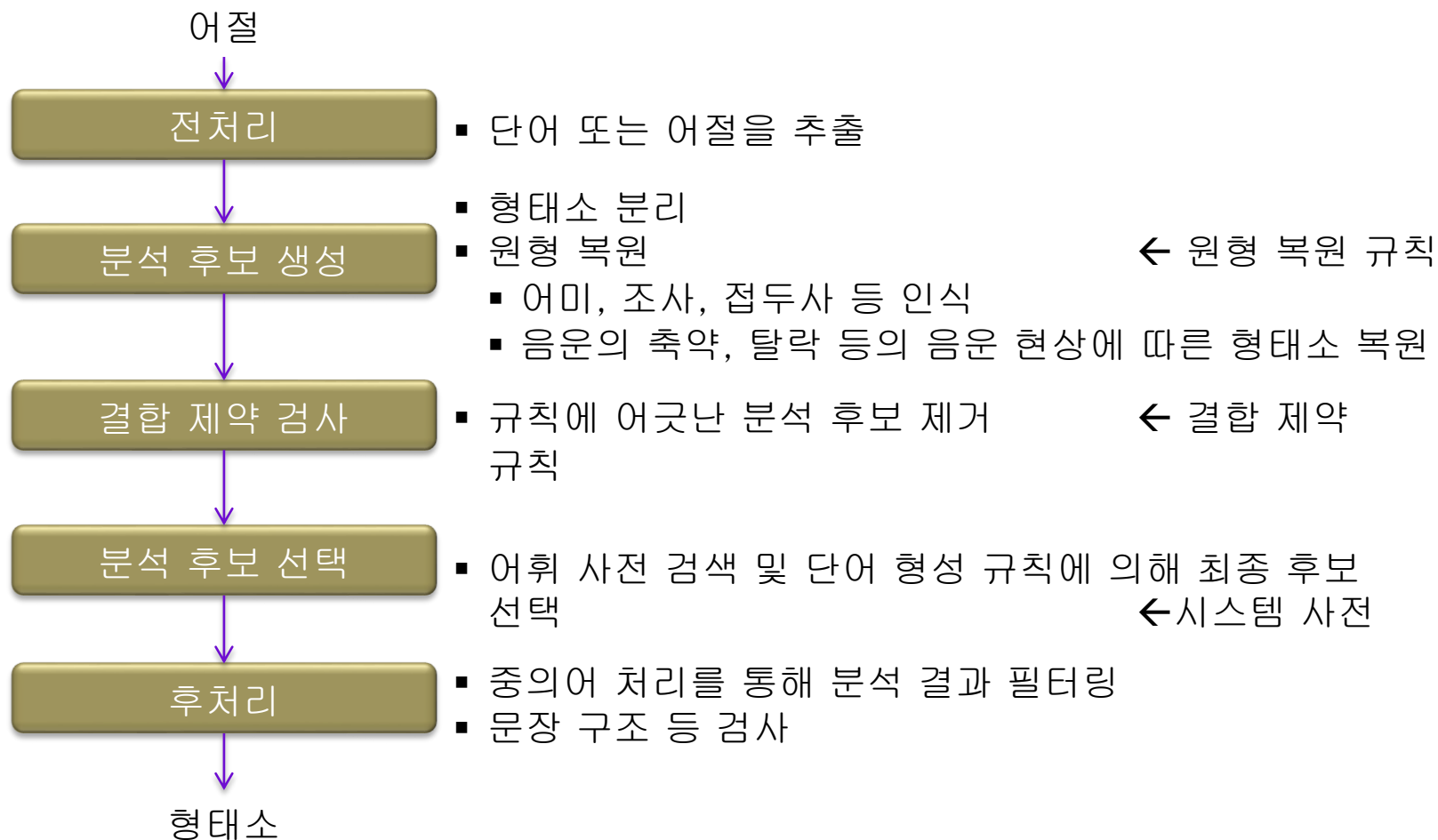
- 의미를 가진 최소의 단위
- 문법적, 관계적인 뜻을 나타내는 단어 또는 단어의 부분
- 체언 (명사, 대명사, 수사), 수식언 (관형사, 부사), 독립언 (감탄사), 관계언 (조사, 서술격조사), 용언 (동사, 형용사, 조동보조어간), 부속어 (어미, 접미)

■ 형태소의 종류

- 실질 형태소
 - 체언, 수식언, 감탄사, 용언의 어근
 - 구체적인 대상, 동작, 상태 등을 나타내는 형태소로 어휘 형태소
 - 검색 엔진에서 색인의 대상이 됩니다.
- 형식 형태소
 - 조사, 어말어미, 접미사, 접두사, 선어말어미
 - 형태소 간의 관계를 나타내는 형태소로 문법 형태소

형태소 분석

- 단어 또는 어절을 구성하는 각 형태소 분리
- 분리된 형태소의 기본형(어근) 및 품사 정보 추출
- 일반적인 형태소 분석 절차



형태소 분석 엔진

■ KoNLPy

- Python용 형태소 분석기(Korean NLP in Python)(GPL v3+License)
- 내부적으로 각기 다른 언어(Java, C++ 등)로 호환성에 문제가 생기기도
- 일부 Library 오래 전 제작/업데이트
- Java로 구현되어 있어서 C++로 구현된 Mecab에 속도에서 불리.
- 사용하기 용이하고 다양한 Library들 사용의 장점도.
- <http://konlpy.readthedocs.org/>

■ Mecab

- 한국어 형태소 분석에서 가장 많이 사용
- 원래 일본어 형태소 분석용 Open Source
- <https://bitbucket.org/eunjeon/mecab-ko-dic/src/master/>

■ KOMORAN

- Java로 만든 오픈소스 형태소 분석기(Apache v2 License)
- <https://shineware.tistory.com/tag/KOMORAN/>

형태소 분석 엔진 (Cont.)

■ HanNanum

- Java로 만들어진 형태소 분석기(GPL v3 License)
- <http://semanticweb.kaist.ac.kr/home/index.php/HanNanum>

■ KoNLP

- R용 형태소 분석기(Korean NLP)(GPL v3 License)
- <https://github.com/haven-jeon/KoNLP>

■ Khaiii

- Kakao Hangul Analyzer III
- 세종 Corpus를 이용해서 CNN 기술을 적용한 형태소 분석기
- C++/Python으로 구현
- <https://tech.kakao.com/2018/12/13/khaiii/>

KoNLPy Package

- 한국어 정보처리를 위한 파이썬 패키지
- Open Source Software이며 [GPL v3 이상] 라이선스로 배포
- 공식 사이트는 다음과 같습니다.
 - <http://konlpy.org/en/latest/>
 - <https://github.com/konlpy/konlpy>
- KoNLPy 패키지 설치
 - KoNLPy는 JPytype1 패키지에 의존
 - (base) C:\Users\COM>pip install konlpy
- JDK를 설치해야 함
 - <http://jdk.java.net/12/> -> Builds -> Windows/x64 zip 다운로드
 - 압축 풀기
 - JAVA_HOME 환경변수 설정

KoNLPy Package (Cont.)

- API

- <https://konlpy-ko.readthedocs.io/ko/v0.4.3/api/konlpy.tag/>

- Hannum Class

- Kkma Class

- Komoran Class

- Mecab Class

- Okt(Open Korean Text, Twitter가 0.5.0부터 Okt로 변경됨) Class

- Korean POS tags comparison chart

- https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0

형태소 분석

- 형태소를 비롯하여, 어근, 접두사/접미사, 품사(POS, part-of-speech) 등 다양한 언어적 속성의 구조를 파악하는 것
- 품사 태그를 알아야 함
 - <https://konlpy-ko.readthedocs.io/ko/v0.4.3/morph/#comparison-between-pos-tagging-classes>

형태소 관련 용어 번역

영어	한국어	예시
adverb	부사	매우
adjective	형용사	예쁘다
conjunction	접속사	그리고
determiner/prenoun	관형사	새-
noun	명사	아이유
pronoun	대명사	그녀
verb	동사	달리다
exclamations/interjections	감탄사	오!
particle/postposition	조사	이/가
preposition	전치사 (영어)	between

품사 Tagging

- 형태소의 뜻과 문맥을 고려하여 그것에 Markup을 하는 일
- 단어와 /(슬래시) 뒤에 품사가 표시되어 나누어지도록 하는 것
- 예)
 - 원문 : 가방에 들어가신다
 - 품사 태깅 : 가방/NNG + 예/JKM + 들어가/VV + 시/EPH + 다/EFN

Mecab

- 한국어 분절에 가장 많이 사용.
- Yet Another Part-of-Speech and Morphological Analyzer
- 원래 일본어 형태소 분석용 Open Source였으나, 이를 한국어 형태소 분석기로 적용.
- <http://taku910.github.io/mecab/>
- API
 - <https://konlpy-ko.readthedocs.io/ko/v0.4.3/api/konlpy.tag/#mecab-class>

Mecab (Cont.)

```
konlpy.tag._mecab.Mecab(dicpath='/usr/local/lib/mecab/dic/mecab-ko-dic')
```

메서드	설명
morphs(phrase)	형태소 분석 문구를 반환.
nouns(phrase)	명사를 추출.
pos(phrase, flatten=True)	flatten이 False이면 어절을 보존.

mecab-ko-dic

- Open Source 형태소 분석 engine인 MeCab을 사용하여, 한국어 형태소 분석을 하기 위한 프로그램.
- Apache License v. 2.0
- <https://bitbucket.org/eunjeon/mecab-ko-dic/src/master/>
- mecab-ko-dic에서 사용한 사전형식과 품사 tag
 - <https://docs.google.com/spreadsheets/d/1-9blXKjtjeKZqsf4NzHeYJCrr49-nXeRF6D80udfcwY/edit#gid=589544265>

mecab-ko-dic (Cont.)

```
instructor@MyDesktop:~$ echo "안녕하세요, 반갑습니다!" | mecab
안녕      NNG,행위,T,안녕,*,*,*,*
하        XSV,*,F,하,*,*,*,*
세요      EP+EF,*,F,세요,Inflect,EP,EF,시/EP/*+어요/EF/*
,         SC,*,*,*,*,*,*,*
반갑      VA,*,T,반갑,*,*,*,*
습니다    EF,*,F,습니다,*,*,*,*
!         SF,*,*,*,*,*,*,*
EOS
```

Mecab (Cont.)

```
from konlpy.tag import Mecab  
mecab = Mecab()
```

```
text = u"""아름답지만 다소 복잡하기도한 한국어는 전세계에서 13번째로 많이 사용되는 언어입니다."""
```

```
mecab.morphs(text)
```

```
['아름답',  
'지만',  
'다소',  
'복잡',  
'하',  
'기',  
'도',  
'한',  
'한국어',  
'는',  
'전',  
'세계',  
'에서',  
'13',  
'번',  
'째',  
'로',  
'많이',  
'사용',  
'되',  
'는',  
'언어',  
'입니다',  
'']
```

Mecab (Cont.)

```
mecab.nouns(text)
```

```
['한국어', '세계', '번', '사용', '언어']
```

```
mecab.pos(text)
```

```
[('아름답', 'YA'),  
 ('지만', 'EC'),  
 ('다소', 'MAG'),  
 ('목잡', 'XR'),  
 ('하', 'XSA'),  
 ('기', 'ETN'),  
 ('도', 'JX'),  
 ('한', 'MM'),  
 ('한국어', 'NNG'),  
 ('는', 'JX'),  
 ('전', 'MM'),  
 ('세계', 'NNG'),  
 ('에서', 'JKB'),  
 ('13', 'SN'),  
 ('번', 'NNBC'),  
 ('째', 'XSN'),  
 ('로', 'JKB'),  
 ('많이', 'MAG'),  
 ('사용', 'NNG'),  
 ('되', 'XSV'),  
 ('는', 'ETM'),  
 ('언어', 'NNG'),  
 ('입니다', 'VCP+EF'),  
 ('.', 'SF')]
```

Komoran

- 2013년 Shineware에서 Java로 만든 오픈소스 한국어 형태소 분석기

`konlpy.tag.Komoran(jvmpath=None, dicpath=None)`

메서드	설명
<code>morphs(phrase)</code>	형태소 분석 문구를 반환.
<code>nouns(phrase)</code>	명사를 추출.
<code>pos(phrase, flatten=True)</code>	<code>flatten</code> 이 <code>False</code> 이면 어절을 보존.

Komoran (Cont.)

■ Komoran 형태소 분석기를 이용해 Tagging하는 예

```
1 text = u"""아름답지만 다소 복잡하기도한 한국어는 전세계에서 13번째로 많이 사용되는 언어입니다."""
```

```
1 from konlpy.tag import Komoran  
2 komoran = Komoran()
```

```
1 print(komoran.morphs(text))
```

['아름답', '지만', '다소', '복잡', '하', '기', '도', '하', 'ㄴ', '한국어', '는', '전', '세계', '에서', '13', '번', '째', '로', '많이', '사용', '되', '는', '언어', '이', 'ㅂ니다', '.']

```
1 print(komoran.nouns(text))
```

['한국어', '전', '세계', '번', '사용', '언어']

```
1 print(komoran.pos(text))
```

[('아름답', 'VA'), ('지만', 'EC'), ('다소', 'MAG'), ('복잡', 'XR'), ('하', 'XSA'), ('기', 'ETN'), ('도', 'JX'), ('하', 'VV'), ('ㄴ', 'ETM'), ('한국어', 'NNP'), ('는', 'JX'), ('전', 'NNG'), ('세계', 'NNG'), ('에서', 'JKB'), ('13', 'SN'), ('번', 'NNB'), ('째', 'XSN'), ('로', 'JKB'), ('많이', 'MAG'), ('사용', 'NNG'), ('되', 'XSV'), ('는', 'ETM'), ('언어', 'NNG'), ('이', 'VCP'), ('ㅂ니다', 'EF'), ('.', 'SF')]

Okt

■ 기존의 Twitter가 0.5.0 부터 이름 변경됨.

```
1 text = u"""아름답지만 다소 복잡하기도한 한국어는 전세계에서 13번째로 많이 사용되는 언어입니다."""
```

```
1 from konlpy.tag import Okt
2 okt=Okt()
3 print(okt.morphs(text))
```

['아름답지만', '다소', '복잡하', '기도', '한', '한국어', '는', '전세계', '에서', '13', '번', '째', '로', '많이', '사용', '되는', '언어', '입니다', '.']

```
1 print(okt.pos(text))
```

[('아름답지만', 'Adjective'), ('다소', 'Noun'), ('복잡하', 'Adjective'), ('기도', 'Noun'), ('한', 'Josa'), ('한국어', 'Noun'), ('는', 'Josa'), ('전세계', 'Noun'), ('에서', 'Josa'), ('13', 'Number'), ('번', 'Noun'), ('째', 'Suffix'), ('로', 'Josa'), ('많이', 'Adverb'), ('사용', 'Noun'), ('되는', 'Verb'), ('언어', 'Noun'), ('입니다', 'Adjective'), ('.', 'Punctuation')]

```
1 print(okt.nouns(text))
```

['다소', '기도', '한국어', '전세계', '번', '사용', '언어']

Kkma

- 서울대학교 IDS(Intelligent Data Systems) 연구실에서 자연어 처리를 위한 다양한 모듈 및 자료를 구축하기 위한 과제.

```
1 text = u"""아름답지만 다소 복잡하기도한 한국어는 전세계에서 13번째로 많이 사용되는 언어입니다."""
```

```
1 from konlpy.tag import Kkma
2 kkma = Kkma()
3 print(kkma.morphs(text))
```

['아름답', '지만', '다소', '복잡', '하', '기', '도', '한', '한국어', '는', '전세계', '에서', '13', '번째', '로', '많이', '사용', '되', '는', '언어', '이', '입니다', '.']

```
1 print(kkma.pos(text))
```

[('아름답', 'VA'), ('지만', 'ECE'), ('다소', 'MAG'), ('복잡', 'NNG'), ('하', 'XSV'), ('기', 'ETN'), ('도', 'JX'), ('한', 'MDN'), ('한국어', 'NNG'), ('는', 'JX'), ('전세계', 'NNG'), ('에서', 'JKM'), ('13', 'NR'), ('번째', 'NNB'), ('로', 'JKM'), ('많이', 'MAG'), ('사용', 'NNG'), ('되', 'XSV'), ('는', 'ETD'), ('언어', 'NNG'), ('이', 'VCP'), ('입니다', 'EFN'), ('.', 'SF')]

```
1 print(kkma.nouns(text))
```

['복잡', '한국어', '전세계', '13', '13번째', '번째', '사용', '언어']

Hannanum

- 1999년 KAIST SWRC(Semantic Web Research Center)에 의해 개발된 Java로 만들어진 형태소 분석기

`konlpy.tag.Hannanum(jvmpath = None)`

메소드	설명
<code>analyze(phrase)</code>	이 분석기는 각 토큰에 대한 다양한 형태 학적후보를 반환합니다.
<code>morphs(phrase)</code>	형태소 분석 문구를 반환합니다.
<code>nouns(phrase)</code>	명사를 추출합니다.
<code>pos(phrase, ntags=9, flatten=True)</code>	ntags는 태그의 수. 9(한 문자)또는 22(두 문자)일 수 있습니다. flatten이 False이면 어절을 보존합니다.

한국어 Tokenizing & POS Tagging Exercise

```
1 text = u"""하지만 한국어는 영어와는 달리 띄어쓰기만으로는 토큰화를 하기에 부족합니다."""
```

```
1 from konlpy.tag import Okt
2 okt=Okt()
3 print(okt.morphs(text))
```

['하지만', '한국어', '는', '영어', '와는', '달리', '띄어쓰기', '만으로는', '토큰', '화', '를', '하기에', '부족합니다', '.']

```
1 print(okt.pos(text))
```

[('하지만', 'Conjunction'), ('한국어', 'Noun'), ('는', 'Josa'), ('영어', 'Noun'), ('와는', 'Josa'), ('달리', 'Noun'), ('띄어쓰기', 'Noun'), ('만으로는', 'Josa'), ('토큰', 'Noun'), ('화', 'Suffix'), ('를', 'Josa'), ('하기에', 'Verb'), ('부족합니다', 'Adjective'), ('.', 'Punctuation')]

```
1 print(okt.nouns(text))
```

['한국어', '영어', '달리', '띄어쓰기', '토큰']

말뭉치

- 컴퓨터를 이용해 자연어 분석 작업을 할 수 있도록 만든 문서 집합
- KoNLPy 패키지를 설치하고 사용할 수 있는 말뭉치
 - kolaw
 - 한국 법률 말뭉치
 - constitution.txt
 - kobill
 - 대한민국 국회 의안 말뭉치
 - 파일 ID는 의안 번호를 의미
 - 1809890.txt - 1809899.txt

말뭉치 (Cont.)

```
1 from konlpy.corpus import kolaw
2 c = kolaw.open('constitution.txt').read()
3 print(c[:100])
```

대한민국헌법

유구한 역사와 전통에 빛나는 우리 대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에 항거한 4·19민주이념을 계승하고, 조국의 민주개혁과 평화적 통일의

```
1 from konlpy.corpus import kobill
2 d = kobill.open('1809890.txt').read()
3 print(d[150:300])
```

초등학교 저학년의 경우에도 부모의 따뜻한 사랑과 보살핌이 필요한 나이이나, 현재 공무원이 자녀를 양육하기 위하여 육아휴직을 할 수 있는 자녀의 나이는 만 6세 이하로 되어 있어 초등학교 저학년인 자녀를 돌보기 위해서는 해당 부모님은 일자리를 그만 두어야



Text Preprocessing **Tokenization**



Tokenization

- 자연어 분석을 위해 문자열을 작은 단위(token)로 나누기
- Token → 나눈 작은 문자열 단위
- 단어 토큰화(Word Tokenization)
- 토큰화시 고려사항
 - 구두점이나 특수 문자를 단순 제외는 안됨.
 - 줄임말과 단어 내에 띄어쓰기가 있는 경우
- 문장 토큰화(Sentence Tokenization)
- 이진 분류기(Binary Classifier)

Tokenization (Cont.)

- 단어 토큰화(Word Tokenization)
- Token의 기준이 단어(Word)인 경우
- 단어(Word)외에 단어구, 의미를 갖는 문자열도 포함.
- **word_tokenize()**

```
1 from nltk.tokenize import word_tokenize
2 word_tokenize(emma[50:100])
```

```
['Emma',
 'Woodhouse',
 ',',
 'handsome',
 ',',
 'clever',
 ',',
 'and',
 'rich',
 ',',
 'with',
 'a']
```

Tokenization (Cont.)

■ `word_tokenize()` 사용시

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

```
1 import nltk
2 from nltk.tokenize import word_tokenize
3 print(word_tokenize(text))
```

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


Tokenization (Cont.)

■ WordPunctTokenizer 사용시

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

```
1 import nltk
2 from nltk.tokenize import WordPunctTokenizer
3 WordPunctTokenizer().tokenize(text)
```

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

Tokenizing (Cont.)

- Token化에 포함할 문자들을 정규표현식을 이용해서 지정하기
- **RegexTokenizer()**

```
1 from nltk.tokenize import RegexTokenizer
2 retokenize = RegexTokenizer('[\W]+')
3 retokenize.tokenize(emma[50:100])
```

```
['Emma', 'Woodhouse', 'handsome', 'clever', 'and', 'rich', 'with', 'a']
```

Tokenizing시 고려할 사항

- 구두점이나 특수 문자를 단순 제외해서는 안된다.
- 구두점(.)은 문자의 경계를 설명하기 때문에 구두점 제외에 고려해야.
- 단어 자체가 갖고 있는 구두점
 - . Ph.D
 - \$ \$45.44
 - / 01/02/06
 - & AT&T
 - , 123,456,789

Tokenizing시 고려할 사항 (Cont.)

- 줄임말과 단어 내에 띄어쓰기가 있는 경우
- 줄임말
 - ' I'm what're we're
- 단어 자체의 띄어쓰기 내제
 - New Work
 - rock 'n' roll

Tokenizing시 고려할 사항 (Cont.)

```
1 text = """Starting a home-based restaurant may be an ideal. it doesn't have a food
2     chain or restaurant of their own."""
```

```
1 import nltk
2 from nltk.tokenize import TreebankWordTokenizer
3 tokenizer=TreebankWordTokenizer()
4 tokenizer.tokenize(text)
```

```
['Starting',
 'a',
 'home-based',
 'restaurant',
 'may',
 'be',
 'an',
 'ideal.',
 'it',
 'does',
 'n't',
 'have',
 'a',
 'food',
 'chain',
 'or',
 'restaurant',
 'of',
 'their',
 'own',
 '.']
```

Sentence Tokenization

- Token의 단위가 문장일 때
- NLTK에서는 `sent_tokenize()` 지원

```
1 from nltk.tokenize import sent_tokenize
2 sent_tokenize(emma[:1000])[3]
```

"Sixteen years had Miss Taylor been in Mr. Woodhouse's family,¶nless as a governess than a friend, very fond of both daughters,¶nbu
t particularly of Emma."

Sentence Tokenization (Cont.)

- Token의 단위가 문장일 때
- NLTK에서는 `sent_tokenize()` 지원

```
1 from nltk.tokenize import sent_tokenize
2 sent_tokenize(emma[:1000])[3]
```

"Sixteen years had Miss Taylor been in Mr. Woodhouse's family, #nless as a governess than a friend, very fond of both daughters, #nbu
t particularly of Emma."

```
1 sentence = """His barber kept his word. But keeping such a huge secret to himself was driving him crazy.
2           Finally, the barber went up a mountain and almost to the edge of a cliff.
3           He dug a hole in the midst of some reeds. He looked about, to mae sure no one was near."""
```

```
1 import nltk
2 from nltk.tokenize import sent_tokenize
3 print(sent_tokenize(sentence))
```

['His barber kept his word.', 'But keeping such a huge secret to himself was driving him crazy.', 'Finally, the barber went up a mo
untain and almost to the edge of a cliff.', 'He dug a hole in the midst of some reeds.', 'He looked about, to mae sure no one was n
ear.']

Sentence Tokenization (Cont.)

- 구두점이 여러 개 있을 경우

```
1 import nltk
2 from nltk.tokenize import sent_tokenize
3 text="I am actively looking for Ph.D. students. and you are a Ph.D student."
4 print(sent_tokenize(text))
```

['I am actively looking for Ph.D students.', 'and you are a Ph.D student.']

영문 Tokenizing & POS Tagging Exercise

```
1 text="I am actively looking for Ph.D. students. and you are a Ph.D. student."
```

```
1 import nltk
2 from nltk.tokenize import word_tokenize
3 print(word_tokenize(text))
```

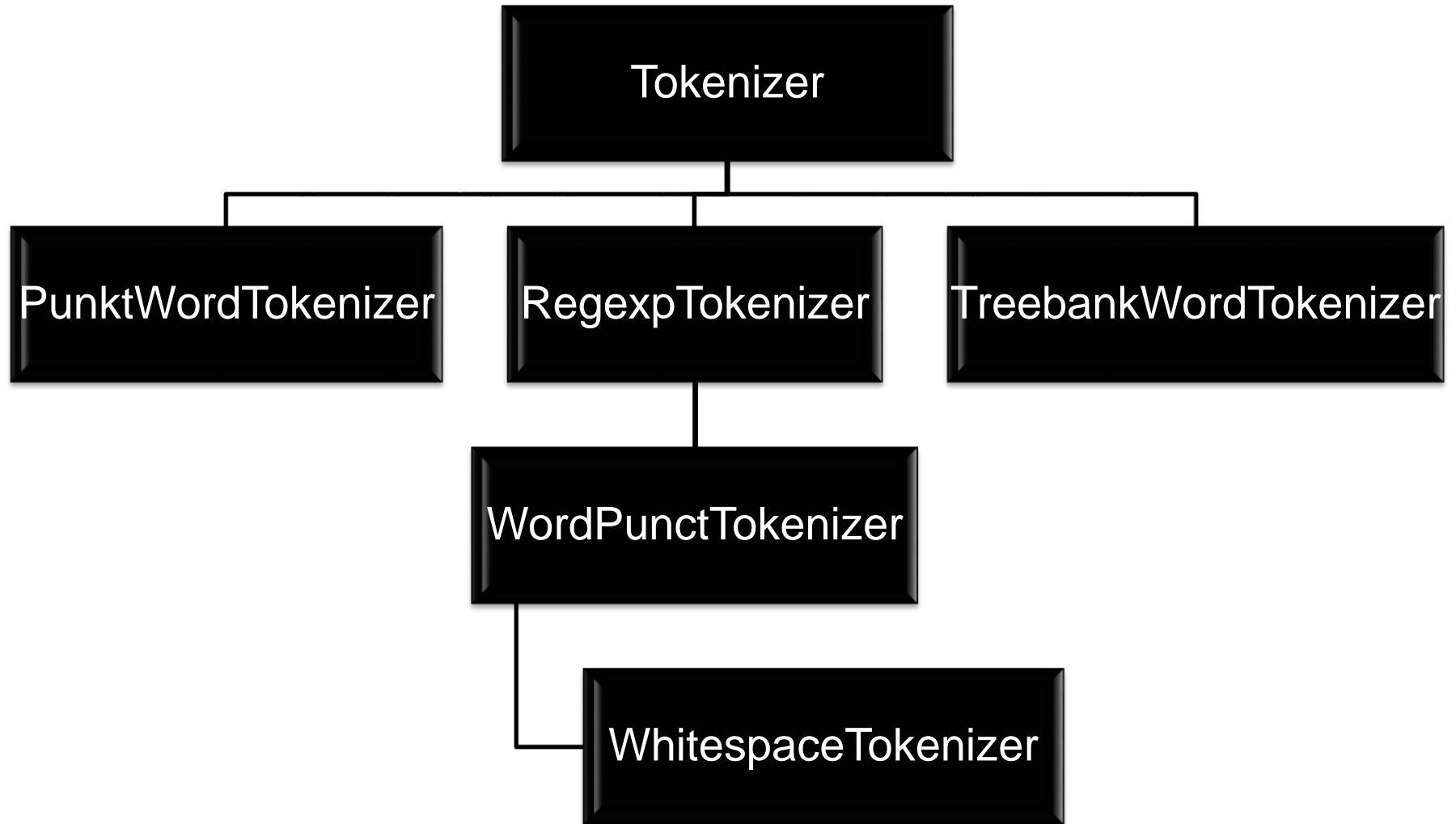
```
['I', 'am', 'actively', 'looking', 'for', 'Ph.D.', 'students', '.', 'and', 'you', 'are', 'a', 'Ph.D.', 'student', '.']
```

```
1 from nltk.tag import pos_tag
2 x=word_tokenize(text)
3 pos_tag(x)
```

```
[('I', 'PRP'),
 ('am', 'VBP'),
 ('actively', 'RB'),
 ('looking', 'VBG'),
 ('for', 'IN'),
 ('Ph.D.', 'NNP'),
 ('students', 'NNS'),
 ('.', '.'),
 ('and', 'CC'),
 ('you', 'PRP'),
 ('are', 'VBP'),
 ('a', 'DT'),
 ('Ph.D.', 'NNP'),
 ('student', 'NN'),
 ('.', '.')]

```

Tokenizer의 상속 Tree



이진 분류기(Binary Classifier)

- 문장 토큰화에서의 예외 사항을 발생시키는 구두점의 처리를 위해 입력에 따라 두 개의 Class로 분류하기도.
- Class
 - Period가 단어의 일부분일 경우, 약어(Abbreviation)인 경우
 - Period가 정말로 문장의 구분자(Boundary)인 경우
- 영어권에서 사용하는 약어 사전(Abbreviation Dictionary)
 - <https://public.oed.com/how-to-use-the-oed/abbreviations/>
- Open Source
 - NLTK, OpenNLP, Stanford CoreNLP, splitta, LingPipe
- 참고 자료
 - <https://tech.grammarly.com/blog/posts/How-to-Split-Sentences.html>

한국어에서의 Tokenization의 어려움

- 한국어는 교착어이다.
 - 영어의 띄어쓰기와 달리 한국어는 조사가 붙어있음.
 - 조사 분리 필요

종류	대표적 언어	특징
교착어	한국어, 일본어, 몽골어	어간에 접사가 붙어 단어를 이루고 의미와 문법적 기능이 정해짐
굴절어	라틴어, 독일어, 러시아어	단어의 형태가 변함으로써 문법적 기능이 정해짐
고립어	영어, 중국어	어순에 따라 단어의 문법적 기능이 정해짐

한국어에서의 Tokenization의 어려움

- 한국어는 띄어쓰기가 영어보다 잘 지켜지지 않는다.
 - 원래 한국어는 띄어쓰기가 없었음.
 - 1933년 한글맞춤법통일안 이후
- 평서문과 의문문
 - 영어와 달리 한국어는 의문문과 평서문이 같은 형태의 문장 구조를 가진다.
 - 물음표나 마침표가 붙지 않으면 잘 알기 어렵다.

언어	평서문	의문문
영어	I ate my lunch.	Did you have lunch?
한국어	점심 먹었어.	점심 먹었어?

한국어에서의 Tokenization의 어려움

■ 주어 생략

- 영어와 달리 주어가 생략되는 경우가 많음.
- 번역시 어려움.

■ 한자 기반 언어

- 한자의 영향으로 한자의 조합으로 이루어지는 단어들이 많음.
- 단어가 조합되어야 하나의 의미로 인식.

언어	단어	조합
영어	concentrate	con(=together) + centr(=center) + ate(=make)
한국어	집중(集中)	集(모을 집) + 中(가운데 중)

형태소 분석(Morphological Analysis)

■ 형태소

- 언어학에서 일정한 의미가 있는 가장 작은 말 단위.

■ 형태소 분석

- 단어로부터 어근, 접두사, 접미사, 품사 등 다양한 언어적 속성을 파악하고 이를 이용하여 형태소를 찾아내거나 처리하는 작업.

■ 어간 추출(Stemming)

■ 표제어 추출(Lemmatizing)

■ 품사 부착(Part-Of-Speech Tagging)



Text Preprocessing

Cleaning and Normalization



정제(Cleaning)과 정규화(Normalization)

- Token화 작업 전, 후에 Data를 용도에 맞게 정제 및 정규화 필요
- 정제
 - 갖고 있는 Corpus로부터 Noise Data 제거
- 정규화
 - 표현 방법이 다른 단어들을 통합시켜서 같은 단어로 만듦.
- 규칙에 기반한 표기가 다른 단어들의 통합
 - USA와 US
- 대소문자 통합
- 불필요한 단어 제거

불필요한 단어의 제거

■ Noise Data

- 정제 작업시 자연어가 아니면서 아무 의미도 갖지 않는 글자들(예, 특수문자)
- 분석하고자 하는 목적에 맞지 않는 불필요한 단어들.

■ 방법

- 불용어(Stopword) 제거
- 등장 빈도가 적은 단어 제거
- 길이가 짧은 단어 제거

```
1 # 길이가 1~2인 단어들을 정규 표현식을 이용하여 삭제
2 import re
3 text = "I was wondering if anyone out there could enlighten me on this car."
4 shortword = re.compile(r'\\w*\\b\\w{1,2}\\b')
5 print(shortword.sub('', text))
```

was wondering anyone out there could enlighten this car.



Text Preprocessing

Stemming & Lemmatization



Stemming

- 변화된 단어에서 접사(affix)를 제거하여 같은 의미를 가지는 형태소의 기본형을 찾는 방법.
- NLTK에서 제공하는 Class
 - PorterStemmer
 - LancasterStemmer
 - RegexpStemmer
 - SnowballStemmer
- 어간 추출법(Stemming)은 단순히 어미를 제거할 뿐이므로 단어의 원형을 정확히 찾아주지는 않는다.

PorterStemmer Class

- Martin Porter의 Porter Stemming Algorithm 구현
- 영어의 접미사(suffix) 제거

```
words = ['sending', 'cooking', 'flies', 'lives', 'crying', 'dying']
```

```
from nltk.stem import PorterStemmer  
ps = PorterStemmer()  
[ps.stem(word) for word in words]
```

```
['send', 'cook', 'fli', 'live', 'cri', 'die']
```

LancasterStemmer Class

- Lancaster 대학이 개발한 Algorithm을 구현.

```
words = ['sending', 'cooking', 'flies', 'lives', 'crying', 'dying']
```

```
from nltk.stem import LancasterStemmer  
ls = LancasterStemmer()  
[ls.stem(word) for word in words]
```

```
['send', 'cook', 'fli', 'liv', 'cry', 'dying']
```

RegexStemmer Class

- 정규 표현식이용해서 어간 추출

```
words = ['sending', 'cooking', 'flies', 'lives', 'crying', 'dying']
```

```
from nltk.stem.regex import RegexStemmer  
rs = RegexStemmer(r'ing')  
[rs.stem(word) for word in words]
```

```
['send', 'cook', 'flies', 'lives', 'cry', 'dy']
```

SnowballStemmer Class

- 영어 포함 15개국 언어에 대한 어간 추출을 지원.

```
words = ['enviar', 'cocina', 'moscas', 'vidas', 'llorar', 'morir']
```

```
from nltk.stem import SnowballStemmer  
ss = SnowballStemmer('spanish')  
[ss.stem(word) for word in words]
```

```
['envi', 'cocin', 'mosc', 'vid', 'llor', 'mor']
```


표제어 추출(Lemmatizing)

- 표제어 추출은 어간 추출(Stemming)과 비슷하지만 동의어 대체와 더 유사하다.
- 어간 추출과 달리 원형 복원 후에도 그 단어는 같은 의미를 가진다.
- 품사 지정시 더 정확히 원형을 찾을 수 있다.
- NLTK에서는 WordNetLammatizer 제공

WordNetLemmatizer

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

```
1 import nltk
2 from nltk.stem import WordNetLemmatizer
3 wnl = WordNetLemmatizer()
4 [wnl.lemmatize(word) for word in words]
```

```
['policy',
 'doing',
 'organization',
 'have',
 'going',
 'love',
 'life',
 'fly',
 'dy',
 'watched',
 'ha',
 'starting']
```

한국어 Stemming

- 한국어는 5언 9품사의 구조임.

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



Text Preprocessing

Stopword



불용어(Stopword)

- 큰 의미가 없는 단어
- 문장 내에서 자주 등장하지만 문장 분석에 큰 도움이 되지 않는 단어
- I, my, me, over, 조사, 접미사 등.
- NLTK에서는 100여개 이상의 영어 단어들 등록

```
1 import nltk
2 from nltk.corpus import stopwords
3 stopwords.words('english')[:20]
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his']
```

불용어(Stopword) (Cont.)

```
1 import nltk
2 from nltk.corpus import stopwords
3 from nltk.tokenize import word_tokenize
4
5 example = "Family is not an important thing. It's everything."
6 stop_words = set(stopwords.words('english'))
7
8 word_tokens = word_tokenize(example)
9
10 result = []
11 for w in word_tokens:
12     if w not in stop_words:
13         result.append(w)
14
15 print(word_tokens)
16 print(result)
```

['Family', 'is', 'not', 'an', 'important', 'thing', '.', 'It', "'s", 'everything', '.']

['Family', 'important', 'thing', '.', 'It', "'s", 'everything', '.']

불용어(Stopword)

- 큰 의미가 없는 단어
- 문장 내에서 자주 등장하지만 문장 분석에 큰 도움이 되지 않는 단어
- 보통 형태소 분석 후, 조사 / 접속사 등을 제거하는 방법
- 현업에서 사용자가 직접 불용어 사전을 만들어 사용
- 보편적으로 선택할 수 있는 한국어 불용어 리스트
 - <https://www.ranks.nl/stopwords/korean>
- 한국어용 불용어를 제거하는 가장 좋은 방법은 코드 내에서 직접 정의하지 않고 txt 파일이나 csv 파일로 수많은 불용어를 정리해놓고, 이를 불러와서 사용하는 방법 추천.

불용어(Stopword) (Cont.)

```
1 import nltk
2 from nltk.corpus import stopwords
3 from nltk.tokenize import word_tokenize
4
5 example = "고기를 아무렇게나 구우려고 하면 안 돼. 고기라고 다 같은 게 아니거든. 예컨대 삼겹살을 구울 때는 중요한 게 있지."
6 stop_words = "아무거나 아무렇게나 어찌하든지 같다 비슷하다 예컨대 이럴정도로 하면 아니거든"
7
8 stop_words=stop_words.split(' ')
9
10 word_tokens = word_tokenize(example)
11
12 result = []
13 for w in word_tokens:
14     if w not in stop_words:
15         result.append(w)
16 # 위의 4줄 코드는 아래의 한 줄로 대체 가능
17 # result=[word for word in word_tokens if not word in stop_words]
18
19 print(word_tokens)
20 print(result)
```

['고기를', '아무렇게나', '구우려고', '하면', '안', '돼', '.', '고기라고', '다', '같은', '게', '아니거든', '.', '예컨대', '삼겹살', '을', '구울', '때는', '중요한', '게', '있지', '.']

['고기를', '구우려고', '안', '돼', '.', '고기라고', '다', '같은', '게', '.', '삼겹살을', '구울', '때는', '중요한', '게', '있지', '.']



Text Preprocessing

Splitting Data





Text Preprocessing

Integer Encoding



정수(Integer) Encoding

- 자연어 처리에서 Text를 숫자로 바꾸는 여러 가지 기법들 중 하나.
- 또는 그런 기법들을 본격적으로 적용시키기 위한 첫 단계로 각 단어를 고유한 숫자에 Mapping시키는 전처리 작업.
- 갖고 있는 Text에 단어가 5,000개가 있다면, 5,000개의 단어들 각각에 0번부터 4,999번까지 단어와 Mapping되는 고유한 숫자 즉 Index를 부여하는 방법.
- Index를 부여하는 방법
 - Random으로
 - 보통은 전처리도 같이 검하기 위해 단어에 대한 빈도수로 정렬한 뒤 부여.

정수(Integer) Encoding (Cont.)

- 자연어 처리에서 Text를 숫자로 바꾸는 여러 가지 기법들 중 하나.
- 또는 그런 기법들을 본격적으로 적용시키기 위한 첫 단계로 각 단어를 고유한 숫자에 Mapping시키는 전처리 작업.
- 갖고 있는 Text에 단어가 5,000개가 있다면, 5,000개의 단어들 각각에 0번부터 4,999번까지 단어와 Mapping되는 고유한 숫자 즉 Index를 부여하는 방법.
- Index를 부여하는 방법
 - Random으로
 - 보통은 전처리도 같이 검하기 위해 단어에 대한 빈도수로 정렬한 뒤 부여.

정수(Integer) Encoding (Cont.)

■ 빈도수 순서대로 단어를 정수로 변환

```
1 text="""A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret!  
2     The Secret He Kept is huge secret. Huge secret. His barber kept his word. a barber kept  
3     his word. His barber kept his secret. But keeping and keeping such a huge secret to himself  
4     was driving the barber crazy. the barber went up a huge mountain."""
```

```
1 # 문장 토큰화  
2 from nltk.tokenize import sent_tokenize  
3 text = sent_tokenize(text)  
4 print(text)
```

```
['A barber is a person.', 'a barber is good person.', 'a barber is huge person.', 'he Knew A Secret!', 'The Secret He Kept is huge  
secret.', 'Huge secret.', 'His barber kept his word.', 'a barber kept his word.', 'His barber kept his secret.', 'But kee  
ping and keeping such a huge secret to himself', 'was driving the barber crazy.', 'the barber went up a huge mountain.']
```

정수(Integer) Encoding (Cont.)

■ 빈도수 순서대로 단어를 정수로 변환

Step1. 문장 Tokening

```
1 text="""A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret!  
2     The Secret He Kept is huge secret. Huge secret. His barber kept his word. a barber kept  
3     his word. His barber kept his secret. But keeping and keeping such a huge secret to himself  
4     was driving the barber crazy. the barber went up a huge mountain."""
```

```
1 # 문장 토큰화  
2 from nltk.tokenize import sent_tokenize  
3 text = sent_tokenize(text)  
4 print(text)
```

```
['A barber is a person.', 'a barber is good person.', 'a barber is huge person.', 'he Knew A Secret!', 'The Secret He Kept is huge  
secret.', 'Huge secret.', 'His barber kept his word.', 'a barber kept his word.', 'His barber kept his secret.', 'But keep  
ing and keeping such a huge secret to himself', 'was driving the barber crazy.', 'the barber went up a huge mountain.']
```

정수(Integer) Encoding (Cont.)

Step2. 정제(Cleaning), 단어 Tokenizing, 각 단어별 빈도수 계산

```
1 from nltk.tokenize import word_tokenize
2 from nltk.corpus import stopwords
3 from collections import Counter
4 vocab = Counter() # Python의 Counter module을 이용하면 단어의 모든 빈도를 쉽게 계산할 수 있다.
5
6 sentences = []
7 stop_words = set(stopwords.words('english'))
8
9 for i in text:
10     sentence = word_tokenize(i) # 단어 토큰화를 수행.
11     result = []
12
13     for word in sentence:
14         word = word.lower() # 모든 단어를 소문자화하여 단어의 개수를 줄인다.
15         if word not in stop_words: # 단어 토큰화 된 결과에 대해서 불용어 제거.
16             if len(word) > 2: # 단어 길이가 2이하인 경우에 대하여 추가로 단어 제거.
17                 result.append(word)
18                 vocab[word] = vocab[word]+1 #각 단어의 빈도를 Count.
19     sentences.append(result)
20 print(sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kept', 'secret'], ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'], ['barber', 'went', 'huge', 'mountain']]
```

```
1 print(vocab)
```

```
Counter({'barber': 8, 'secret': 6, 'huge': 5, 'kept': 4, 'person': 3, 'word': 2, 'keeping': 2, 'good': 1, 'knew': 1, 'driving': 1, 'crazy': 1, 'went': 1, 'mountain': 1})
```

정수(Integer) Encoding (Cont.)

Step3. 각 단어들을 빈도순으로 정렬

```
1 vocab_sorted=sorted(vocab.items(), key=lambda x:x[1], reverse=True)
2 print(vocab_sorted)
```

```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3), ('word', 2), ('keeping', 2), ('good', 1), ('knew', 1), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)]
```

Step4. 높은 빈도수에 따라 Index 부여

```
1 word_to_index={}
2 i=0
3 for (word, frequency) in vocab_sorted :
4     if frequency > 1 : # 정제(Cleaning) 챕터에서 언급했듯이 빈도수가 적은 단어는 제외한다.
5         i=i+1
6         word_to_index[word]=i
7 print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7}
```

- 1의 Index가 가장 빈도수가 높다.

Keras의 Text 전처리

■ `fit_on_texts()`

- Text의 list를 가지고 단어 빈도수에 기반한 사전 생성

```
1 from keras.preprocessing.text import Tokenizer
2 text=["A barber is a person. a barber is good person. a barber is huge person. ♯
3     he Knew A Secret! The Secret He Kept is huge secret. Huge secret. His barber kept his word. ♯
4     a barber kept his word. His barber kept his secret. But keeping and keeping such a huge secret ♯
5     to himself was driving the barber crazy. the barber went up a huge mountain."]
6 t = Tokenizer()
7 t.fit_on_texts(text)
8 print(t.word_index)
```

```
{'a': 1, 'barber': 2, 'secret': 3, 'huge': 4, 'his': 5, 'is': 6, 'kept': 7, 'person': 8, 'the': 9, 'he': 10, 'word': 11, 'keeping': 12, 'good': 13, 'knew': 14, 'but': 15, 'and': 16, 'such': 17, 'to': 18, 'himself': 19, 'was': 20, 'driving': 21, 'crazy': 22, 'went': 23, 'up': 24, 'mountain': 25}
```

■ `word_index`

- 각 단어에 부여된 Index 확인

Keras의 Text 전처리 (Cont.)

■ word_counts

- 단어별 개수 파악

```
1 print(t.word_counts)
```

```
OrderedDict([('a', 8), ('barber', 8), ('is', 4), ('person', 3), ('good', 1), ('huge', 5), ('he', 2), ('knew', 1), ('secret', 6), ('the', 3), ('kept', 4), ('his', 5), ('word', 2), ('but', 1), ('keeping', 2), ('and', 1), ('such', 1), ('to', 1), ('himself', 1), ('was', 1), ('driving', 1), ('crazy', 1), ('went', 1), ('up', 1), ('mountain', 1)])
```

■ texts_to_sequences()

- 입력으로 들어온 Corpus를 **word_index**에서 정해진 Index에 맞춰서 변환하여 출력

```
1 print(t.texts_to_sequences(text))
```

```
[[1, 2, 6, 1, 8, 1, 2, 6, 13, 8, 1, 2, 6, 4, 8, 10, 14, 1, 3, 9, 3, 10, 7, 6, 4, 3, 4, 3, 5, 2, 7, 5, 11, 1, 2, 7, 5, 11, 5, 2, 7, 5, 3, 15, 12, 16, 12, 17, 1, 4, 3, 18, 19, 20, 21, 9, 2, 22, 9, 2, 23, 24, 1, 4, 25]]
```

Keras의 Text 전처리 (Cont.)

■ 빈도수가 1인 단어 제거

```
1 words_frequency = [w for w,c in t.word_counts.items() if c < 2] # 빈도수가 2미만 단어를 w라고 저장
2 for w in words_frequency:
3     del t.word_index[w] # 해당 단어에 대한 인덱스 정보를 삭제
4     del t.word_counts[w] # 해당 단어에 대한 카운트 정보를 삭제
5 print(t.texts_to_sequences(text))
6 print(t.word_index)
```

```
[[1, 2, 6, 1, 8, 1, 2, 6, 8, 1, 2, 6, 4, 8, 10, 1, 3, 9, 3, 10, 7, 6, 4, 3, 4, 3, 5, 2, 7, 5, 11, 1, 2, 7, 5, 11, 5, 2, 7, 5, 3, 1
2, 12, 1, 4, 3, 9, 2, 9, 2, 1, 4]]
{'a': 1, 'barber': 2, 'secret': 3, 'huge': 4, 'his': 5, 'is': 6, 'kept': 7, 'person': 8, 'the': 9, 'he': 10, 'word': 11, 'keeping':
12}
```

NLTK의 FreqDist Class

- NLTK에서는 Token들의 빈도를 손쉽게 셀 수 있도록 지원하는 빈도수 계산 클래스
- Token들을 입력으로 받아 해당 Token의 빈도 정보를 계산.

```
1 test_list = ['barber', 'barber', 'person', 'barber', 'good', 'person']  
2 from nltk import FreqDist  
3 fdist = FreqDist(test_list)
```

```
1 fdist.N() # 전체 단어 개수 출력
```

6

```
1 fdist.freq("barber") # 'barber'라는 단어의 확률.
```

0.5

```
1 fdist["barber"] # 'barber'라는 단어의 빈도수 출력
```

3

```
1 fdist.most_common(2) # 등장 빈도수가 높은 상위 2개의 단어만 출력
```

[('barber', 3), ('person', 2)]



Text Preprocessing

One-Hot Encoding



One-Hot Encoding

- 자연어 처리에서 문자를 숫자로 바꾸는 여러 가지 기법들중 하나.
- 가장 기본적인 표현 방법이며, Machine Learning, Deep Learning을 하기 위해서는 반드시 배워야 하는 표현 방법.
- Vocabulary(단어집합)
 - 서로 다른 단어들의 집합
 - book과 books는 서로 다른 단어로 간주.

One-Hot Encoding (Cont.)

- 먼저 해야 할 일은 갖고 있는 우선 단어 집합을 만드는 일
- Text의 모든 단어를 중복을 허락하지 않고 모아놓는다면 이를 단어 집합이라고 한다.
- 그리고 이 단어 집합에 고유한 숫자를 부여하는 일을 진행한다.
- 갖고 있는 텍스트에 단어가 5,000개면 단어 집합의 크기도 5,000이라고 한다.
- 5,000개의 단어가 있는 이 단어 집합의 단어들마다 0번부터 4,999번까지 Index를 부여한다.

One-Hot Encoding (Cont.)

- 단어 집합의 크기를 벡터의 차원으로 하고, 표현하고 싶은 단어의 Index에 1의 값을 부여하고, 다른 Index에는 0을 부여하는 단어의 벡터 표현.
- 이렇게 해서 One-Hot Vector를 만든다.
- Encoding 과정.
 1. 각 단어에 고유한 Index를 부여(정수 Encoding)
 2. 표현하고 싶은 단어의 Index의 위치에 1을 부여하고, 다른 단어의 Index의 위치에는 0을 부여.

One-Hot Encoding (Cont.)

```
1 from konlpy.tag import Okt
2 okt = Okt()
3 token = okt.morphs("나는 자연어 처리를 배운다")
4 print(token)
```

['나', '는', '자연어', '처리', '를', '배운다']

```
1 word2index={}
2 for voca in token:
3     if voca not in word2index.keys():
4         word2index[voca]=len(word2index)
5 print(word2index)
```

{'나': 0, '는': 1, '자연어': 2, '처리': 3, '를': 4, '배운다': 5}

```
1 def one_hot_encoding(word, word2index):
2     one_hot_vector = [0]*(len(word2index))
3     index = word2index[word]
4     one_hot_vector[index] = 1
5     return one_hot_vector
6
7 one_hot_encoding("자연어", word2index)
```

[0, 0, 1, 0, 0, 0]

Keras를 이용한 One-Hot Encoding

■ `to_categorical()`

- Keras에서 자동으로 One-Hot Encoding을 만들어 주는 유용한 도구

■ 정수 Encoding

```
1 text="나랑 점심 먹으러 갈래 점심 메뉴는 햄버거 갈래 갈래 햄버거 최고야"
2 from keras_preprocessing.text import Tokenizer
3 t = Tokenizer()
4 t.fit_on_texts([text])
5
6 print(t.word_index)
```

{'갈래': 1, '점심': 2, '햄버거': 3, '나랑': 4, '먹으러': 5, '메뉴는': 6, '최고야': 7}

```
1 text2 = "점심 먹으러 갈래 메뉴는 햄버거 최고야"
2 x=t.texts_to_sequences([text2])
3 print(x)
```

[[2, 5, 1, 6, 3, 7]]

Keras를 이용한 One-Hot Encoding (Cont.)

```
1 vocab_size = len(t.word_index) # 단어 집합의 크기. 이 경우는 단어의 개수가 7이므로 7.  
2  
3 from keras.utils import to_categorical  
4 x = to_categorical(x, num_classes=vocab_size+1) # 실제 단어 집합의 크기보다 +1로 크기를 만들어야함.  
5  
6 print(x)
```

```
[[[0. 0. 1. 0. 0. 0. 0. 0.]  
  [0. 0. 0. 0. 0. 1. 0. 0.]  
  [0. 1. 0. 0. 0. 0. 0. 0.]  
  [0. 0. 0. 0. 0. 0. 1. 0.]  
  [0. 0. 0. 1. 0. 0. 0. 0.]  
  [0. 0. 0. 0. 0. 0. 0. 1.]]]
```

Keras를 이용한 One-Hot Encoding (Cont.)

■ 주의할 점

- `to_categorical()` 은 정수 encoding으로 부여된 Index를 그대로 배열의 Index로 사용
- `t.fit_on_texts()` 를 사용하여 정수 encoding시 실제 단어 집합의 크기보다 +1의 크기를 인자로 주어야 한다.
- `t.fit_on_texts()` 는 Index를 1부터 부여하기 때문에 배열의 Index가 0부터 시작하므로 맨 마지막 Index를 가진 단어의 Index가 7 이므로, 이를 One-Hot Vector로 만들기 위해서 8의 크기를 가진 배열이 필요.

One-Hot Encoding (Cont.)

■ One-Hot Encoding의 한계

- 단어의 개수가 늘어날 수록, Vector를 저장하기 위해 필요한 공간이 계속 늘어난다.
- One-Hot Vector는 단어 집합의 크기가 곧 Vector의 차원 수가 된다.
- 매우 비효율적 방법
- 단어의 유사성을 전혀 표현하지 못한다는 단점도 있다.
- 검색 시스템 등에서 심각한 문제 유발

One-Hot Encoding (Cont.)

- One-Hot Encoding의 한계 해결방안
 - 단어의 '의미'를 다차원 공간에 Vector화 하는 기법 2가지
 - Count 기반으로 단어의 의미를 Vector화 → LSA, HAL
 - 예측 기반으로 단어의 의미를 Vector화 → 전통 NNLM, RNNLM, Word2Vec, FastText 등
 - Count 기반과 예측 기반 두 가지 방법을 모두 사용하는 방법 → Glove



Text Preprocessing

Subword Segmentation



단어 분리(Subword Segmentation)

- 현실적으로 이 세상의 모든 단어를 기계에서 학습하는 것은 불가능
- 단어 집합(Vocabulary)
 - 기계가 훈련 단계에서 학습한 단어들의 집합
 - 기계가 암기한 단어들의 리스트
- OOV(Out-Of-Vocabulary)
 - 테스트 단계에서 기계가 미처 배우지 못한 모르는 단어들
 - 단어 집합에 없는 단어
 - UNK(Unknown Word)으로 표현하기도
- 그렇다면 기계가 모르는 단어로 인해 문제를 풀지 못하는 상황을 OOV 문제라고 한다.

단어 분리(Subword Segmentation) (Cont.)

■ Subword Segmentation

- 기계가 아직 배운 적이 없는 단어더라도 대처할 수 있도록 도와주는 기법
- 기계 번역 등에서 주요 전처리로 사용되고 있다.

■ OOV 문제 해결 방법

- BPE(Byte Pair Encoding)
- WPM(Word Piece Model)

BPE(Byte Pair Encoding) Algorithm

- 1994년에 데이터 압축 Algorithm으로 제안 된 후 자연어 처리의 단어 분리 Algorithm으로 응용.
- 기본적으로 연속적으로 가장 많이 등장한 글자의 쌍을 찾아서 하나의 글자로 병합하는 방식 수행

aaabddaaabac

- 위의 문자열 중 가장 자주 등장하고 있는 바이트의 쌍(byte pair)은 aa
- 이 aa라는 바이트의 쌍을 하나의 바이트인 Z로 치환.

aaabddaaabac → ZabdZabac | Z = aa

BPE(Byte Pair Encoding) Algorithm (Cont.)

$aaabdaaabc \rightarrow ZabdZabac \quad | \quad Z = aa$

- 이제 위 문자열 중에서 가장 많이 등장하고 있는 바이트의 쌍은 ab .
- 이제 이 ab 를 Y 로 치환.

$ZabdZabac \rightarrow ZYdZYac \quad | \quad Z = aa \quad Y = ab$

- 이제 가장 많이 등장하고 있는 바이트의 쌍은 ZY .
- 이를 X 로 치환.

$aaabdaaabc \rightarrow ZabdZabac \rightarrow ZYdZYac \rightarrow XdXac$
 $Z = aa \quad Y = ab \quad X = ZY$

BPE(Byte Pair Encoding) Algorithm (Cont.)

- 단어 분리(Word Segmentation) Algorithm.
- 글자(Character) 단위에서 점차적으로 단어 집합(Vocabulary)을 만들어 내는 Bottom up 방식의 접근 사용.
- 훈련 데이터에 있는 단어들을 모든 글자(Characters) 또는 유니코드(Unicode) 단위로 단어 집합(Vocabulary) 생성
- 가장 많이 등장하는 Unigram을 하나의 Unigram으로 통합.

기존 접근 방법

```
# dictionary
# 훈련 데이터에 있는 단어와 등장 빈도수
low : 5, lower : 2, newest : 6, widest : 3
```

- 이 훈련 데이터에는 'low'란 단어가 5회 등장하였고, 'lower'란 단어는 2회 등장하였으며, 'newest'란 단어는 6회, 'widest'란 단어는 3회 등장했다는 의미.

```
# vocabulary
low, lower, newest, widest
```

- 현재의 이 훈련 데이터의 단어 집합에는 'low', 'lower', 'newest', 'widest'라는 4개의 단어가 존재.
- 만일 'lowest'란 단어가 등장하면 기계는 이 단어를 학습한 적이 없으므로 해당 단어에 대해서 제대로 대응하지 못하는 OOV 문제가 발생.

BPE Algorithm 적용

```
# dictionary
```

```
# 훈련 데이터에 있는 단어와 등장 빈도수
```

```
low : 5, lower : 2, newest : 6, widest : 3
```

- 우선 dictionary의 모든 단어들을 글자(Character) 단위로 분리

```
# dictionary
```

```
# 훈련 데이터에 있는 단어와 등장 빈도수
```

```
l o w : 5, l o w e r : 2, n e w e s t : 6, w i d e s t : 3
```

- dictionary를 참고로 만든 초기 단어 집합(Vocabulary)
- 초기 구성은 글자 단위로 분리된 상태

```
# vocabulary
```

```
l , o , w , e , r , n , s , t , i , d
```

BPE Algorithm 적용 (Cont.)

- BPE Algorithm의 특징은 Algorithm의 동작을 몇 회 반복할 것인지를 사용자가 정해야 한다.
- 만일 총 10회를 수행한다고 가정하면, 즉, 가장 빈도수가 높은 Unigram의 쌍을 하나의 Unigram으로 통합하는 과정을 총 10회 반복한다는 것.
- 아래의 dictionary에 따르면 빈도수가 현재 가장 높은 Unigram의 쌍은 (e, s).

dictionary

훈련 데이터에 있는 단어와 등장 빈도수

l o w : 5 , l o w e r : 2 , n e w e s t : 6 , w i d e s t : 3

BPE Algorithm 적용 (Cont.)

1회 - dictionary를 참고로 하였을 때 빈도수가 9로 가장 높은 (e, s)의 쌍을 es로 통합한다.

```
# dictionary update
```

```
l o w : 5,  
l o w e r : 2,  
n e w e s t : 6,  
w i d e s t : 3
```

```
# vocabulary update
```

```
l , o , w , e , r , n , s , t , i , d , es
```


BPE Algorithm 적용 (Cont.)

2회 – 빈도가 9로 가장 높은 (es, t)의 쌍을 est로 통합한다.

```
# dictionary update
```

```
l o w : 5,  
l o w e r : 2,  
n e w est : 6,  
w i d est : 3
```

```
# vocabulary update
```

```
l , o , w , e , r , n , s , t , i , d , es , est
```

BPE Algorithm 적용 (Cont.)

3회 – 빈도가 7로 가장 높은 (l, o)의 쌍을 lo로 통합한다.

```
# dictionary update
```

```
lo w : 5,  
lo w e r : 2,  
n e w est : 6,  
w i d est : 3
```

```
# vocabulary update
```

```
l, o, w, e, r, n, s, t, i, d, es, est, lo
```

BPE Algorithm 적용 (Cont.)

10회째

```
# dictionary update
```

```
low : 5,
```

```
low e r : 2,
```

```
newest : 6,
```

```
widest : 3
```

```
# vocabulary update
```

```
l, o, w, e, r, n, s, t, i, d, es, est, lo, low, ne, new, newest, wi, wid, widest
```

BPE Algorithm 적용 (Cont.)

vocabulary update

l, o, w, e, r, n, s, t, i, d, es, est, lo, low, ne, new, newest, wi, wid, widest

- 이 경우 테스트 과정에서 *lowest*란 단어가 등장한다면
 - 기존에는 OOV에 해당되는 단어
 - BPE Algorithm을 사용한 위의 단어 집합에서는 더 이상 *lowest*는 OOV가 아니다.
 - 기계는 우선 *lowest*를 전부 글자 단위로 분할.
l, o, w, e, s, t
 - 기계는 위의 단어 집합을 참고로 하여 low와 est를 찾는다.
 - *lowest*를 기계는 low와 est 두 단어로 encoding.
 - 이 두 단어는 둘 다 단어 집합에 있는 단어이므로 OOV가 아니다.

BPE Algorithm 적용 (Cont.)

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\W\S)' + bigram + r'(?!\W\S)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out
```

```
vocab = {'l o w' : 5,
         'l o w e r' : 2,
         'n e w e s t' : 6,
         'w i d e s t' : 3
        }
```

```
num_merges = 10
```

```
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

BPE Algorithm 적용 (Cont.)

- BPE 알고리즘의 기본 원리는 가장 많이 등장한 문자열에 대하여 병합 하는 작업을 반복하는데, 원하는 단어 집합의 크기. 즉, 단어의 갯수가 될 때까지 이 작업을 반복하는 것이다.
- 단어 분리(Subword Segmentation)가 의미 있는 이유
 - 단어 분리(Subword Segmentation) 작업은 하나의 단어는 의미 있는 여러 단어들의 조합으로 구성된 경우가 많기 때문
 - 단어를 여러 단어로 분리해보겠다는 전처리 작업
 - 실제로, 언어의 특성에 따라 영어권 언어나 한국어는 단어 분리를 시도했을 때 어느 정도 의미 있는 단위로 나누는 것이 가능하다.

WPM(Word Piece Model)

- 하나의 단어를 내부 단어(Subword Unit)들로 분리하는 단어 분리 모델.
- 2016년에 Google이 낸 논문(Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation [Wu et al.2016])에서
- Google Translator에서 WPM이 어떻게 수행되는지에 대해 기술.

WPM(Word Piece Model) (Cont.)

- WPM을 수행하기 이전의 문장
 - Jet makers feud over seat width with big orders at stake
- WPM을 수행한 결과(wordpieces)
 - _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake
- 기존에 존재하던 띄어쓰기는 underline()로 치환
- 단어는 내부단어(subword)로 통계에 기반하여 띄어쓰기로 분리

WPM(Word Piece Model) (Cont.)

- 기존의 띄어쓰기를 underline()로 치환하는 이유는 차후 다시 문장 복원을 위한 장치이다.
- WPM의 결과로 나온 문장을 보면, 기존에 없던 띄어쓰기가 추가되어 내부 단어(subwords)들을 구분하는 구분자 역할을 수행.
- 본래의 띄어쓰기를 underline으로 치환해놓지 않으면, 기존 문장으로 복원할 수가 없다.
- WPM이 수행된 결과로부터 다시 수행 전의 결과로 돌리는 것 방법은 현재 있는 띄어쓰기를 전부 삭제하여 내부 단어들을 다시 하나의 단어로 연결시키고, underline을 다시 띄어쓰기로 바꾸면 된다.



Word Cloud



Word Cloud

- 단어를 출현 빈도에 비례하는 크기로 단어의 빈도수를 시각화하는 기법
- 빈도분석이 된 데이터를 시각화하기 위해 Word Cloud를 사용
- `pip install wordcloud`

Sample을 이용한 Word Cloud 그리기

```
1 from konlpy.corpus import kolaw
2 data = kolaw.open('constitution.txt').read()
```

```
1 from konlpy.tag import Komoran
2 komoran = Komoran()
```

헌법 말뭉치를 불러와 형태
소 분석 후 명사만 추출함

```
1 print(komoran.nouns("%r"%data[0:1000]))
```

['대한민국', '헌법', '유구', '한', '역사', '전통', '국민', '운동', '건
립', '대한민국', '임시', '정부', '법통', '불의', '항거', '민주', '이념',
'계승', '조국', '민주개혁', '평화', '통일', '사명', '입각', '정의', '인
도', '동포애', '민족', '단결', '사회', '폐습', '불의', '타파', '자율',
'조화', '바탕', '자유', '민주', '기본', '질서', '정치', '경제', '사회',
'문화', '영역', '각인', '기회', '능력', '최고', '도로', '발휘', '자유',
'권리', '책임', '의무', '완수', '안', '국민', '생활', '균등', '향상',
'밖', '항구', '세계', '평화', '인류', '공영', '이바지', '우리들의', '자
소', '아저', '자유', '해본', '화부', '거', '다진', '녀', '7월 12일', '제

Sample을 이용한 Word Cloud 그리기 (Cont.)

```
1 word_list = komoran.nouns("%r"%data[0:1000])
```

```
1 text = ' '.join(word_list)
```

명사들을 공백으로 이어서
하나의 문장으로 만듦

```
1 text
```

'대한민국 헌법 유구 한 역사 전통 국민 운동 건립 대한민국 임시 정부 법통 불의 항거 민주 이념 계승 조국 민주개혁 평화 통일 사명 입각 정의 인도 동포애 민족 단결 사회 폐습 불의 타파 자율 조화 바탕 자유 민주 기본 질서 정치 경제 사회 문화 영역 각인 기회 능력 최고 도로 발취 자유 권리 책임 의무 완수 안 국민 생활 균등 향상 밖 항구 세계 평화 인류 공영 이바지 우리들의 자손 안전 자유 행복 확보 것 다짐 년 7월 12일 제정 차 개정 헌법 국회 의결 국민 투표 개정 장 강 대한민국 민주공화국 대한민국 주권 국민 권력 국민 대한민국 국민 요건 법률 국가 법률 바 재외국민 보호 의무 대한민국 영토 한반도 부속 도서 대한민국 통일 지향 자유 민주 기본 질서 입각 평화 통일 정책 수립 추진 대한민국 국제 평화 유지 노력 침략 전쟁 부인 국군 국가 안전 보장 국토방위 신성 의무 수행 사명 정치 중립 준수 헌법 체결 공포 조약 일반 승인 국제 법규 국내법 효력 외국인 국제법 조약 바 지위 보장 공무원 국민 전체 봉사자 국민 책임 공무원 신분 정치 중립 법률 바 보장 정당 설립 자유 복수 정당'

Sample을 이용한 Word Cloud 그리기 (Cont.)

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 from wordcloud import WordCloud
```

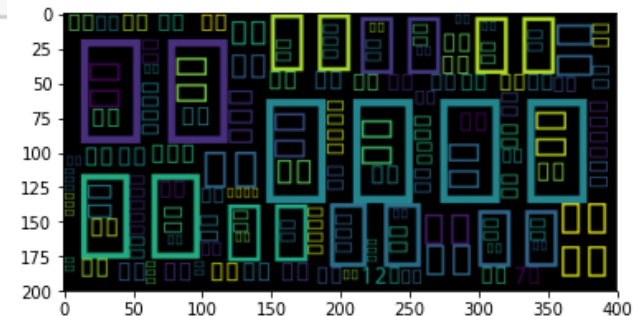
```
1 wordc = WordCloud()  
2 wordc.generate(text)
```

한글이 깨짐

```
<wordcloud.wordcloud.WordCloud at 0x1d532a20>
```

```
1 plt.figure()  
2 plt.imshow(wordc, interpolation="bilinear")
```

```
<matplotlib.image.AxesImage at 0x1f2da6d8>
```



한글 처리

```
1 wordc = WordCloud(background_color='white', max_words=20,  
2 font_path='c:/Windows/Fonts/malgun.ttf',  
3 relative_scaling=0.2)
```

```
1 wordc.generate(text)
```

<wordcloud.wordcloud.WordCloud at 0x1d536cc0>

```
1 plt.figure()  
2 plt.imshow(wordc, interpolation="bilinear")  
3 plt.axis('off')
```

통일 자유국민 법률
헌법 개정헌법 사명기초
대한민국
책임 안전 의무

전체 데이터를 이용한 Word Cloud 생성

```
1 word_list = komoran.nouns("%r"%data)
2 text = ' '.join(word_list)
```

```
1 wordcloud = WordCloud(background_color='white', max_words=2000,  
2                       font_path='c:/Windows/Fonts/malgun.ttf',  
3                       relative_scaling=0.2)
```

```
1 wordcloud.generate(text)
```

```
<wordcloud.wordcloud.WordCloud at 0x203626a0>
```

```
1 plt.figure(figsize=(15, 10))
2 plt.imshow(wordcloud, interpolation="bilinear")
3 plt.axis('off')
```



불용어 사전 추가

```
1 from wordcloud import STOPWORDS
2 from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
3
4 불용어 = STOPWORDS | ENGLISH_STOP_WORDS | set(["대통령", "국가"])
```

```
1 wordcloud = WordCloud(background_color='white', max_words=2000,  
2 stopwords=불용어,  
3 font_path='c:/Windows/Fonts/malgun.ttf',  
4 relative_scaling=0.2)  
5 wordcloud.generate(text)
```

```
<wordcloud.wordcloud.WordCloud at 0x204a2f28>
```

```
1 plt.figure(figsize=(15, 10))
2 plt.imshow(wordcloud, interpolation="bilinear")
3 plt.axis('off')
```



“대통령”, “국가” 단어를 불용어로 지정

Masking

- ✓ Word Cloud를 지정된 Mask Image에 맞도록 표시
- ✓ Mask Image는 <http://coderby.com/img/16>

```
1 from PIL import Image
2 import numpy as np
3 img = Image.open("south_korea.png").convert("RGBA")
4 mask = Image.new("RGB", img.size, (255,255,255))
5 mask.paste(img, img)
6 mask = np.array(mask)
```



```
1 wordcloud = WordCloud(background_color='white', max_words=2000,  
2                       font_path='C:/Windows/Fonts/malgun.ttf',  
3                       mask=mask, random_state=42)  
4 wordcloud.generate(text)  
5 wordcloud.to_file("result1.png")
```

Pallet 변경

파일은 <http://coderby.com/img/15> 에서

```
1 import random
2 def grey_color(word, font_size, position, orientation,
3                 random_state=None, **kwargs):
4     return 'hsl(0, 0%%, %d%%)' % random.randint(50, 100)
```

%표현

```

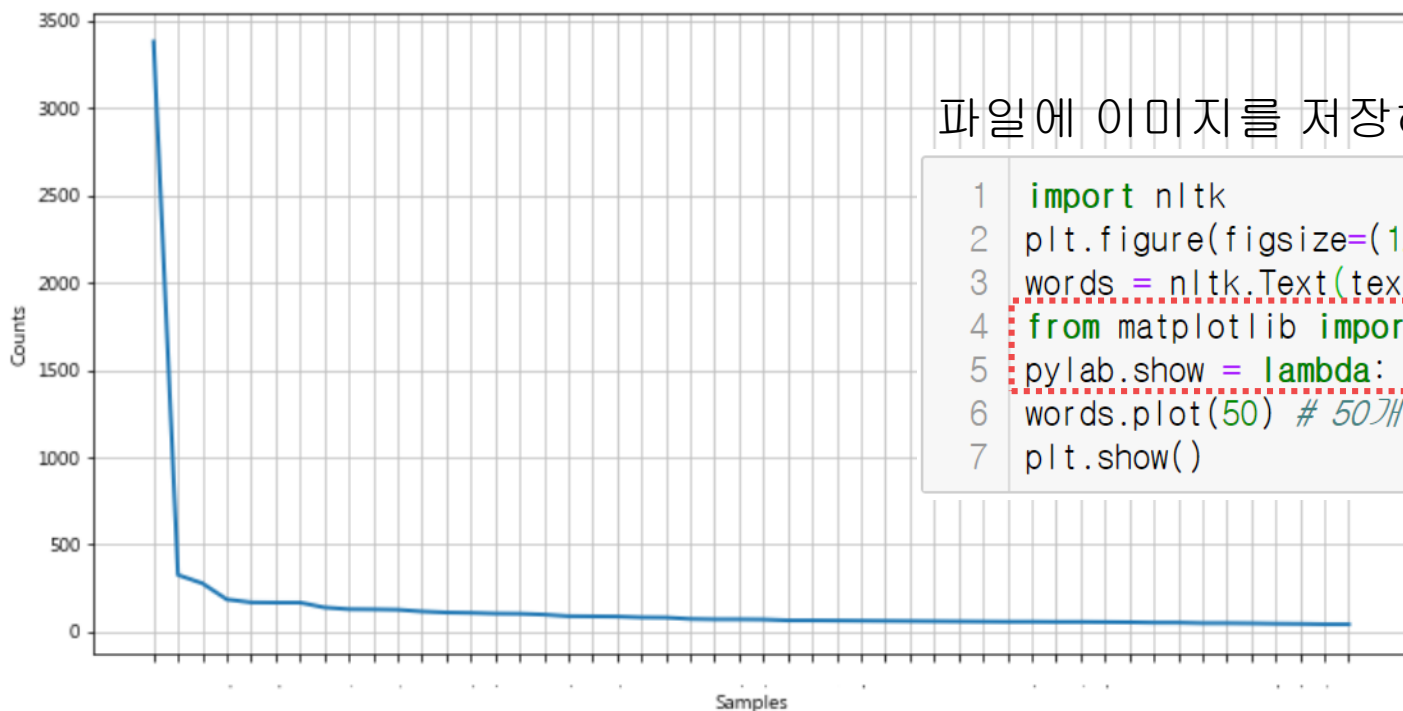
1  from PIL import Image
2  img = Image.open("south_korea_4x.png").convert("RGBA")
3  mask = Image.new("RGB", img.size, (255,255,255))
4  mask.paste(img, img)
5  mask = np.array(mask)
6  wordcloud = WordCloud(background_color='black', max_words=2000,
7                        font_path='C:/Windows/Fonts/malgun.ttf',
8                        mask=mask, random_state=42)
9  wordcloud.generate(text)
10 wordcloud.recolor(color_func=grey_color, random_state=3)
11 wordcloud.to_file("result2.png")

```



단어 빈도수 계산

```
1 import nltk
2
3 plt.figure(figsize=(12,6))
4 words = nltk.Text(text)
5 words.plot(50) # 50개만
6 plt.show()
```



파일에 이미지를 저장하고 싶을 경우...

```
1 import nltk
2 plt.figure(figsize=(12,6))
3 words = nltk.Text(text)
4 from matplotlib import pylab
5 pylab.show = lambda: pylab.savefig('word_count.png')
6 words.plot(50) # 50개만
7 plt.show()
```





Association Analysis

오렌지주스를 구매하는 사람은 와인을 구매할까?

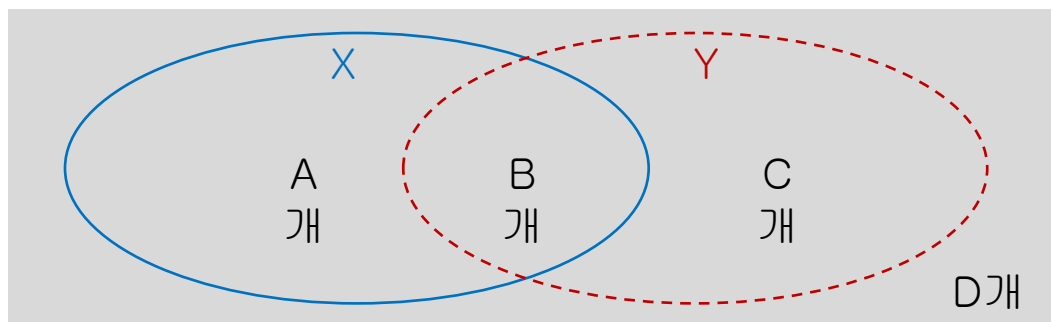
- 1 소주, 콜라, 와인
- 2 소주, 오렌지주스, 콜라
- 3 콜라, 맥주, 와인
- 4 소주, 콜라, 맥주
- 5 오렌지주스, 와인

연관 분석

- 연관 분석(Association Analysis)은 데이터들 사이에서 '자주 발생하는 속성을 찾는' 그리고 그 속성들 사이에 '연관성이 어느 정도 있는지'를 분석하는 방법
- 연관 규칙을 찾는 알고리즘으로는 apriori, FP-growth, DHP 알고리즘 등이 있음
- apriori 알고리즘이 비교적 구현이 간단하고 성능 또한 높음
- pip install apyori

연관 분석 평가

- 지지도(Support)는 전체 Transaction에서 항목 집합(X, Y)을 포함하는 Transaction의 비율을 의미
- 신뢰도(Confidence)는 X를 포함하는 Transaction 중 Y도 포함하는 Transaction의 비율을 의미



범례 : ○ X를 포함하는 트랜잭션
○ Y를 포함하는 트랜잭션

$$\text{지지도} = \text{B개} / (\text{A개} + \text{B개} + \text{C개} + \text{D개})$$

$$\text{신뢰도} = \text{B개} / (\text{A개} + \text{B개})$$

연관 분석 평가 척도

평가 척도	설명
지지도 (Support)	<ul style="list-style-type: none">· 연관 규칙이 얼마나 중요한지에 대한 평가· 낮은 지지도의 연관 규칙은 우연히 발생한 트랜잭션에서 생성된 규칙일 수 있음
신뢰도 (Confidence)	<ul style="list-style-type: none">· 연관 규칙이 얼마나 믿을 수 있는지 평가
향상도 (Lift)	<ul style="list-style-type: none">· $Lift = P(Y X) / P(Y) = P(X \cap Y) / P(X)P(Y) = \text{신뢰도} / P(Y)$· $X \rightarrow Y$의 신뢰도 / Y의 지지도· 1 보다 작으면 음의 상관관계(설사약과 변비약), 1이면 상관관계 없음(과자와 후추), 1 보다 크면 양의 상관관계(빵과 버터)
파이 계수 (Phi coefficient)	<ul style="list-style-type: none">· -1 : 완전 음의 상관관계, 0 : 상관관계 없음, 1 : 완전 양의 상관관계
IS 척도 (Interest-Support)	<ul style="list-style-type: none">· 비대칭적 이항 변수에 적합한 척도로 값이 클수록 상관관계가 높음· $P(X, Y) / P(X) * P(Y)$



Transaction Data



CSV file로부터 Transaction Data 생성

- 트랜잭션(Transaction)은 서로 관련 있는 데이터의 모음
- 연관분석 시 Transaction으로부터 연관 규칙을 도출
- Python의 Transaction Data는 list형식

```
1 import csv
2 with open('basket.csv', 'r', encoding='UTF8') as cf:
3     transactions = []
4     r = csv.reader(cf)
5     for row in r:
6         transactions.append(row)
7
```

basket.csv

```
1 소주,콜라,와인
2 소주,오렌지주스,콜라
3 콜라,맥주,와인
4 소주,콜라,맥주
5 오렌지주스,와인
```

```
1 transactions
```

```
[['소주', '콜라', '와인'],
 ['소주', '오렌지주스', '콜라'],
 ['콜라', '맥주', '와인'],
 ['소주', '콜라', '맥주'],
 ['오렌지주스', '와인']]
```

연관 규칙 생성

- `apriori()` 연역적 알고리즘(A Priori Algorithm)을 사용하여 연관 규칙을 알아내는 함수

```
1 from apyori import apriori
2 rules = apriori(transactions, min_support=0.1, min_confidence=0.1)
3 results = list(rules)
4 type(results)
```

list

```
1 results[0]
```

```
RelationRecord(items=frozenset({'맥주'}), support=0.4, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'맥주'}), confidence=0.4, lift=1.0)])
```

```
1 results[10]
```

```
RelationRecord(items=frozenset({'콜라', '소주'}), support=0.6, ordered_statistics=[OrderedStatistic(items_base=frozenset({'소주'}), items_add=frozenset({'콜라'}), confidence=1.0, lift=1.25), OrderedStatistic(items_base=frozenset({'콜라'}), items_add=frozenset({'소주'}), confidence=0.7499999999999999, lift=1.2499999999999998)])
```

이렇게 생성한 규칙은 **namedtuple** 형식으로 저장되어 있음

연관 규칙 형식

- 연관 규칙은 *namedtuple* 형식으로 저장되어 있기 때문에 원하는 값을 찾거나 출력하기 위해서는 연관 규칙 결과가 어떤 형식으로 저장되어 있는지 이해해야 함
- apyori.py 파일에 정의되어 있는 **RelationRecord**와 **OrderStatistic**은 다음처럼 *namedtuple*로 정의되어 있음

```
SupportRecord = namedtuple( 'SupportRecord', ('items', 'support'))
```

```
RelationRecord = namedtuple( 'RelationRecord', SupportRecord._fields + ('ordered_statistics',))
```

```
OrderedStatistic = namedtuple( 'OrderedStatistic', ('items_base', 'items_add', 'confidence', 'lift',))
```

C:\Users\사용자\Anaconda3\Lib\site-packages\apyori.py

연관 규칙 조회

```
1 print("lhs Wtrhs WtWtsupportWtWtconfidenceWtlift")
2 for row in results:
3     support = row[1]
4     ordered_stat = row[2]
5     for ordered_item in ordered_stat:
6         lhs = [x for x in ordered_item[0]]
7         rhs = [x for x in ordered_item[1]]
8         confidence = ordered_item[2]
9         lift = ordered_item[3]
10    print(lhs, " => ", rhs, "Wt{:>5.4f}Wt{:>5.4f}Wt{:>5.4f}".W
11          format(support, confidence, lift))
12
```

lhs	rhs	support	confidence	lift
[] =>	['맥주']	0.4000	0.4000	1.0000
[] =>	['소주']	0.6000	0.6000	1.0000
[] =>	['오렌지주스']	0.4000	0.4000	1.0000
[] =>	['와인']	0.6000	0.6000	1.0000
[] =>	['콜라']	0.8000	0.8000	1.0000
['맥주'] =>	['소주']	0.2000	0.5000	0.8333
['소주'] =>	['맥주']	0.2000	0.3333	0.8333
['맥주'] =>	['와인']	0.2000	0.5000	0.8333

연관 규칙 평가

- 오렌지주스를 구매한 사람은 와인을 구매할까?
- 앞의 결과에서 오렌지주스와 와인의 연관관계를 보면 다음과 같음
 - ['오렌지주스'] => ['와인'], 0.2000, 0.5000, 0.8
 - ['와인'] => ['오렌지주스'], 0.2000, 0.3333, 0.8
- 오렌지주스와 와인을 구매한 사람은 순서에 상관없이 전체의 20%
- 오렌지주스를 산 고객의 50%가 와인을 구매
- 향상도는 0.833 이고 향상도가 1보다 작은 것은 음의 상관관계를 의미
- 그러므로 오렌지주스를 구매한 사람은 와인을 구매하지 않음





뉴스 기사 연관 분석 실습

뉴스 RSS 서버에서 링크 주소 가져오기

- JTBC 경제분야 Crawling 후 연관 분석
- RSS 주소: <http://fs.jtbc.joins.com/RSS/economy.xml>

```
1 import requests
2 rss_url = "http://fs.jtbc.joins.com/RSS/economy.xml"
3 jtbc_economy = requests.get(rss_url)
```

```
1 from bs4 import BeautifulSoup
2 economy_news_list = BeautifulSoup(jtbc_economy.content, "xml")
3 link_list = economy_news_list.select("item > link")
```

```
1 len(link_list)
```

20

```
1 link_list[0].text
```

'http://news.jtbc.joins.com/article/article.aspx?news_id=NB11822526'

기사 수집 및 형태소 분석

- 기사 원 글을 수집해서 기사의 본문 내용을 형태소 분석 후 명사만 뽑음

```
1 #!pip install konlpy
```

```
1 from konlpy.tag import Kkma
2 kkma = Kkma()
```

[illegible]

연관 분석

■ 연관 규칙을 생성하고, DataFrame으로 만들

```
1 from apyori import apriori
2 rules = apriori(news, min_support=0.3, min_confidence=0.2)
3 results = list(rules)
```

```
1 import pandas as pd
2 result_df = pd.DataFrame(None,
3                           columns=["lhs", "rhs", "support",
4                                   "confidence", "lift"])
5 index = 0
6 for row in results:
7     support = row[1]
8     ordered_stat = row[2]
9     for ordered_item in ordered_stat:
10         lhs = " ".join(x.strip() for x in ordered_item[0])
11         rhs = " ".join(x.strip() for x in ordered_item[1])
12         confidence = ordered_item[2]
13         lift = ordered_item[3]
14         result_df.loc[index] = [lhs, rhs, support, confidence, lift]
15         index = index + 1
```

연관 분석 탐색

■ 뉴스 기사 연관 분석

```
1 result_df.loc[(result_df.lhs.str.contains("택시")) &  
2               (result_df.rhs=="공유")].sort_values(by=["lift"],  
3                                                    ascending=False)
```

		lhs	rhs	support	confidence	lift
4084	서비스 대표 택시 앵커 업계	공유	0.3	1.000000	3.333333	
217	경제 택시	공유	0.3	1.000000	3.333333	
4402	혁신 이재웅 이재 택시 업계	공유	0.3	1.000000	3.333333	
4632	서비스 대표 경제 택시 앵커 업계	공유	0.3	1.000000	3.333333	
4653	서비스 이재 경제 택시 앵커 대표	공유	0.3	1.000000	3.333333	
4667	이재웅 서비스 경제 택시 앵커 대표	공유	0.3	1.000000	3.333333	
4681	혁신 서비스 경제 택시 앵커 대표	공유	0.3	1.000000	3.333333	