딥러닝을 활용한 자연어 분석

구조/ 의미분석

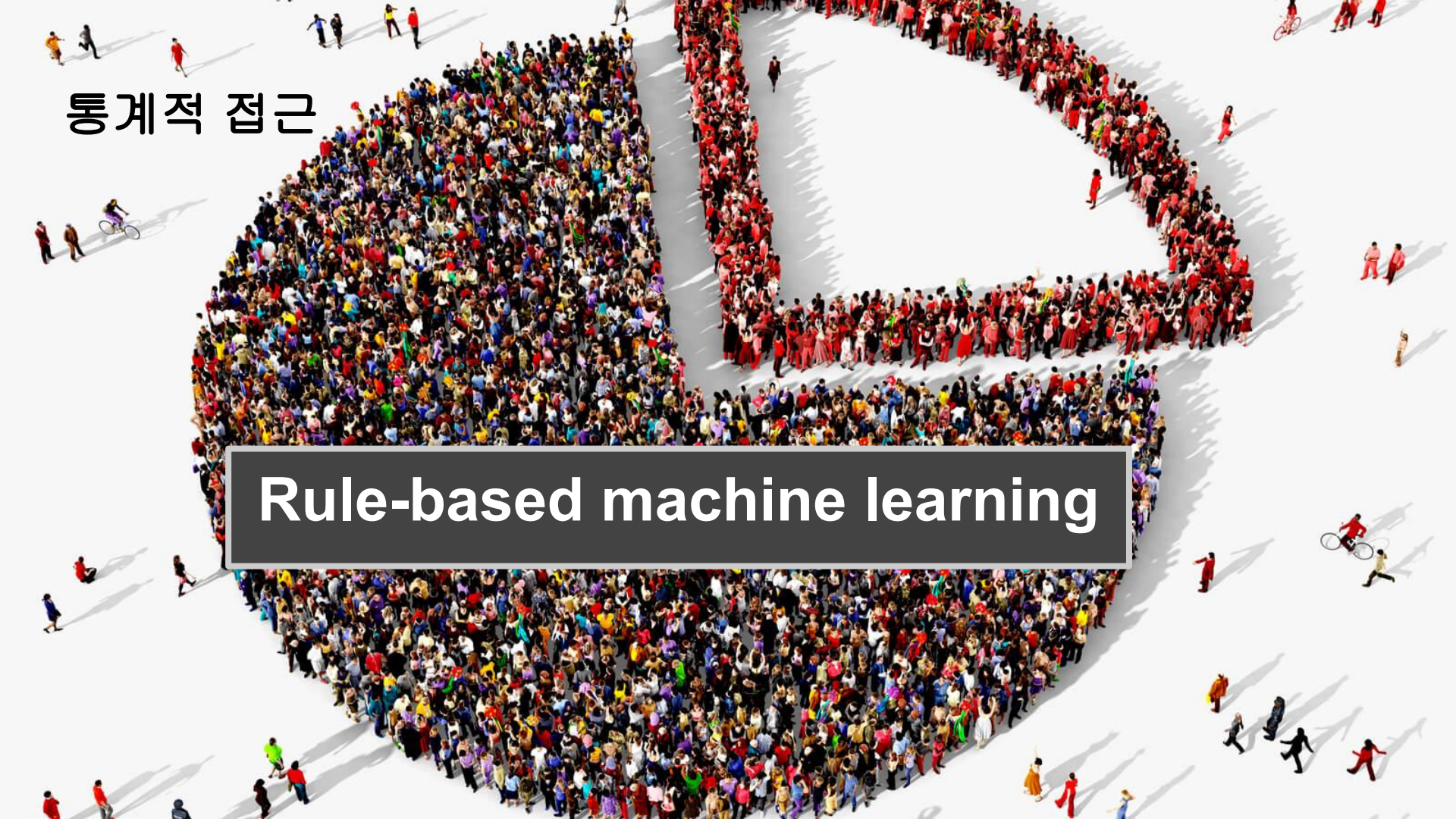
김용범

무영인터내쇼날

Parse Tree & Parsing

문장의 구조

Parsing


통계적 접근
Rule-based machine learning

Parse Tree

In [1]:
```
%%time
text = '민병삼 대령의 항명행위로 초치했다'

from konlpy.tag import Okt
twitter = Okt()
words = twitter.pos(text, stem=True)
print(words)
```

```
[('민병삼', 'Noun'), ('대령', 'Noun'), ('의', 'Josa'), ('항', 'Noun'), ('명', 'Suffix'), ('행
위', 'Noun'), ('로', 'Josa'), ('초치', 'Noun'), ('하다', 'Verb')]
CPU times: user 8.02 s, sys: 381 ms, total: 8.4 s
Wall time: 3.17 s
```

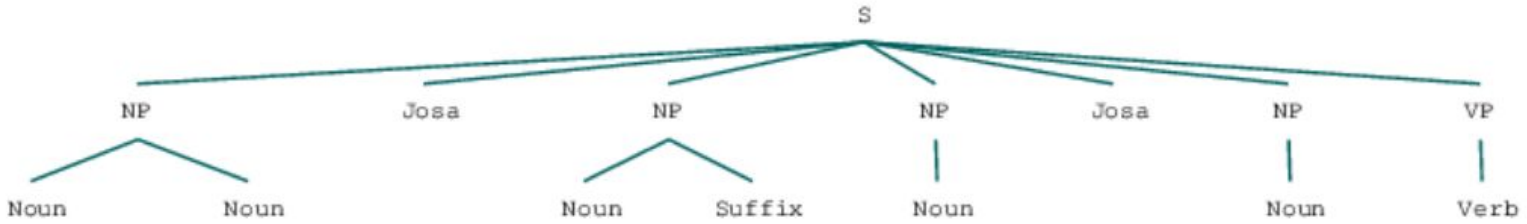
In [2]:
```
from nltk import RegexpParser

grammar = """
NP: {<N.*>*<Suffix>?}   # 명사구를 정의한다
VP: {<V.*>*}            # 동사구를 정의한다
AP: {<A.*>*}            # 형용사구를 정의한다 """
parser = RegexpParser(grammar)
parser
```

Out[2]:
```
<chunk.RegexpParser with 3 stages>
```

Parse Tree

In [3]:
```
chunks = parser.parse(words)
chunks
```

Out[3]:



In [4]:
```
text_tree = [list(txt)    for txt in chunks.subtrees()]
text_tree[1:]
```

Out[4]:
```
[[('민병삼', 'Noun'), ('대령', 'Noun')],
 [('항', 'Noun'), ('명', 'Suffix')],
 [('행위', 'Noun')],
 [('초치', 'Noun')],
 [('하다', 'Verb')]]
```

Parsing

1. **Pars** (품사|라틴어) 단어에서 유래
2. 문자열을 의미있는 **토큰(token)**으로 분해하고 이들로 이루어진 **파스 트리(parse tree)**를 만드는 과정
3. **Parse Tree**로 **형식문법(formal Grammer)**을 분석
4. 개별 **Parse** 의 **의미**를 추가하여 **의미적 정확도**를 판단

Parsing 분석 이론들

1. **CFG (Context Free Grammer)** : 중심 구조분석으로 '노암촘스키'가 제안
2. **Earley 차트 파싱 알고리즘 :** 좌측 재귀적 구조분석

Parsing 확률측정을 위한 DB

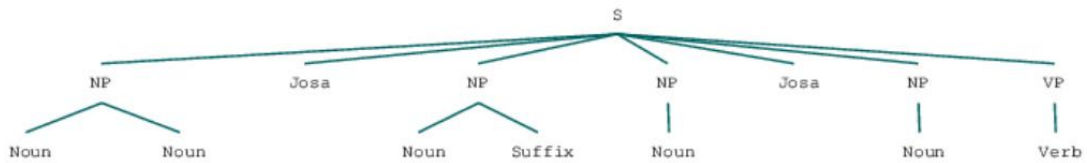
In [4]:
```
from nltk.corpus import treebank
print(treebank.words('wsj_0007.mrg'))
print(treebank.tagged_words('wsj_0007.mrg'))
print(treebank.parsed_sents('wsj_0007.mrg')[2])
```

```
['McDermott', 'International', 'Inc.', 'said', '0', ...]
[('McDermott', 'NNP'), ('International', 'NNP'), ...]
(S
  (NP-SBJ
    (NP (NNP Bailey) (NNP Controls))
    (, ,)
    (VP
      (VBN based)
      (NP (-NONE- *))
      (PP-LOC-CLR
        (IN in)
        (NP (NP (NNP Wickliffe)) (, ,) (NP (NNP Ohio)))))
    (, ,))
  (VP
    (VBZ makes)
    (NP
```

촘스키 CFG (문맥자유문법규칙) - (Top Down 방식)

In [8]:
```
from nltk.grammar import toy_pcfg2
grammar = toy_pcfg2
print(grammar)
```

```
Grammar with 23 productions (start state = S)
    S -> NP VP [1.0]
    VP -> V NP [0.59]
    VP -> V [0.4]
    VP -> VP PP [0.01]
    NP -> Det N [0.41]
    NP -> Name [0.28]
    NP -> NP PP [0.31]
    PP -> P NP [1.0]
    V -> 'saw' [0.21]
    V -> 'ate' [0.51]
    V -> 'ran' [0.28]
    N -> 'boy' [0.11]
    N -> 'cookie' [0.12]
    N -> 'table' [0.13]
    N -> 'telescope' [0.14]
    N -> 'hill' [0.5]
    Name -> 'Jack' [0.52]
    Name -> 'Bob' [0.48]
    P -> 'with' [0.61]
    P -> 'under' [0.39]
    Det -> 'the' [0.41]
    Det -> 'a' [0.31]
    Det -> 'my' [0.28]
```



CFG 한글 응용연구(Blog)

Earley 차트 파싱 알고리즘 - (좌측 재귀처리 방식)

In [7]:
```
import nltk
nltk.parse.featurechart.demo( print_times = False, print_grammar = True,
                              parser = nltk.parse.featurechart.FeatureChartParser, sent = 'I saw a dog' )
```

```
Grammar with 18 productions (start state = S[])
    S[] -> NP[] VP[]
    PP[] -> Prep[] NP[]
    NP[] -> NP[] PP[]
    VP[] -> VP[] PP[]
    VP[] -> Verb[] NP[]
    VP[] -> Verb[]
    NP[] -> Det[pl=?x] Noun[pl=?x]
    NP[] -> 'John'
    NP[] -> 'I'
    Det[] -> 'the'
    Det[] -> 'my'
    Det[-pl] -> 'a'
    Noun[-pl] -> 'dog'
    Noun[-pl] -> 'cookie'
    Verb[] -> 'ate'
    Verb[] -> 'saw'
    Prep[] -> 'with'
    Prep[] -> 'under'
```

Earley 차트 파싱 알고리즘 - (좌측 재귀처리 방식)

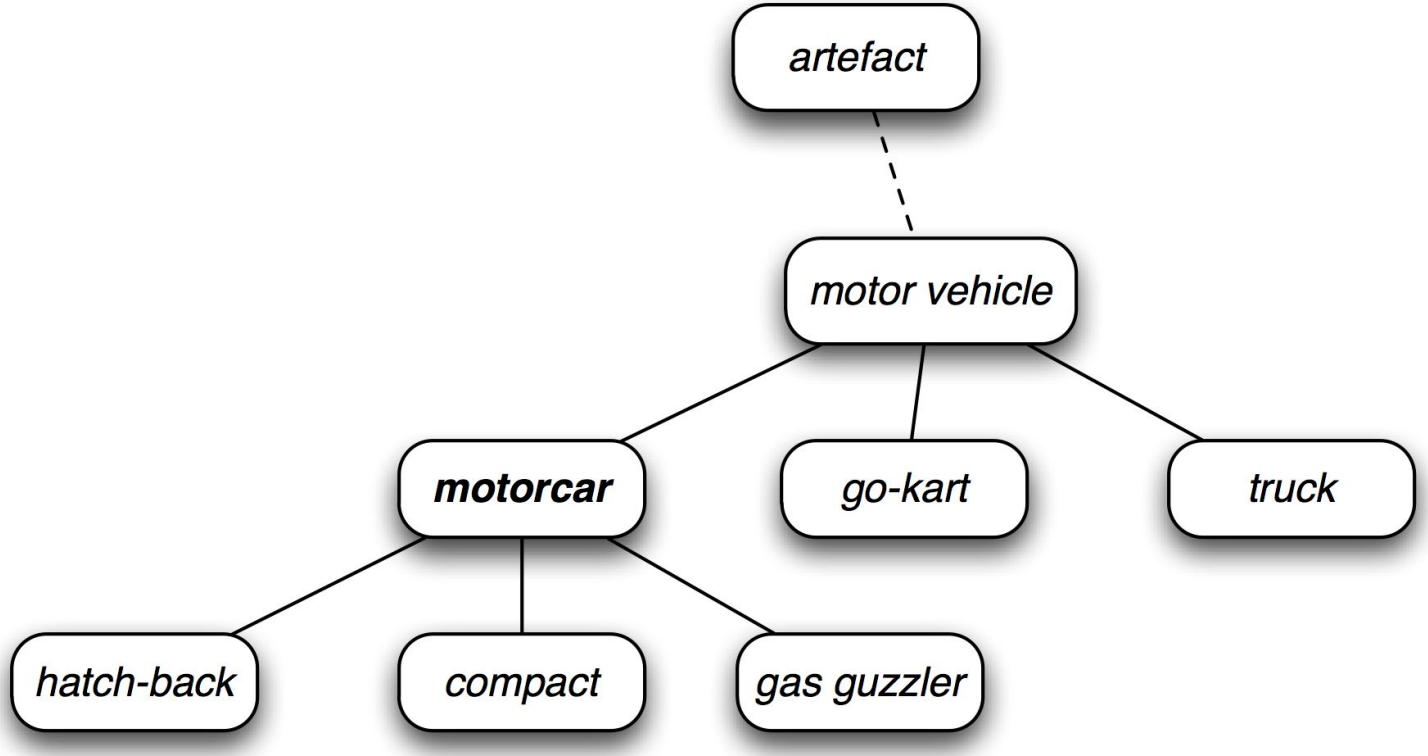
```
* FeatureChartParser
Sentence: I saw a dog
|. I .saw. a .dog.|
|[---]   .   .   .| [0:1] 'I'
|.   [---]   .   .| [1:2] 'saw'
|.   .   [---]   .| [2:3] 'a'
|.   .   .   [---]| [3:4] 'dog'
|[---]   .   .   .| [0:1] NP[] -> 'I' *
|[--->   .   .   .| [0:1] S[] -> NP[] * VP[] {}
|[--->   .   .   .| [0:1] NP[] -> NP[] * PP[] {}
|.   [---]   .   .| [1:2] Verb[] -> 'saw' *
|.   [--->   .   .| [1:2] VP[] -> Verb[] * NP[] {}
|.   [---]   .   .| [1:2] VP[] -> Verb[] *
|.   [--->   .   .| [1:2] VP[] -> VP[] * PP[] {}
|[-------]   .   .| [0:2] S[] -> NP[] VP[] *
|.   .   [---]   .| [2:3] Det[-pl] -> 'a' *
|.   .   [--->   .| [2:3] NP[] -> Det[pl=?x] * Noun[pl=?x] {?x: False}
|.   .   .   [---]| [3:4] Noun[-pl] -> 'dog' *
|.   .   [-------]| [2:4] NP[] -> Det[-pl] Noun[-pl] *
|.   .   [------->| [2:4] S[] -> NP[] * VP[] {}
|.   .   [------->| [2:4] NP[] -> NP[] * PP[] {}
|.   [-----------]| [1:4] VP[] -> Verb[] NP[] *
|.   [----------->| [1:4] VP[] -> VP[] * PP[] {}
|[===============]| [0:4] S[] -> NP[] VP[] *
(S[]
  (NP[] I)
  (VP[] (Verb[] saw) (NP[] (Det[-pl] a) (Noun[-pl] dog))))
```

Word DataBase

어휘망

Word Net

Word Net


artefact
motor vehicle
motorcar
go-kart
truck
hatch-back
compact
gas guzzler

Word Net 의 구성

1. **명사, 형용사, 부사**로 구분한 **동의어 계층적 네트워크 (Synset)** 를 활용
2. **동의관계, 반의관계, 상의관계, 하의관계, 분의 관계, 양식관계, 함의관계** 등의 **단어 관계망**을 정리한 DB
3. http://www.itdaily.kr/conference/image/Model-based.pdf

Word Net 유사도 측정

- **Path similarity**
  - 노드(node) 사이의 최소 거리(shortest path)를 활용

$$Sim_{path} = \frac{1}{length + 1}$$

여기에서 $length$ 는 두 단어(노드) 의 최소거리

- **Leacock & Chodorow Similarity**
  - 노드의 최소 거리 및 최대 깊이를 활용

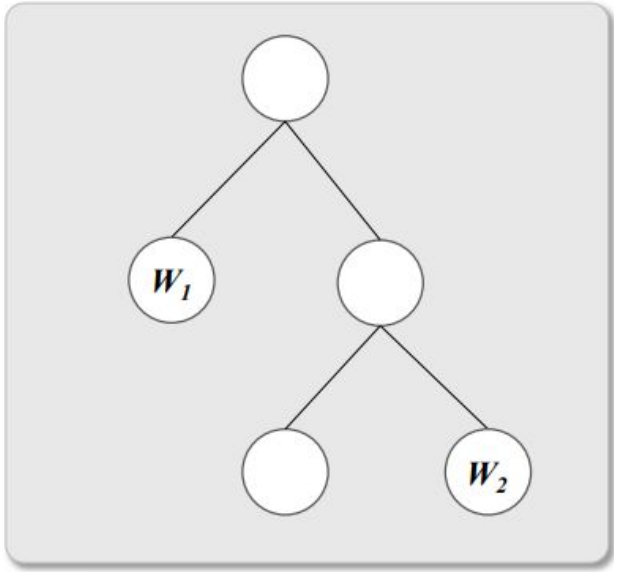
$$Sim_{LCH} = -\log \frac{length}{2 \cdot D}$$

여기에서 $length$는 두 단어(노드) 의 최소거리이며,
D 는 워드넷 계층에서 관찰되는 노드의 최대 깊이

- **Wu & Palmer Similarity**
  - 깊이(depth) 및 최소 상위 노드 활용

$$Sim_{WUP} = \frac{2 \times depth(LCS)}{depth(w_1) + depth(w_2)}$$

여기에서 LCS (least common subsummer) 는 두 단어가 가지는 가장 가까운 상위 노드



기존의 텍스트 유사도 측정

서로 다른 두 개의 텍스트에 대한 유사도 측정은 각각의 텍스트에 포함된 단어들을 추출하고 선택하고 그들의 유사도와, 그 단어의 텍스트 내의 중요도 가중치를 활용하여 측정한다

$$sim(T_1, T_2) = \frac{1}{2}\left(\left(\frac{\sum_{w \in \{T_1\}} (maxSim(w, T_2) \times idf(w))}{\sum_{w \in \{T_1\}} idf(w)}\right) + \left(\frac{\sum_{w \in \{T_2\}} (maxSim(w, T_1) \times idf(w))}{\sum_{w \in \{T_2\}} idf(w)}\right)\right),$$

여기에서

- ***idf(w)*** 는 역문서 빈도(*inverse document frequency*)로서,
  단어 w 가 전체 텍스트에서 얼마나 공통적으로 나타나는 지를 나타냄

$$idf(w) = log \frac{|D|}{|\{d \in D : w \in d\}|} = log \frac{\text{전체문서수}}{\text{단어}w\text{가포함된문서수}}$$

- ***max Sim(w, T)*** 는 단어 w 와 문서 T 에 속한 모든 단어와의 유사도중 가장 큰 값

$$max\ Sim(w, T) = \arg\max_{w' \in T} sim(w, w')$$

Word Net - 단어와 연결된 언어망 목록

In [1]:
```
from nltk.corpus import wordnet as wn
```

In [2]:
```
wn.synsets('dog')
```

Out[2]:
```
[Synset('dog.n.01'),
 Synset('frump.n.01'),
 Synset('dog.n.03'),
 Synset('cad.n.01'),
 Synset('frank.n.02'),
 Synset('pawl.n.01'),
 Synset('andiron.n.01'),
 Synset('chase.v.01')]
```

Word Net - 단어망 내용 살펴보기

```
In [3]: wn.synset('dog.n.01').examples()[0]

Out[3]: 'the dog barked all night'

In [4]: wn.synset('chase.v.01').examples()[1]

Out[4]: 'the dog chased the rabbit'

In [5]: dog = wn.synset('dog.n.01')
        dog.definition()

Out[5]: 'a member of the genus Canis (probably descended from the common wolf) that has been domesti
        cated by man since prehistoric times; occurs in many breeds'

In [6]: cat = wn.synset('cat.n.01')
        cat.definition()

Out[6]: 'feline mammal usually having thick soft fur and no ability to roar: domestic cats; wildcat
        s'
```

Word Net - 단어간 유사도 측정

```
In [9]: dog.path_similarity(cat)
Out[9]: 0.2

In [10]: dog.lch_similarity(cat)
Out[10]: 2.0281482472922856

In [11]: dog.wup_similarity(cat)
Out[11]: 0.8571428571428571
```

의미분석

Lemmatization

Lemmatization : 표제어 찾기

1. **Stemming**은 **단어**만 대상으로 속성을 부여
2. **Lemmatization(표제어)** 은 **문장내 품사** 판단
3. **구조분석** 후 **Lemmatization**를 추출하여 **단어의 품사와 의미**를 찾는다

출처 :
http://4four.us/article/2008/05/lemmatization

pip install -U pywsd - 문장에 따른 단어구분

In [14]:
```
sent = 'I went to the bank to deposit my money'
ambiguous = 'bank'

from pywsd.lesk import simple_lesk
answer = simple_lesk(sent, ambiguous, pos='n')
answer
```

```
Warming up PyWSD (takes ~10 secs)... took 2.7813966274261475 secs.
```

Out[14]:
```
Synset('depository_financial_institution.n.01')
```

In [15]:
```
answer.definition()
```

Out[15]:
```
'a financial institution that accepts deposits and channels the money into lending activities'
```

In [16]:
```
sent = 'I bank my money'
ambiguous = 'bank'
answer = simple_lesk(sent, ambiguous, pos='n')
answer
```

Out[16]:
```
Synset('savings_bank.n.02')
```

In [17]:
```
answer.definition()
```

Out[17]:
```
'a container (usually with a slot in the top) for keeping money at home'
```

한글DB

세종계획

표준국어 대사전

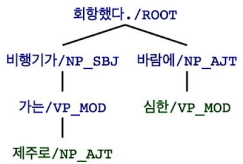
http://stdweb2.korean.go.kr/search/List_dic.jsp

http://andrewmatteson.name/psg_tree.htm

SyntaxNet Tree Generator (Korean)

Sentence to Parse: 제주로 가는 비행기가 심한 바람에 회항했다. Submit

Visualization



CoNLL-U Output

Details

1 POS Tagging | 2 Dependency Parsing | 3 Reassembly | 4 Semantic Analysis | D Debug Logs

ID	FORM	LEMMA	UPOSTAG	XPOSTAG	FEATS	HEAD	DEPREL
1	제주로	제주/NNG + 로/JKB	_	_	_	2	NP_AJT
2	가는	가/VV + 는/ETM	_	_	_	3	VP_MOD
3	비행기가	비행기/NNG + 가/JKS	_	_	_	6	NP_SBJ
4	심한	심하/VA + ㄴ/ETM	_	_	_	5	VP_MOD
5	바람에	바람/NNG + 에/JKB	_	_	_	6	NP_AJT
6	회항했다.	회항/NNG + 하/XSV + 았/EP + 다/EF + ./SF	_	_	_	0	ROOT

https://github.com/dsindex/syntaxnet

2018 ~ 2022 2차 세종계획 추진 (문체부, 국립국어원)

1. 국립국어원 정보나눔터 (link)
2. **한글 말뭉치 2007년**에 멈춘이유 (기사)
3. **2017년 11월** 2차 세종계획 추진 (기사)
4. 분석기사 (기사)
5. 기초 자료의 부족을 **딥러닝**으로 해결

Naive Bayes Classification

나이브 베이즈

NLTK

나이브 베이즈

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

나이브 베이즈

1. **구글 스팸메일 Filtering** 에 사용되는 이론
2. **머신러닝 이론**을 활용하여 **문장의 성격**을 분류
3. **단어의 우선순위**를 정하지 않아서 구조가 간단하고,
   성능도 우수하여 가볍게 제작 응용이 가능

실습코드

konlpy

Naver 리뷰 활용

1. **GitHub** 소스코드 (GitHub)
2. **Slide** 2015 PyCon (슬라이드)

Naver sentiment movie corpus v1.0

1. 데이터 출처: 네이버
2. 영화 당 100개의 140자평(이하 '리뷰')을 초과하지 않음
3. **ratings.txt** : 총 20만개 리뷰 (수집된 64만개 중 샘플링)
4. **ratings_train.txt** : 15만개
5. **ratings_test.txt** : 5만개
6. 각각 긍정/부정 리뷰의 비율을 동일하게 샘플링
7. 중립 리뷰는 제외

Naver sentiment movie corpus v1.0

In [13]:
```
! cat ./data/ratings_train.txt | head -n 5
```

```
id	document	label
9976970 아 더빙.. 진짜 짜증나네요 목소리	0
3819312 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
10265843	너무재밓었다그래서보는것을추천한다	0
9045019 교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
```

Naver 데이터 불러오기

In [14]:
```
def read_data(filename):
    with open(filename, 'r') as f:
        data = [line.split('\t') for line in f.read().splitlines()]
        data = data[1:]   # header 제외

    from random import randint
    random_data = [data[randint(1, len(data))]  for no in range(int(len(data)/50)) ]
    return random_data

train_data = read_data('./data/ratings_train.txt')
test_data  = read_data('./data/ratings_test.txt')
print('Train_data : {}\nsample     : {}'.format(len(train_data), train_data[:3]))
print('Test_data  : {}\nsample     : {}'.format(len(test_data), test_data[:3]))
```

```
Train_data : 3000
sample     : [['9287304', '가장 성룡스러운 영화 ㅋ 재밌음', '1'], ['9833070', 'CGV야 만우절 낚시하니까 조으디?', '0'], ['1223918', '그녀의 숭고한 삶에 경의를 표합니다.', '1']]
Test_data  : 1000
sample     : [['4354632', '마음이 훈훈해지는 영화', '1'], ['2399796', '쩝 분노를 느끼고 싶을때 잠이 안올때 볼만함', '0'], ['5558452', '오죽하면 내가 평점남기러 왔겠냐 ㅋㅋㅋㅋㅋ', '0']]
```

Token에 Tag 추가하기

In [15]:
```
%%time
from konlpy.tag import Twitter
pos_tagger = Twitter()

def tokenize(doc):
    result = ['/'.join(t) for t in pos_tagger.pos(doc, norm=True, stem=True)]
    return result

train_docs = [(tokenize(row[1]), row[2])    for row in train_data]
test_docs  = [(tokenize(row[1]), row[2])    for row in test_data]

from pprint import pprint
pprint(train_docs[:2])
```

```
[(['가장/Noun',
   '성룡/Noun',
   '스러운/Josa',
   '영화/Noun',
   'ㅋ/KoreanParticle',
   '재밌다/Adjective'],
  '1'),
 (['CGV/Alpha',
   '야/Exclamation',
   '만우절/Noun',
   '낚시/Noun',
   '하다/Verb',
   '좋다/Adjective',
   '?/Punctuation'],
  '0')]
CPU times: user 10.4 s, sys: 101 ms, total: 10.5 s
Wall time: 5.29 s
```

분석을 위한 Token 추출

In [4]:
```
tokens = [t   for d in train_docs
              for t in d[0]]
print("Token Total :{}\nSample : {}".format(
    len(tokens), tokens[:5]))
```

```
Token Total :43237
Sample : ['너무/Adverb', '느리다/Adjective', '뜨다/Verb', '리/Noun', '는/Josa']
```

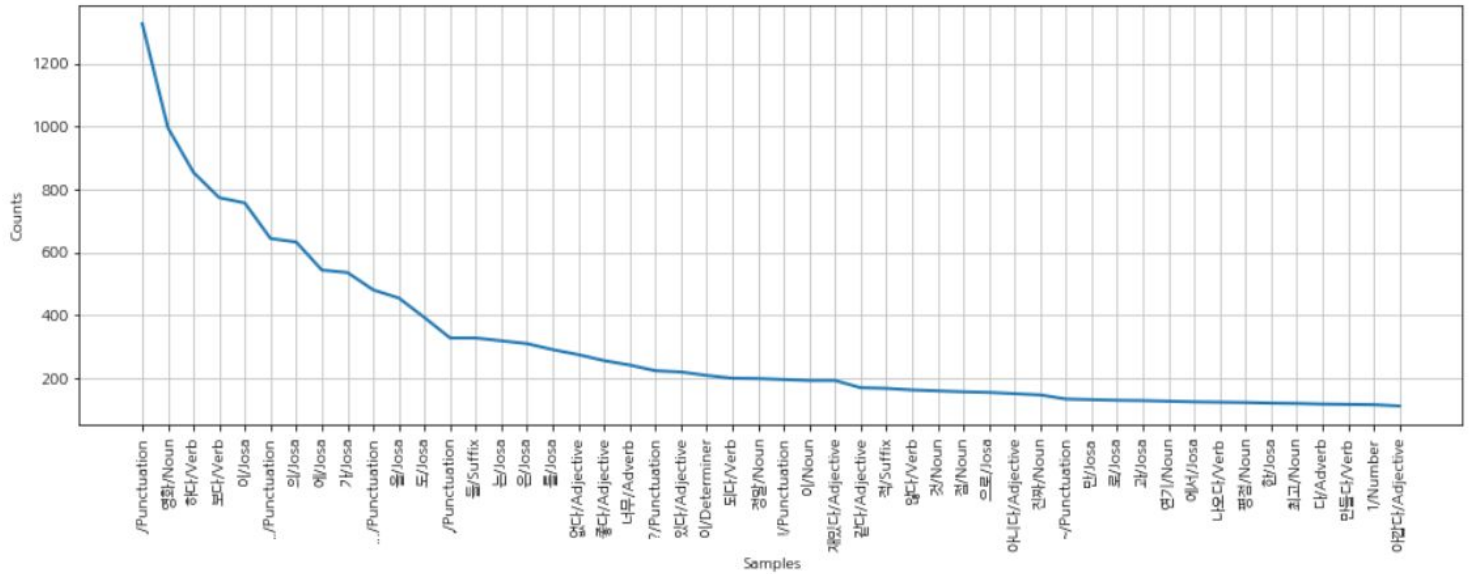
NLTK 객체 만들기

In [5]:
```
import nltk
text = nltk.Text(tokens, name='NMSC')

print("number of Token : {} \nunique Token    : {}\n".format(
    len(text.tokens), len(set(text.tokens))))
pprint(text.vocab().most_common(5))
```

```
number of Token : 43237
unique Token    : 6911

[('./Punctuation', 1327),
 ('영화/Noun', 996),
 ('하다/Verb', 854),
 ('보다/Verb', 774),
 ('이/Josa', 757)]
```

Token 빈도 시각화 - 상위 50개 빈도분석

In [6]:
```
%matplotlib inline
from matplotlib import rc
rc('font', family='NanumGothic')

import matplotlib.pyplot as plt
plt.figure(figsize=(16,5))
text.plot(50)
```



모델의 정확도/ 일반화를 높이는 방법들

1. Token 중 **빈도 상위 4,000 개**를 활용하여 모델을 만든다
2. **tf-idf 비중**을 포함하는 등의 다양한 후처리 방법이 가능

베이지안 분류모델 학습

In [8]:
```
selected_words = [f[0] for f in text.vocab().most_common(4000)]
selected_words[:5]
```

Out[8]:
```
['./Punctuation', '영화/Noun', '하다/Verb', '보다/Verb', '이/Josa']
```

In [9]:
```
%%time
def term_exists(doc):
    return {'exists({})'.format(word): (word in set(doc)) for word in selected_words}

train_docs = train_docs[:10000]
train_xy   = [(term_exists(d), c) for d, c in train_docs]
test_xy    = [(term_exists(d), c) for d, c in test_docs]
```

```
CPU times: user 14.1 s, sys: 596 ms, total: 14.7 s
Wall time: 14.7 s
```

In [10]:
```
%%time
classifiers = nltk.NaiveBayesClassifier.train(train_xy)
```

```
CPU times: user 14.7 s, sys: 7.84 ms, total: 14.8 s
Wall time: 14.7 s
```

베이지안 모델 살펴보기

```
In [11]: classifiers.labels()

Out[11]: ['0', '1']

In [18]: classifiers.show_most_informative_features(15)
```

```
Most Informative Features
        exists(쓰레기/Noun) = True              0 : 1      =     41.1 : 1.0
          exists(냐/Josa) = True              0 : 1      =     17.3 : 1.0
   exists(아깝다/Adjective) = True              0 : 1      =     15.9 : 1.0
         exists(눈물/Noun) = True              1 : 0      =     13.4 : 1.0
        exists(차라리/Noun) = True              0 : 1      =     12.6 : 1.0
   exists(더럽다/Adjective) = True              0 : 1      =     11.9 : 1.0
  exists(아름답다/Adjective) = True              1 : 0      =     11.6 : 1.0
          exists(돈/Noun) = True              0 : 1      =     11.2 : 1.0
  exists(짜증나다/Adjective) = True              0 : 1      =     11.2 : 1.0
         exists(최고/Noun) = True              1 : 0      =     10.9 : 1.0
          exists(짱/Noun) = True              1 : 0      =     10.8 : 1.0
  exists(재미있다/Adjective) = True              1 : 0      =     10.6 : 1.0
        exists(0/Number) = True              0 : 1      =     10.5 : 1.0
exists(ㅡㅡ/KoreanParticle) = True              0 : 1      =      9.8 : 1.0
   exists(멋지다/Adjective) = True              1 : 0      =      9.6 : 1.0
```

베이지안 모델의 활용 - 0 : 부정리뷰 , 1 : 긍정리뷰

In [14]:
```
review = """영화가 졸잼 굿"""
```

In [15]:
```
review = tokenize(review)     # 문법 Tag 추가한 객체로 변환
review
```

Out[15]:
```
['영화/Noun', '가/Josa', '졸잼/Noun', '굿/Noun']
```

In [16]:
```
review = term_exists(review) # 기준 용어들이 포함여부 판단
for k, v in review.items():
    if v == True:
        print("{} = {}".format(k, v))
```

```
exists(영화/Noun) = True
exists(가/Josa) = True
exists(굿/Noun) = True
exists(졸잼/Noun) = True
```

In [17]:
```
classifiers.classify(review)  # 분류모델 평가
```

Out[17]:
```
'1'
```