딥러닝을 활용한 자연어 분석

구조/ 의미분석

김용범

무영인터내쇼날

Jupyter Notebook 소스보기

http://nbviewer.jupyter.org/github/YongBeomKim/nltk_tutorial/blob/master/03.meaning.ipynb

자연어 분석과정

1. **형태론(Morphology) :** 단어와 형태소를 연구
2. **통사론(syntax) :** 문법적 구조분석(Passing)
3. **의미론(Senmantics) :** 단어 의미차이 **ex) 뉘앙스, 톤, 의도(긍/부정)**

Parse Tree & Parsing

문장의 구조

Parsing

Parse Tree

In [5]:
```
%%time
text = '민병삼 대령의 항명행위로 초치했다'

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

```
[('민병삼', 'Noun'), ('대령', 'Noun'), ('의', 'Josa'), ('항', 'Noun'), ('명', 'Suffix'), ('행위', 'Noun'), ('로', 'Josa'), ('초치', 'Noun'), ('하다', 'Verb')]
CPU times: user 44.7 ms, sys: 7.21 ms, total: 51.9 ms
Wall time: 25.2 ms
```

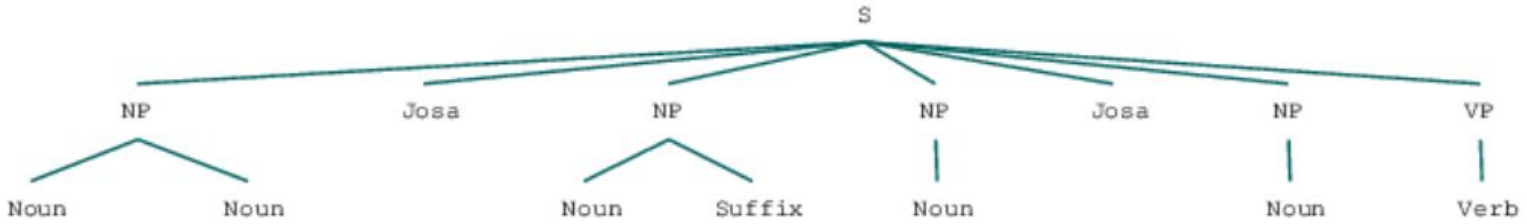
In [2]:
```
from nltk import RegexpParser

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 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
```

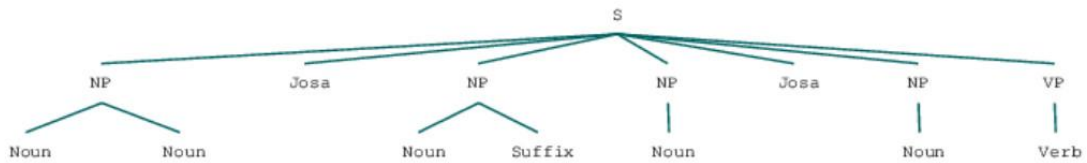
Parsing 분석 이론들

1. **CFG (Context Free Grammer)** : 중심 구조분석으로 '노암촘스키'가 제안
2. **Earley 차트 파싱 알고리즘 :** 좌측 재귀적 구조분석
3. **ATIS** 문법 : 공항 안내시스템 개발용 문법규칙

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

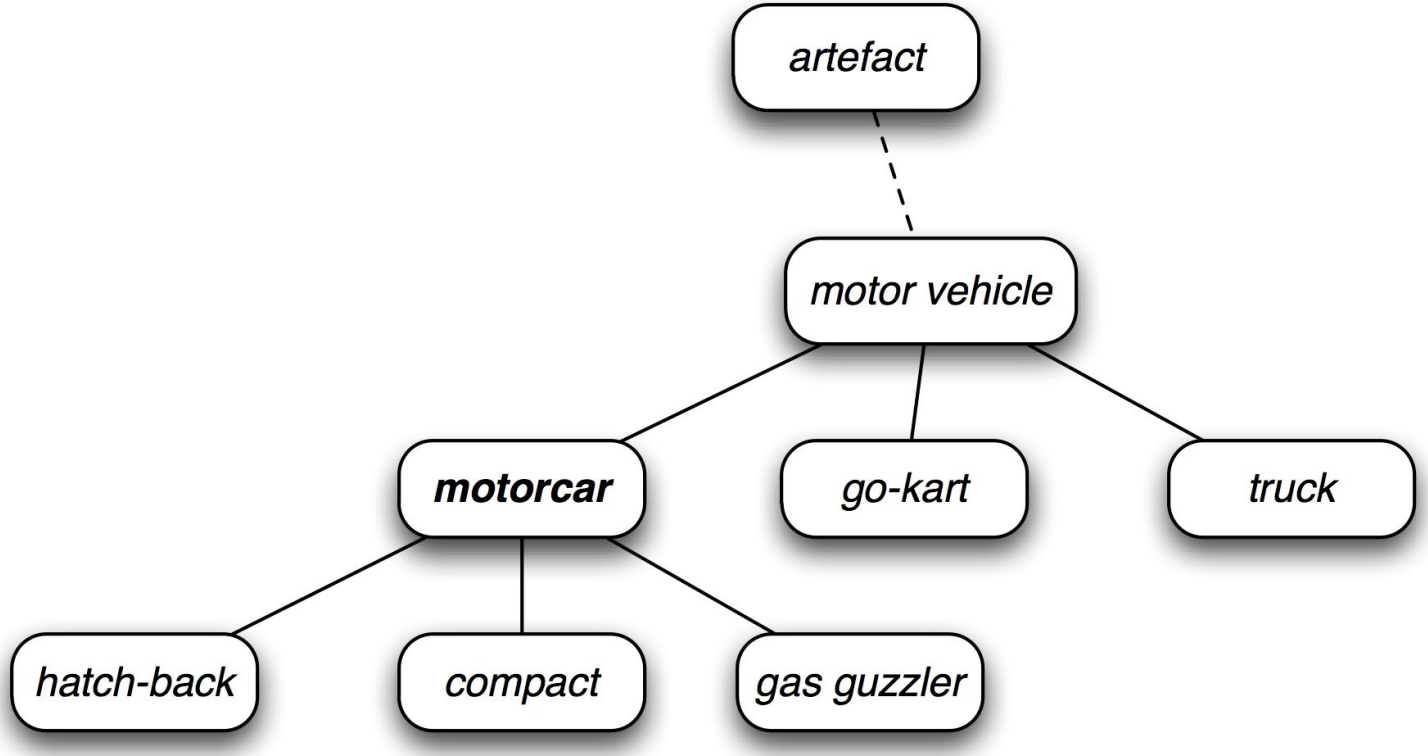
어휘망

Word Net

감정분석사전 lexicon (어휘목록|사서) 활용

1. labMT (10,000단어 분석)
2. Warringer (13,915단어 분석)
3. OpinionFinder's Subjectivity Lexic (8221단어 분석)
4. ANEW (1034단어 분석) : Affective Norms for English Words
5. AFINN (2477단어 분석) : Finn Arup Nielson 에 의한 분류
6. Balance Affective (277 단어) : 1(긍정), 2(부정), 3(불안정), 4(중립)
7. BAWL (2200단어 분석) : Berlin Affective Word List Reloaded
8. BFAN (210단어로 구성) : Bilingual Finnish Affective Norms
9. CDGE : Compass DeRose Guide to Emotion Words
10. DAL : Dictionary of Affect in Language
11. WDAL : Whissell's Dictionary of Affect in Language

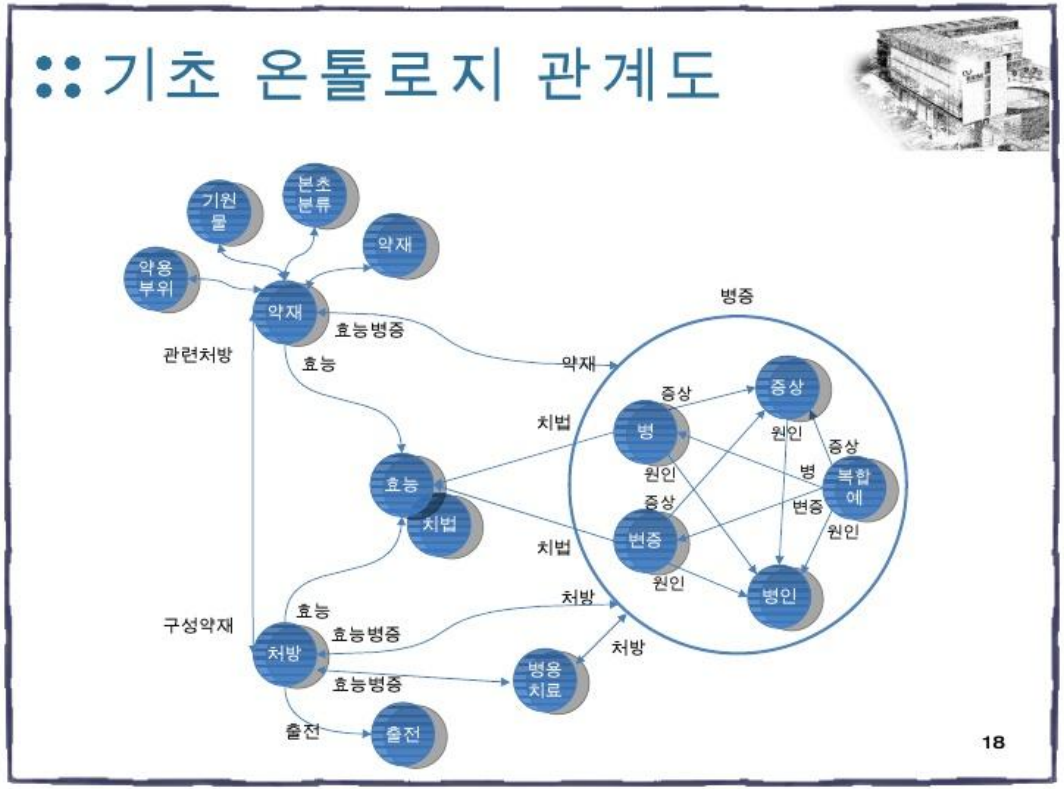
Word Net


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

Word Net 을 활용한 의미판별

1. 어휘들의 개념을 바탕으로 네트워크를 구축한 지식DB
2. **개념 DB**는 단어별 정의를 기록하고
3. **시소러스**(동의어 반의어 사전)**DB** 는 용어간 관계를 나타내기 위해 **상하위관계, 동등관계, 부분-전체관계, 연관관계, 사례관계** 등을 활용한다
4. **워드넷**에서는 **동의관계, 반의관계, 상의관계, 하의관계, 분의 관계, 양식관계, 함의관계** 등을 정리한 DB

On Tology


기초 온톨로지 관계도
기원물
본초본류
약재
약용부위
약재
효능병증
관련처방
효능
약재
병증
증상
증상
치법
병
원인
증상
병
복합예
원인
증상
효능
변증
원인
치법
치법
변증
원인
병인
처방
구성약재
효능
효능병증
처방
처방
병용치료
효능병증
출전
출전
18

On Tology

1. 철학 학문 가운데 **형이상학**의 대표적인 세부 학제
2. **존재로서의 존재**(being qua being) 를 다루는 학제로 정의
3. 컴퓨터 과학에서는 이를 응용하여 **컴퓨터 공간 상에서 데이터** 및 데이터들을 아우르는 **개념**들에 관한 **존재론** 을 의미

한글DB

세종계획

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 많이 사용

NLTK 영화리뷰 긍정/ 부정

1. **긍정리뷰** 1000개, **부정리뷰** 1000개 Text 파일
2. nltk.**NaiveBayesClassifier**() 로 학습
3. **학습 모델**을 활용하여 Text **긍정/ 부정**을 판단한다

NLTK 영화리뷰 긍정/ 부정 수집

In [8]:
```
from nltk.corpus import movie_reviews

print('Category : {}\n  Length : {}'.format(
    movie_reviews.categories(),
    len(movie_reviews.fileids(movie_reviews.categories()[1])) ))

print('\npos_file_list : {} \nneg_file_list : {}'.format(
    movie_reviews.fileids(movie_reviews.categories()[0])[:5],
    movie_reviews.fileids(movie_reviews.categories()[1])[:5] ))
```

```
Category : ['neg', 'pos']
  Length : 1000

pos_file_list : ['neg/cv000_29416.txt', 'neg/cv001_19502.txt', 'neg/cv002_17424.txt', 'neg/c
v003_12683.txt', 'neg/cv004_12641.txt']
neg_file_list : ['pos/cv000_29590.txt', 'pos/cv001_18431.txt', 'pos/cv002_15918.txt', 'pos/c
v003_11664.txt', 'pos/cv004_11636.txt']
```

In [9]:
```
# ! cat /home/markbaum/nltk_data/corpora/movie_reviews/neg/cv435_24355.txt
example = """a couple of criminals ( mario van peebles and loretta devine ) move into a rich
hopes of conning them out of their jewels . however... """
```

NLTK 영화리뷰 분류별 1개의 객체로 묶는다

In [10]:
```
import nltk, random
docs = [(list(movie_reviews.words(fid)), cat)
        for cat in movie_reviews.categories()    # ['neg', 'pos']
        for fid in movie_reviews.fileids(cat)]   # 'neg/cv000_29416.txt', ....

random.shuffle(docs)
all_tokens     = nltk.FreqDist(x.lower()    for  x  in  movie_reviews.words())
token_features = list(all_tokens.keys())[:2000]
all_tokens
```

Out[10]:
```
FreqDist({',': 77717, 'the': 76529, '.': 65876, 'a': 38106, 'and': 35576, 'of': 34123, 'to':
31937, "'": 30585, 'is': 25195, 'in': 21822, ...})
```

NLTK 학습한 뒤 모델을 검증한다

In [11]:
```
def doc_features(docs):
    doc_words = set(docs)
    features  = {'word( %s )'%word : (word in doc_words)
                 for word in token_features }
    return features
```

In [12]:
```
feature_sets = [(doc_features(d), c) for (d,c) in docs]
train_sets, test_sets = feature_sets[100:], feature_sets[:100]

classifiers = nltk.NaiveBayesClassifier.train(train_sets)
print('Accuracy :', nltk.classify.accuracy(classifiers, test_sets))
classifiers.show_most_informative_features()
```

```
Accuracy : 0.81
Most Informative Features
   word( unimaginative ) = True              neg : pos    =      7.8 : 1.0
         word( suvari ) = True               neg : pos    =      7.1 : 1.0
         word( shoddy ) = True               neg : pos    =      7.1 : 1.0
           word( mena ) = True               neg : pos    =      7.1 : 1.0
      word( schumacher ) = True              neg : pos    =      7.1 : 1.0
       word( atrocious ) = True              neg : pos    =      6.3 : 1.0
         word( singers ) = True              pos : neg    =      6.3 : 1.0
         word( justin ) = True               neg : pos    =      5.9 : 1.0
```

나이브 베이즈

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 내용을 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
```

Token 만 추출 (학습용)

In [16]:
```
tokens = [t   for d in train_docs
              for t in d[0]]
print(len(tokens))
```

```
43925
```

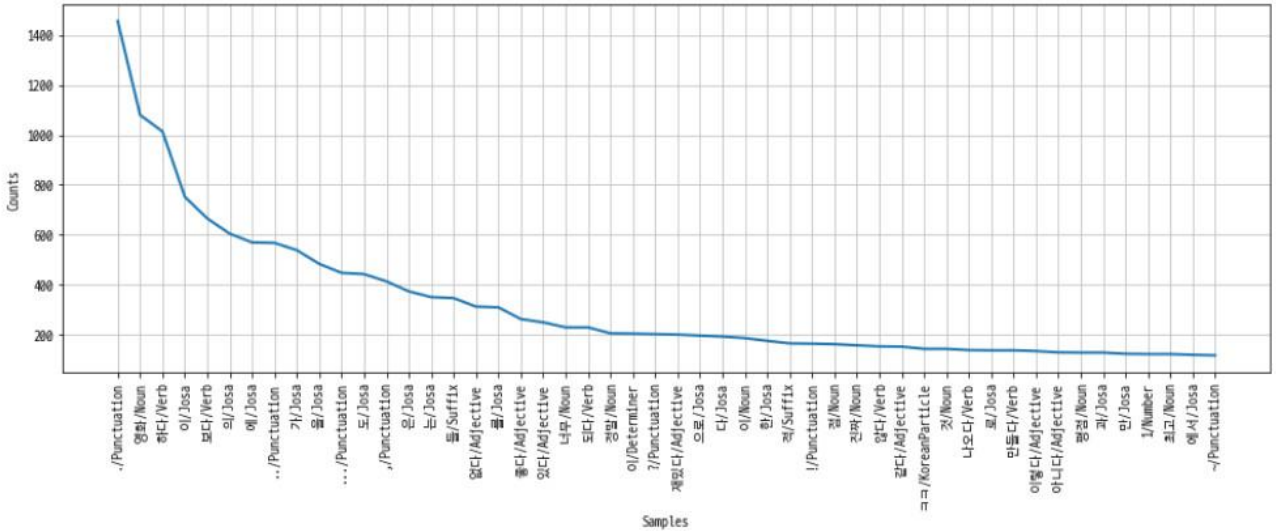
NLTK 로 Token 빈도분석 객체 만들기

In [17]:
```
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(10))
```

```
number of Token : 43925
unique Token    : 6548

[('./Punctuation', 1456),
 ('영화/Noun', 1080),
 ('하다/Verb', 1014),
 ('이/Josa', 752),
 ('보다/Verb', 666),
 ('의/Josa', 605),
 ('에/Josa', 570),
 ('../Punctuation', 568),
 ('가/Josa', 539),
 ('을/Josa', 484)]
```

Token 빈도 시각화 - 상위 50개 빈도분석

In [18]:
```
%matplotlib inline
from matplotlib import font_manager, rc
# 한글폰트를 별도로 불러온다
font_fname = './data/D2Coding.ttf'
font_name = font_manager.FontProperties(fname=font_fname).get_name()
rc('font', family=font_name)

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



모델의 정확도/ 일반화를 높이는 방법들

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

베이지안 분류모델 만들기

In [19]:
```
%%time
selected_words = [f[0] for f in text.vocab().most_common(2000)]
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 7.18 s, sys: 314 ms, total: 7.49 s
Wall time: 7.51 s
```

In [20]:
```
%%time
classifier = nltk.NaiveBayesClassifier.train(train_xy)
print('네이버 긍부정 모델의 Accuracy : {}'.format(
    nltk.classify.accuracy(classifier, test_xy)))
classifier.show_most_informative_features(10)
```

```
네이버 긍부정 모델의 Accuracy : 0.792
Most Informative Features
        exists(쓰레기/Noun) = True                0 : 1      =     20.8 : 1.0
  exists(재미없다/Adjective) = True                0 : 1      =     16.8 : 1.0
         exists(별로/Noun) = True                0 : 1      =     15.1 : 1.0
         exists(점도/Noun) = True                0 : 1      =     14.8 : 1.0
         exists(찍다/Verb) = True                0 : 1      =     13.4 : 1.0
         exists(잔잔/Noun) = True                1 : 0      =     13.3 : 1.0
         exists(명작/Noun) = True                1 : 0      =     11.5 : 1.0
         exists(수작/Noun) = True                1 : 0      =     10.7 : 1.0
         exists(실망/Noun) = True                0 : 1      =     10.7 : 1.0
          exists(적/Noun) = True                1 : 0      =     10.0 : 1.0
CPU times: user 11.7 s, sys: 3.99 ms, total: 11.7 s
```

모델의 활용

0 : 부정리뷰 , 1 : 긍정리뷰

In [24]:
```
review = """나는 정말 재미있었음
정말 재미있게 봤고 액션신도 멋짐
왜 재미없는지는 모르겠다"""
```

In [25]:
```
review = tokenize(review)     # 문법 Tag 추가한 객체로 변환
review = term_exists(review) # 기준 용어들이 포함여부 판단
classifiers.classify(review)  # 분류모델로 평가
```

Out[25]:
```
'1'
```