

노가다 없는 텍스트 분석을 위한 한국어 NLP

파이콘 코리아 2017

김현중 (soy.lovit@gmail.com)

Back to the Basic

노가다 없는 텍스트 분석을 위한 한국어 NLP

Hyunjoong Kim

soy.lovit@gmail.com

- KoNLPy는 Python에서 사용할 수 있는 한국어 자연어처리 패키지

```
from konlpy.tag import Kkma
```

```
kkma = Kkma()
```

```
print(kkma.nouns(u'질문이나 건의사항은 깃헙 이슈 트래커에 남겨주세요.'))
```

[질문, 건의, 건의사항, 사항, 깃헙, 이슈, 트래커]

```
print(kkma.pos(u'오류보고는 실행환경, 에러메세지와함께 설명을 최대한상세히!^^'))
```

[(오류, NNG), (보고, NNG), (는, JX), (실행, NNG), (환경, NNG), (,, SP),
(에러, NNG), (메세지, NNG), (와, JKM), (함께, MAG), (설명, NNG), (을, JKO),
(최대한, NNG), (상세히, MAG), (!, SF), (^^, EMO)]

미등록단어 문제

- 새롭게 만들어진 단어들은 인식이 잘 되지 않습니다

```
from konlpy.tag import Kkma, Twitter
kkma = Kkma()
twitter = Twitter()

kkma.pos('파이콘에서 엔엘피이야기를 합니다')
```

파이/NNG, 콘/NNG, 에서/JKM, 엔엘피/NNG, 이야기/NNG, 를/JKO, 하/VV, 하시다/EF

```
twitter.pos('파이콘에서 엔엘피이야기를 합니다')
```

파/Noun, 이콘/Noun, 에서/Josa, 엔/Josa, 엘피/Noun, 이야기/Noun, 를/Josa, 합/Verb, 시/PreEomi, 다/Eomi

미등록단어 문제

- 새롭게 만들어진 단어들은 인식이 잘 되지 않습니다

```
from konlpy.tag import Kkma, Twitter
kkma = Kkma()
twitter = Twitter()

kkma.pos('너무너무너무는 아이오아이의 노래예요')
```

너무/MAG, 너무너무/MAG, 는/JX, 아이오/NNG, 아이/NNG, 의/JKG, 노래/NNG, 예/JKM, 요/JX

```
twitter.pos('너무너무너무는 아이오아이의 노래예요')
```

너무/Noun, 너무/Noun, 너무/Noun, 는/Josa, 아이오/Noun, 아이/Noun, 의/Josa, 노래/Noun, 예요/Josa

미등록단어 문제

- 이를 해결하기 위하여 **사용자 사전**을 추가하여 사용합니다
 - 주로 이 작업은 품사 판별을 한 뒤, 빈도수가 높은 단어들 중에서 잘못 처리된 단어를 고르는 방식입니다
 - 이 **반복적/노동집약적인 과정을 최대한 자동화** 하는게 목표입니다

우리가 다룰 이야기

- **사용자 사전을** 사람이 만들지 말고, **데이터 기반으로 추출**할 것입니다
 - 여러 단어 추출 기법들 중 **cohesion**에 대하여 살펴봅니다.
- 단어 중에서도 **명사**는 따로 **추출**할 것입니다
 - 명사는 새로운 단어가 가장 많이 생성되고 사용도 가장 많이 되는 품사입니다

Part 1: 단어 추출 + 토큰나이징

Part 2: 명사 추출

부록 : KoNLPy + 사용자사전

단어 추출 + 토큰나이징

<https://github.com/lovit/soynlp>

단어 추출

- 우리도 뉴스나 글을 읽으면서 새로운 단어들을 쉽게 학습합니다
- **단어**는 (특히 명사는) **경계에 특징**이 있습니다.
우리는 이를 이용하여 통계 기반으로 단어를 추출합니다
- 단, 1음절 단어는 제외합니다.
단어 길이가 1이면 봐도 무슨 말인지 해석하기 힘듭니다 (= 모호합니다)

Cohesion (Character n-gram)

- 한국어의 의미를 지니는 단어 (명사/동사/형용사/부사)는 **어절 왼쪽**에 있습니다

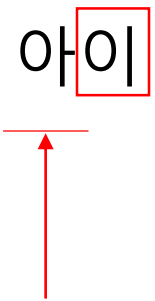
짜장면/명사 + **을**/조사

먹/동사 + **었어**/어미

Cohesion (Character n-gram)

- 맥락이 충분히 주어지지 않으면 다음에 등장할 글자의 확률이 작습니다
 - 한글자 ('아')는 매우 모호한 문맥입니다

아이



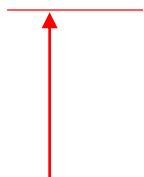
한글자는 특별한 문맥을
가지기가 어렵습니다

- > 아니 17.15 %
- > **아이 14.86 %**
- > 아시 8.06 %
- > 아닌 4.74 %
- > 아파 4.43 %
- > 아직 3.85 %
- ...

Cohesion (Character n-gram)

- 맥락이 충분히 주어지지 않으면 다음에 등장할 글자의 확률이 작습니다

아이오



어떤 경우는 두 글자라 하더라도
다양한 맥락에서 등장하기도 합니다

- > 아이폰 16.60 %
- > 아이들 13.37 %
- > 아이디 9.66 %
- > 아이돌 6.77 %
- > 아이뉴 6.77 %
- > **아이오 6.53 %**

...

Cohesion (Character n-gram)

- Subword 다음에 등장할 글자가 쉽게 예상된다면 (확률이 높다면) 아직 단어가 끝나지 않았다는 의미입니다

아이오아

문맥이 명확해 질수록
이전 단어 → 다음 글자 확률이
높아집니다

> **아이오아 87.95 %**

> 아이오닉 7.49 %

> 아이오와 3.26 %

> 아이오빈 0.65 %

> 아이오페 0.33 %

> 아이오케 0.33 %

Cohesion (Character n-gram)

- Subword 다음에 등장할 글자가 쉽게 예상된다면 (확률이 높다면) 아직 단어가 끝나지 않았다는 의미입니다

> 아이오아이 100.00 %

아이오아이

문맥이 확실하면 다음글자의
등장 확률이 높습니다

Cohesion (Character n-gram)

- 단어의 경계를 넘으면 다음 글자에 대한 확률이 다시 작아집니다

아이오아이는

단어 경계 뒤에는 다양한
조사/어미 들이 등장합니다

- > 아이오아이의 31.97 %
- > **아이오아이는 27.21 %**
- > 아이오아이와 13.61 %
- > 아이오아이가 12.24 %
- > 아이오아이에 9.52 %
- > 아이오아이까 1.36 %

...

Cohesion (Character n-gram)

- 단어의 점수(cohesion)를 아래처럼 정의해 봅니다

$$cohesion(c_{1:n}) = \sqrt[n-1]{\prod_{i=1}^{n-1} P(c_{1:i+1} | c_{1:i})}$$

$$P(c_{1:2} | c_1) = \frac{\#c_{1:2}}{\#c_1}$$

$$\begin{aligned} cohesion('아이오아이') = & \{ p(\text{아} \rightarrow \text{아이}) * \\ & p(\text{아이} \rightarrow \text{아이오}) * \\ & p(\text{아이오} \rightarrow \text{아이오아}) * \\ & p(\text{아이오아} \rightarrow \text{아이오아이}) \\ & \}^{1/(5-1)} \end{aligned}$$

학습은 오로지
string count

Cohesion (Character n-gram)

- 어절의 왼쪽부터 subword의 빈도수를 세어봅니다.

```
docs = ['파이콘에서 발표를 합니다',  
        ... ]  
  
from collections import defaultdict  
count= defaultdict(lambda: 0)  
  
for doc in docs:  
    for word in doc.split():  
        n = len(word)  
        for e in range(1, n+1):  
            count[word[:e]] += 1
```

Cohesion (Character n-gram)

- Subword의 빈도수와 $P(AB|A)$ 를 계산해 봅시다

```
word = '아이오아이는'
n = len(word)

for e in range(2, n+1):
    w = word[:e]
    f = count[w]
    p = f/count[:e-1]

    print('{:6}, f={}, p={:.2}'.
          format(w, f, p))
```

| | |
|---------|----------------|
| 아이, | f=4910, p=0.15 |
| 아이오, | f=307, p=0.06 |
| 아이오아, | f=270, p=0.88 |
| 아이오아이, | f=270, p=1.00 |
| 아이오아이는, | f=40, p=0.15 |

Cohesion (Character n-gram)

- 단어의 점수(cohesion)를 아래처럼 구현해 봅니다

```
def cohesion(w):  
    return pow(count[w]/count[w[0]],  
               1/(len(w)-1))
```

```
word = '아이오아이가'  
n = len(word)
```

```
for e in range(2, n+1):  
    w = word[:e]  
    f = count[w]  
    s = cohesion(w)  
    print('{:6}, f={}, s={:.2}'.  
          format(w, f, s))
```

아이, f=4910, s=0.15

아이오, f=307, s=0.10

아이오아, f=270, s=0.20

아이오아이, f=270, s=0.30 **단어로 선택**

아이오아이가, f=18, s=0.22

$$\frac{\#c_{1:4}}{\#c_1} = \frac{\#c_{1:2}}{\#c_1} * \frac{\#c_{1:3}}{\#c_{1:2}} * \frac{\#c_{1:4}}{\#c_{1:3}}$$

Cohesion (Character n-gram)

- 단어의 점수(cohesion)를 아래처럼 구현해 봅니다

```
def cohesion(w):  
    return pow(count[w]/count[w[0]],  
               1/(len(w)-1))
```

```
word = '아이오아이가'  
n = len(word)
```

```
for e in range(2, n+1):  
    w = word[:e]  
    f = count[w]  
    s = cohesion(w)  
    print('{:6}, f={}, s={:.2}'.  
          format(w, f, s))
```

아이, f=4910, s=0.15

아이오, f=307, s=0.10

아이오아, f=270, s=0.20

아이오아이, f=270, s=0.30 **단어로 선택**

아이오아이가, f=18, s=0.22

$$\frac{\#c_{1:4}}{\#c_1} = \frac{\#c_{1:2}}{\#c_1} * \frac{\#c_{1:3}}{\#c_{1:2}} * \frac{\#c_{1:4}}{\#c_{1:3}}$$

Tokenizer

- 단어를 잘 인식할 수 있다면 토크나이징도 쉽게 할 수 있습니다
 - 토크나이징은 여러 개의 단어로 이뤄진 문장/어절에서 단어를 구분하는 것
- 띄어쓰기 오류 정도에 따라 다른 토크나이징 전략을 사용할 수 있습니다

L-Tokenizer

- 띄어쓰기가 잘 되어 있다면, 어절의 **왼쪽에서**부터 단어의 **점수가 가장 큰 subword**를 기준으로 어절을 나눕니다

```
def ltokenize(w):  
    n = len(w)  
    if n <= 2: return (w, '')  
    tokens = []  
    for e in range(2, n+1):  
        tokens.append(w[:e], w[e:], cohesion(w[:e]))  
    tokens = sorted(tokens, key=lambda x:-x[2])  
    return tokens[0][:2]
```

```
sent = '뉴스의 기사를 이용했던 예시입니다'  
for word in sent.split():  
    print( ltokenize(word) )
```

('뉴스', '의')

('기사', '를')

('이용', '했던')

('예시', '입니다')

Max Score Tokenizer

- 띄어쓰기가 잘 되어있지 않다면 **아는 단어부터** 잘라내면 됩니다

```
cohesions = {'파스': 0.3, '파스타': 0.7, '좋아요': 0.2, '좋아': 0.5}
```

```
score = lambda x: cohesions.get(x, 0)
```

```
tokenize('파스타가좋아요')
```

[('파스', 0, 2, 0.3),
(('파스타', 0, 3, 0.7),
(('스타', 1, 3, 0),
(('스타가', 1, 4, 0),
(('타가', 2, 4, 0),
(('타가중', 2, 5, 0),
(('가중', 3, 5, 0),
(('가중아', 3, 6, 0),
(('좋아', 4, 6, 0.5),
(('좋아요', 4, 7, 0.2),
(('아요', 5, 7, 0)]

Subword 별 score 계산
(subword, begin, end, score)

[('파스타', 0, 3, 0.7),
(('좋아', 4, 6, 0.5),
(('파스', 0, 2, 0.3),
(('좋아요', 4, 7, 0.2),
(('스타', 1, 3, 0),
(('스타가', 1, 4, 0),
(('타가', 2, 4, 0),
(('타가중', 2, 5, 0),
(('가중', 3, 5, 0),
(('가중아', 3, 6, 0),
(('아요', 5, 7, 0)]

Score 기준으로 정렬

[('파스타', 0, 3, 0.7),
(('좋아', 4, 6, 0.5),
~~(('파스', 0, 2, 0.3),~~
(('좋아요', 4, 7, 0.2),
~~(('스타', 1, 3, 0),~~
~~(('스타가', 1, 4, 0),~~
~~(('타가', 2, 4, 0),~~
~~(('타가중', 2, 5, 0),~~
(('가중', 3, 5, 0),
(('가중아', 3, 6, 0),
(('아요', 5, 7, 0)]

최고점수의 단어 선택,
위치가 겹치는 단어 제거

Max Score Tokenizer

- 띄어쓰기가 잘 되어있지 않다면 **아는 단어부터** 잘라내면 됩니다

```
cohesions = {'파스': 0.3, '파스타': 0.7, '좋아요': 0.2, '좋아': 0.5}
```

```
score = lambda x: cohesions.get(x, 0)
```

[파스타]가좋아요

[('파스타', 0, 3, 0.7),
('좋아', 4, 6, 0.5),
~~('파스', 0, 2, 0.3),~~
('좋아요', 4, 7, 0.2),
~~('스타', 1, 3, 0),~~
~~('스타가', 1, 4, 0),~~
~~('타가', 2, 4, 0),~~
~~('타가좋', 2, 5, 0),~~
('가좋', 3, 5, 0),
('가좋아', 3, 6, 0),
('아요', 5, 7, 0)]



[파스타]가**[좋아]**요

('파스타', 0, 3, 0.7),
('좋아', 4, 6, 0.5),
~~('파스', 0, 2, 0.3),~~
~~('좋아요', 4, 7, 0.2),~~
~~('스타', 1, 3, 0),~~
~~('스타가', 1, 4, 0),~~
~~('타가', 2, 4, 0),~~
~~('타가좋', 2, 5, 0),~~
~~('가좋', 3, 5, 0),~~
~~('가좋아', 3, 6, 0),~~
~~('아요', 5, 7, 0)]~~



[파스타, 가, 좋아, 요]

Max Score Tokenizer

- 확실히 아는 단어라면 scores에 최고 점수를 줄 수 있습니다

```
cohesions = {'파스': 0.3, '파스타': 0.7, '좋아요': 0.2, '좋아': 0.5, '아이오아이': 1.0}
score = lambda x: cohesions.get(x, 0)
```

soynlp

- Cohesion, Branching Entropy 를 포함한, 단어 추출에 관련된 함수들을 github에 구현해 두었습니다.

```
from soynlp import DoublespaceLineCorpus
from soynlp.word import WordExtractor

corpus = DoublespaceLineCorpus(fname, iter_sent=True)
word_extractor = WordExtractor(corpus, min_count=10)
words = word_extractor.extract()
```

soynlp

- extract()는 단어 경계와 관련된 점수들이 계산되어 있는 dict를 return 합니다

```
words = word_extractor.extract()  
words['드라마']
```

```
Scores(cohesion_forward=0.6093651029086764,  
       cohesion_backward=0.5282705437953743,  
       left_branching_entropy=3.6583115265560924,  
       right_branching_entropy=3.675624807575614,  
       left_accessor_variety=128,  
       right_accessor_variety=136,  
       leftside_frequency=2375,  
       rightside_frequency=1284)
```

soynlp

- Tokenizer 역시 구현되어 있습니다

```
from soynlp.tokenizer import LTokenizer
```

```
scores = {w:s.cohesion_forward for w, s in words.items()}
```

```
tokenizer = LTokenizer(scores=scores)
```

```
tokenizer.tokenize('뉴스의 기사를 이용했던 예시입니다')
```

```
['뉴스', '의 ', '기사', '를 ', '이용', '했던', '예시', '입니다']
```

soynlp

- 사전이 없이도 데이터 기반으로 토큰라이저를 학습하면 잘 작동합니다
 - 현재 분석 중인 데이터에 해당 알고리즘을 적용한 예시입니다

```
from soynlp.tokenizer import MaxScoreTokenizer

scores = {w:s.cohesion_forward for w, s in words.items()}
tokenizer = MaxScoreTokenizer(scores=scores)
tokenizer.tokenize('맛있는짜파게티파스타일식초밥소바김볶다먹고싶어라일단김밥천국으로고고')
```

```
['맛있', '는', '짜파게티', '파스타', '일식', '초밥', '소바', '김볶', '다', '먹고', '싶어', '라',  
'일단', '김밥천국', '으로', '고고']
```

명사 추출

<https://github.com/lovit/soynlp>

명사

- **지금까지** 분석할 데이터를 기반으로, 그 데이터에서 사용되는 **단어를 추출**하는 방법을 이야기했습니다
- 이번에는 {'파스타', '좋아'} 와 같은 단어에서 '파스타'라는 **명사만 선택**하는 방법을 알아보니다

A는 명사일까?

어제 A라는 가게에 가봤어

A에서 보자

A로 와줘

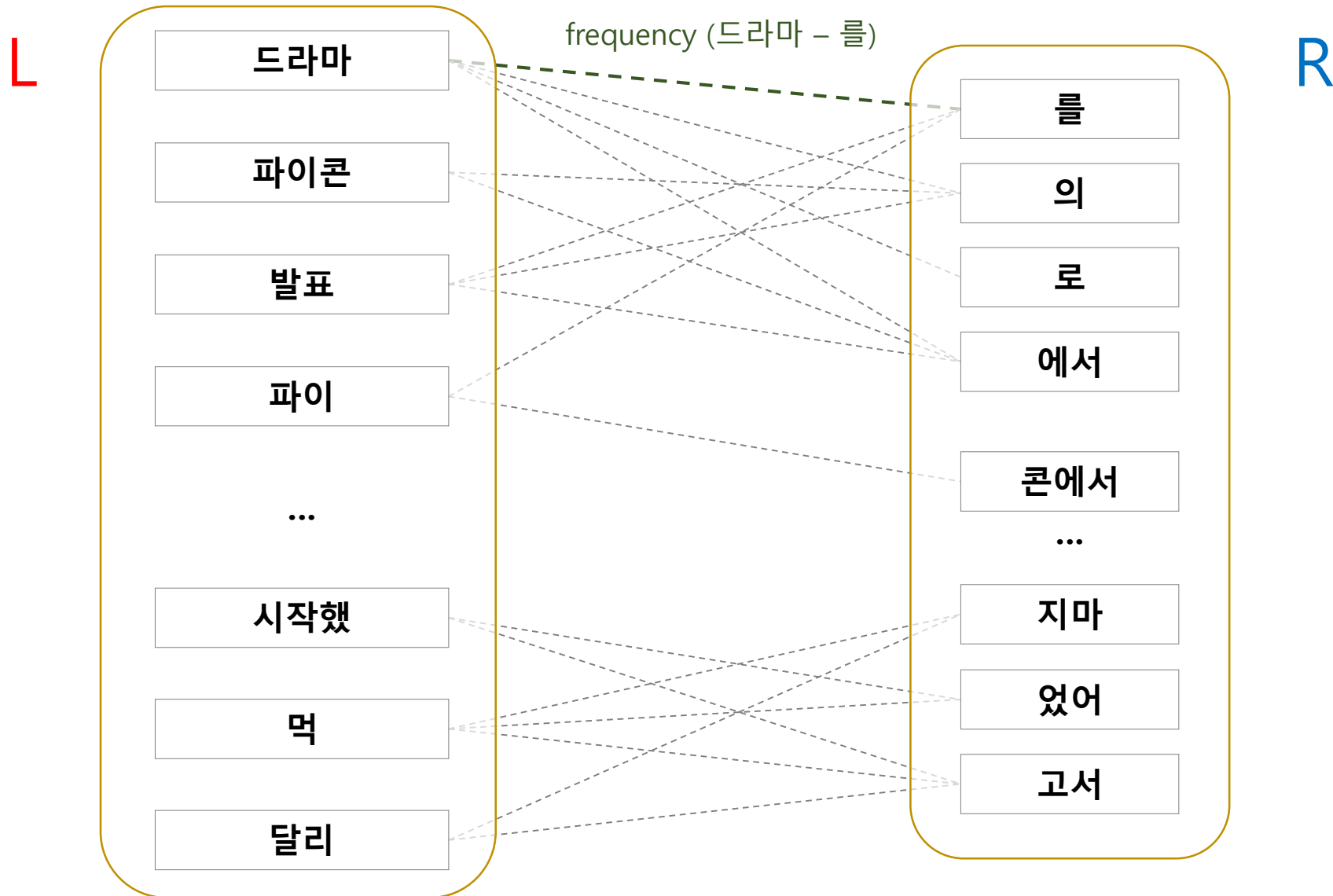
명사 우측에 등장하는 글자 분포를 이용하여

A가 명사임을 유추할 수 있다

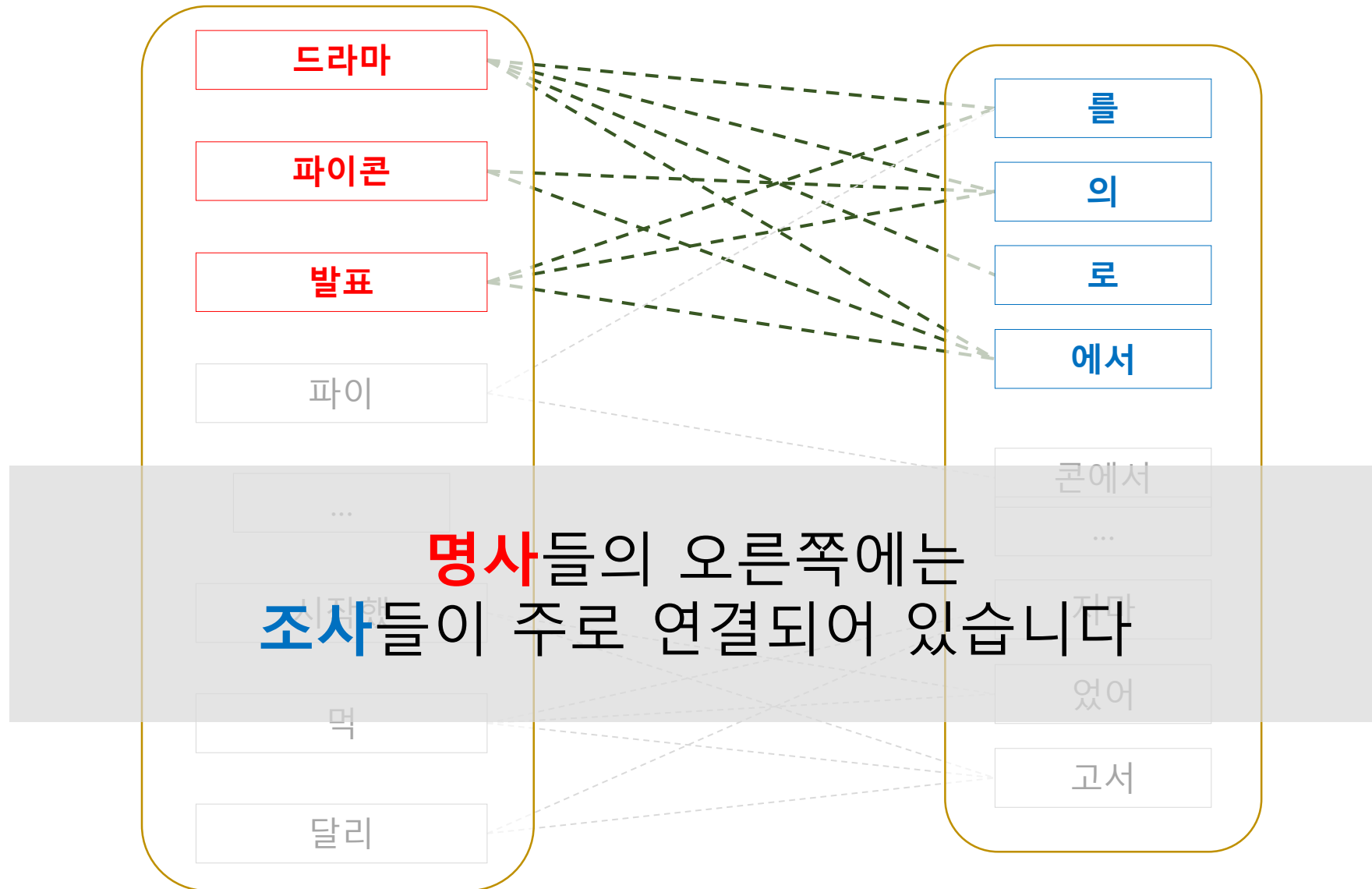
L – R graph

- 어절은 L + [R] 구조로 나눌 수 있습니다
 - 발표 + 를
 - 하 + 면서
- 데이터에서 모든 어절들을 두 개의 subwords로 나눈 뒤 연결하면 L – R graph를 만들 수 있습니다

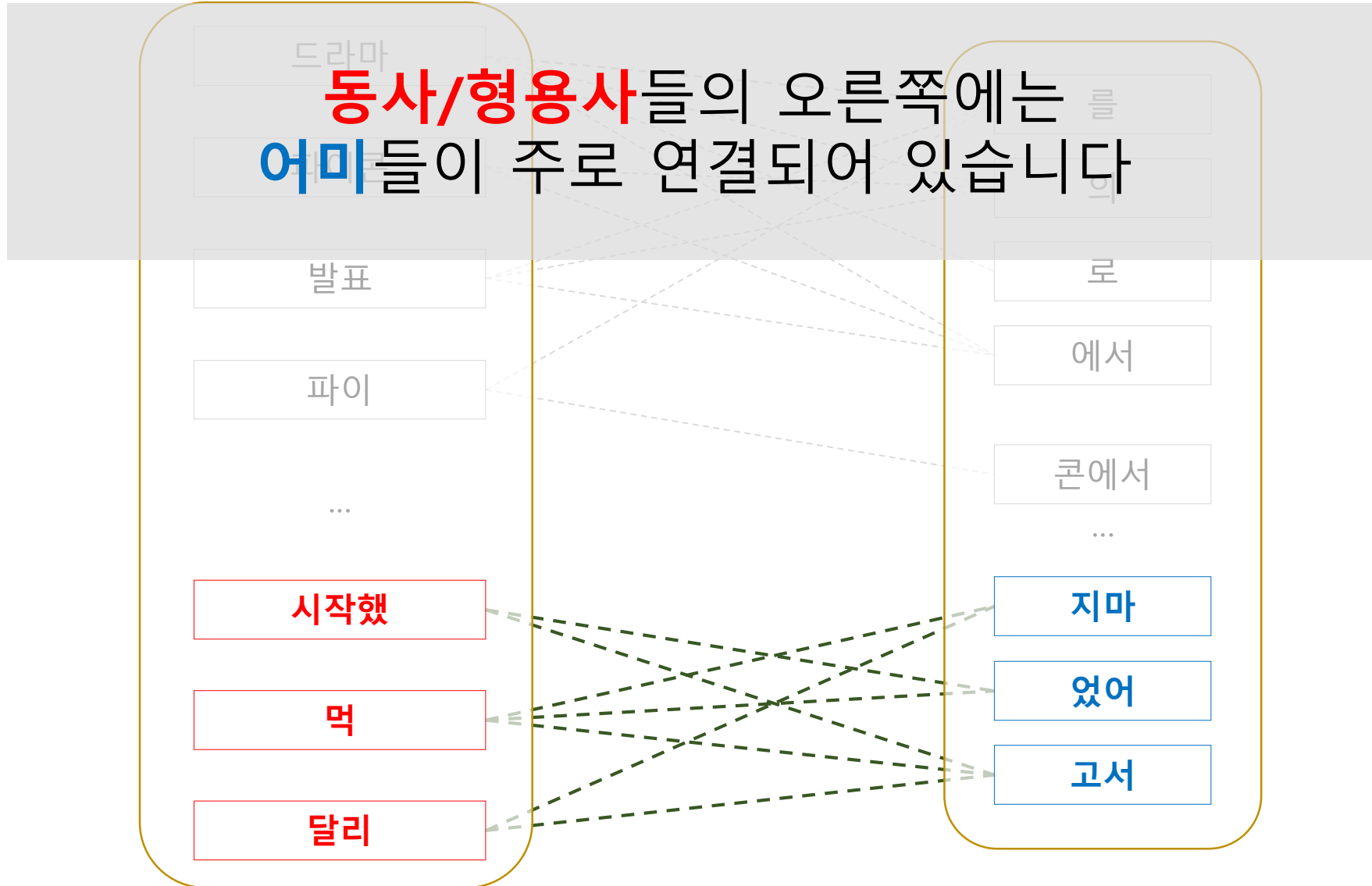
L - R graph



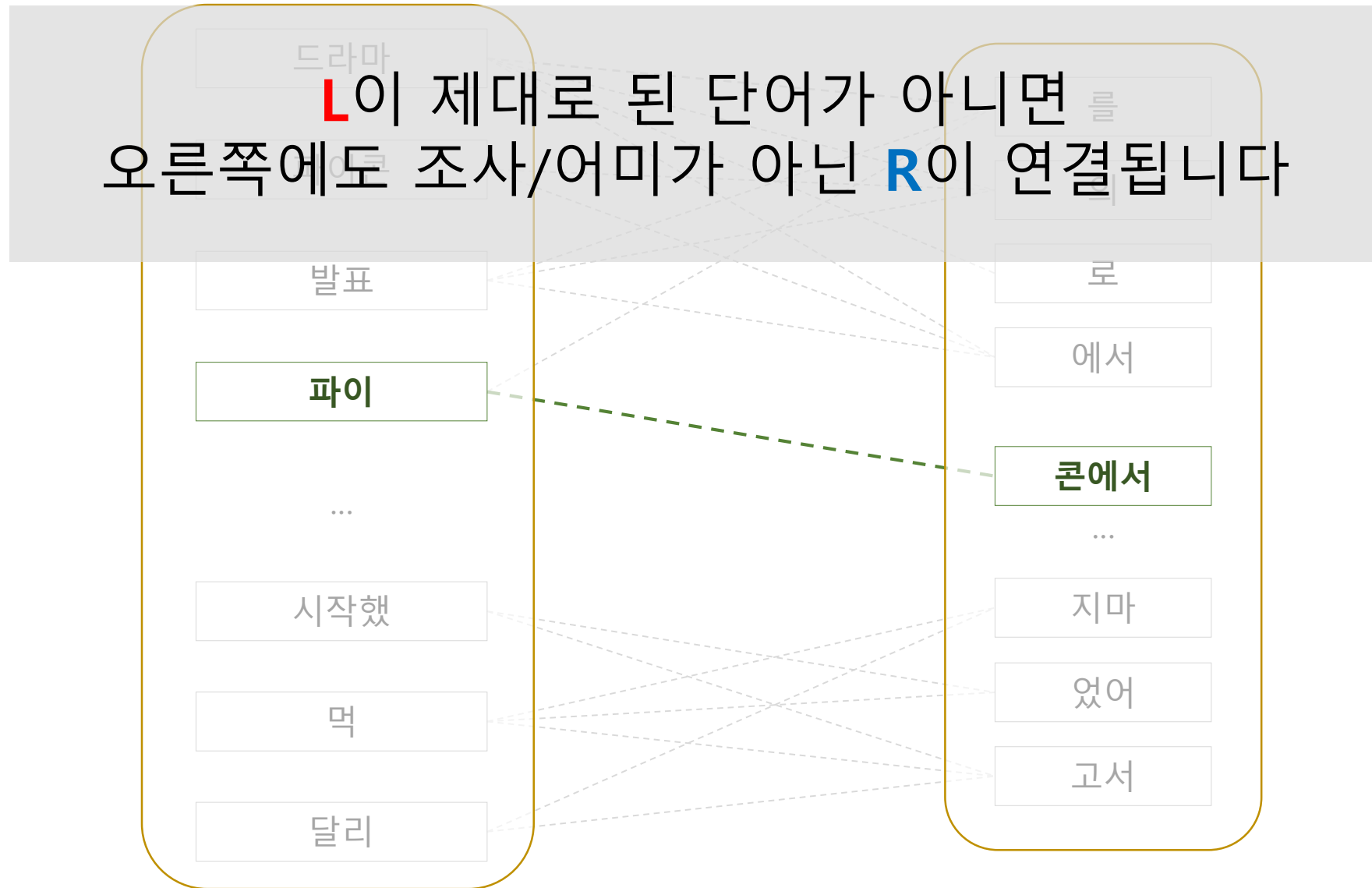
L – R graph



L – R graph



L - R graph



Noun Extraction

- L – R graph를 만들면 명사가 눈에 보입니다

```
lgraph = defaultdict(lambda: defaultdict(lambda: 0))

for doc in docs:
    for w in doc.split():
        n = len(w)
        for e in range(1, n+1):
            lgraph[ w[:e] ][ w[e: ] ] += 1

def get_r(L):
    return sorted( lgraph[L].items(), key=lambda x:-x[1])

get_r('드라마')
```

```
[(' ', 1268),
 ('를', 164),
 ('다', 152),
 ('의', 140),
 ('로', 138),
 ('에서', 98),
 ('와', 62),
 ('는', 55),
 ('에', 55),
 ('가', 48),
 ('이다', 24),
 ('인', 14),
 ... ]
```

Noun Extraction

- L – R graph: **[명사 + 조사]**, [동사 + 어미], [틀린 단어 + 틀린 단어]

get_r('드라마')

```
[(' ', 1268),  
 ('를', 164),  
 ('다', 152),  
 ('의', 140),  
 ('로', 138),  
 ('에서', 98),  
 ('와', 62),  
 ('는', 55),  
 ('에', 55),  
 ('가', 48),  
 ('이다', 24),  
 ('인', 14),  
 ... ]
```

get_r('시작했')

```
[('다', 567),  
 ('고', 73),  
 ('다고', 61),  
 ('습니다', 42),  
 ('는데', 26),  
 ('으며', 16),  
 ('지만', 15),  
 ('던', 12),  
 ('어요', 10),  
 ('다는', 7),  
 ('으나', 5),  
 ('죠', 4),  
 ... ]
```

get_r('드라')

```
[('마', 1268),  
 ('마를', 164),  
 ('마다', 152),  
 ('마의', 140),  
 ('마로', 138),  
 ('마에서', 98),  
 ('기', 65),  
 ('마와', 62),  
 ('마는', 55),  
 ('마에', 55),  
 ('마가', 48),  
 ('이브', 28),  
 ... ]
```


Noun Extraction

- L – R graph: [명사 + 조사], **[동사 + 어미]**, [틀린 단어 + 틀린 단어]

get_r(' 드라마')

```
[(' ', 1268),  
 ('를', 164),  
 ('다', 152),  
 ('의', 140),  
 ('로', 138),  
 ('에서', 98),  
 ('와', 62),  
 ('는', 55),  
 ('에', 55),  
 ('가', 48),  
 ('이다', 24),  
 ('인', 14),  
 ... ]
```

get_r(' **시작했**')

```
[('다', 567),  
 ('고', 73),  
 ('다고', 61),  
 ('습니다', 42),  
 ('는데', 26),  
 ('으며', 16),  
 ('지만', 15),  
 ('던', 12),  
 ('어요', 10),  
 ('다는', 7),  
 ('으나', 5),  
 ('죠', 4),  
 ... ]
```

get_r('드라')

```
[('마', 1268),  
 ('마를', 164),  
 ('마다', 152),  
 ('마의', 140),  
 ('마로', 138),  
 ('마에서', 98),  
 ('기', 65),  
 ('마와', 62),  
 ('마는', 55),  
 ('마에', 55),  
 ('마가', 48),  
 ('이브', 28),  
 ... ]
```

Noun Extraction

- L – R graph: [명사 + 조사], [동사 + 어미], **[틀린 단어 + 틀린 단어]**

get_r(' 드라마')

```
[(' ', 1268),  
 ('를', 164),  
 ('다', 152),  
 ('의', 140),  
 ('로', 138),  
 ('에서', 98),  
 ('와', 62),  
 ('는', 55),  
 ('에', 55),  
 ('가', 48),  
 ('이다', 24),  
 ('인', 14),  
 ... ]
```

get_r('시작했')

```
[('다', 567),  
 ('고', 73),  
 ('다고', 61),  
 ('습니다', 42),  
 ('는데', 26),  
 ('으며', 16),  
 ('지만', 15),  
 ('던', 12),  
 ('어요', 10),  
 ('다는', 7),  
 ('으나', 5),  
 ('죠', 4),  
 ... ]
```

get_r(' **드라**')

```
[('마', 1268),  
 ('마를', 164),  
 ('마다', 152),  
 ('마의', 140),  
 ('마로', 138),  
 ('마에서', 98),  
 ('기', 65),  
 ('마와', 62),  
 ('마는', 55),  
 ('마에', 55),  
 ('마가', 48),  
 ('이브', 28),  
 ... ]
```

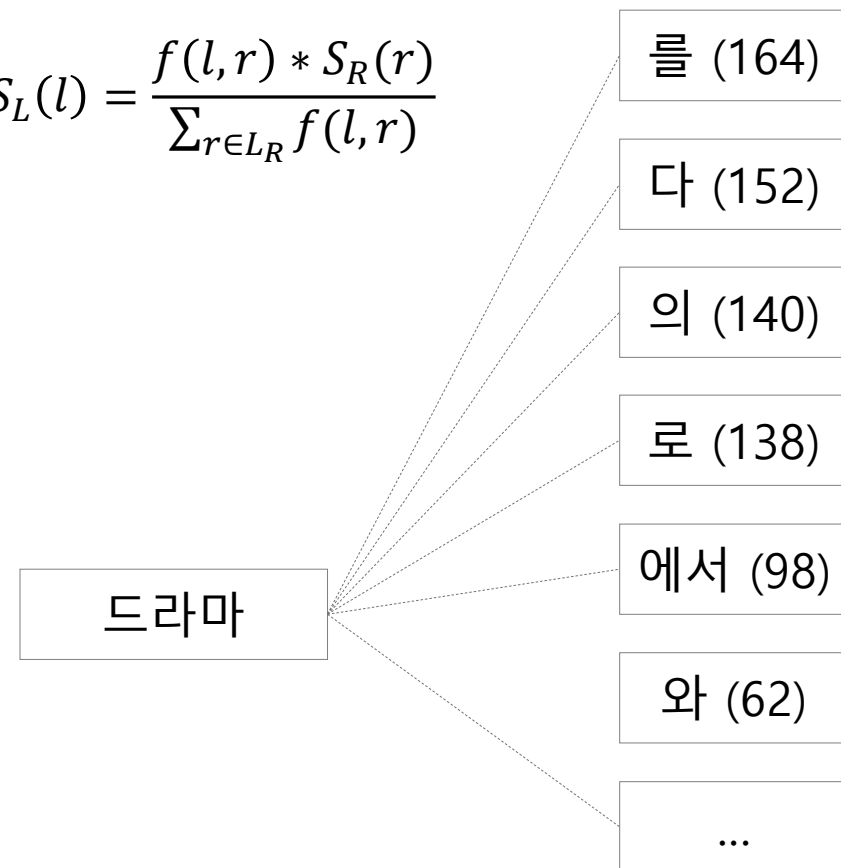
Noun Extraction

- R 의 분포를 이용하여 L의 명사 점수를 계산할 수 있습니다

```
r_scores = {'은': 0.5, '있다' : -0.9, ... }
```

```
def noun_score(L):  
    (norm, score, _total) = (0, 0, 0)  
  
    for R, frequency in get_r(L):  
        _total += frequency  
        if not R in r_scores:  
            continue  
        norm += frequency  
        score += frequency * r_scores[R]  
  
    score = score / norm if norm else 0  
    prop = norm / _total if _total else 0  
    return score, prop
```

$$S_L(l) = \frac{f(l, r) * S_R(r)}{\sum_{r \in L_R} f(l, r)}$$



Noun Extraction

- R 의 분포를 이용하여 L의 명사 점수를 계산할 수 있습니다

(명사 점수, 알려진 R 비율) = noun_score('Word')

```
noun_score('드라마')
```

```
(0.574, 0.921)
```

```
noun_score('시작했')
```

```
(-0.976, 0.999)
```

```
noun_score('드라')
```

```
(-0.661, 0.579)
```

틀린 단어 뒤에는 조사/어미 등이 아닌
글자들이 등장

Noun Extraction

- R 점수는 세종 말뭉치를 이용하여 계산합니다
 - 세종 말뭉치의 품사가 태깅된 정보로부터 L - R 구조의 테이블을 만듭니다

...
 예술가의 예술가/NNG+의/JKG 113
 예술가는 예술가/NNG+는/JX 45
 예술가가 예술가/NNG+가/JKS 43
 예술가들의 예술가/NNG+들/XSN+의/JKG 30
 ...



| 단어 품사 | 단어 / R | - 는 | - 의 | - 고 | - 었다 | - 었던 |
|-------|--------|-----|-----|-----|------|------|
| 명사 | 예술가 | 45 | 113 | 2 | 0 | 0 |
| 동사 | 먹 | 33 | 0 | 27 | 0 | 27 |
| ... | ... | ... | ... | ... | ... | ... |

Noun Extraction

- 명사의 빈도를 고려하여 **- R 앞에 명사가 등장할 점수**를 계산합니다

| 품사 | 단어 / R | - 는 | - 의 | - 고 | - 었다 | -구나 |
|-----|--------|-----|-----|-----|------|-----|
| 명사 | 예술가 | 45 | 113 | 2 | 0 | 0 |
| 명사 | 나이 | 54 | 87 | 27 | 0 | 27 |
| 동사 | 들 | 0 | 0 | 0 | 255 | 0 |
| 형용사 | 춡 | 0 | 0 | 0 | 0 | 87 |
| ... | ... | ... | ... | ... | ... | ... |



| 품사 | 빈도수 (비율) |
|-----|-----------------|
| 명사 | 704,197 (0.838) |
| ^명사 | 136,598 (0.162) |

| 품사 | 빈도수 | 보정빈도수 | 점수 |
|-----|-----|------------------------------|-----------------------------------|
| 명사 | 980 | 980 | 0.810 |
| ^명사 | 20 | $103.5 = 20 * (0.838/0.162)$ | $= (980 - 103.5) / (980 + 103.5)$ |

soynlp

- 후처리과정이 추가된 명사 추출기를 구현하였습니다

```
from soynlp.noun import LRNounExtractor
```

```
noun_extractor = LRNounExtractor(min_count=50)
```

```
nouns = noun_extractor.train_extract(corpus, minimum_noun_score=0.5)
```

```
nouns['설입']
```

```
NounScore(frequency=67, score=0.926, known_r_ratio=0.529)
```

```
nouns['드라마']
```

```
NounScore(frequency=4976, score=0.522, known_r_ratio=0.601)
```

KoNLPy + 사용자사전

https://github.com/lovit/customized_konlpy

사용자 사전 + 템플릿매칭을 이용한 토큰나이징/품사판별

- 명사/단어에 대해서 사용자 사전을 만든다면, 내가 임의의 템플릿을 만들어서 템플릿 매칭으로 아는 단어들을 처리할 수도 있습니다

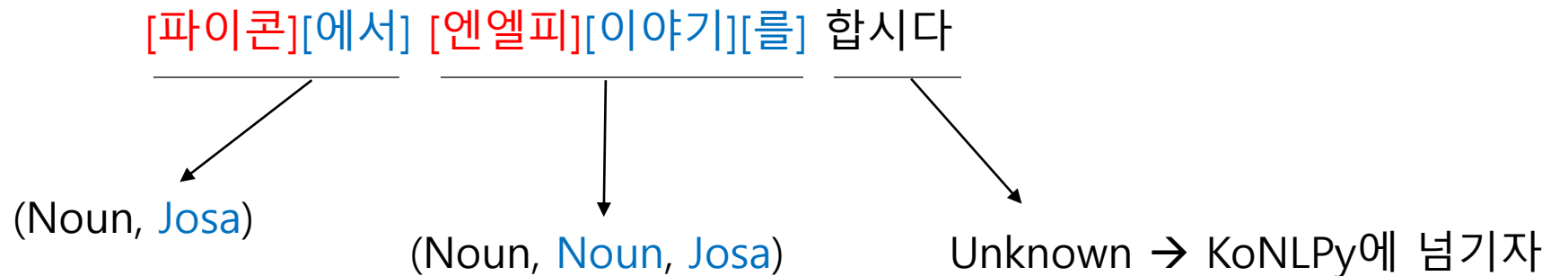
```
custom_dictionary = {'Noun': ['파이콘', '엔엘피', '아이오아이', '너무너무너무']}
```

```
templates = [('Noun', 'Noun'),  
              ('Noun', 'Josa'),  
              ('Noun', 'Noun', 'Josa')]
```

사용자 사전 + 템플릿매칭을 이용한 토큰나이징/품사판별

```
custom_dictionary = {'Noun': ['파이콘', '엔엘피', '아이오아이', '너무너무너무']}
```

```
templates = [('Noun', 'Noun'),  
             ('Noun', 'Josa'),  
             ('Noun', 'Noun', 'Josa')]
```



- * 빨간색은 사용자에게 의하여 추가된 사전에서 매칭
- * 파란색은 KoNLPy의 트위터 분석기에 포함되어 있는 사전에서 매칭

사용자 사전 + 템플릿매칭을 이용한 토큰나이징/품사판별

- customized_KoNLPy에 사전/템플릿 추가를 쉽게 할 수 있도록 KoNLPy를 래핑해두었습니다

```
from ckonlpy.tag import Twitter
```

```
tagger = Twitter()
```

```
tagger.add_dictionary(['파이콘', '엔엘피', '아이오아이', '너무너무너무'], 'Noun')
```

```
tagger.pos('파이콘에서 엔엘피이야기를 합시다')
```

```
tagger.pos('너무너무너무는 아이오아이의 노래예요')
```

```
[('파이콘', 'Noun'), ('에서', 'Josa'), ('엔엘피', 'Noun'), ('이야기', 'Noun'), ('를', 'Josa'), ('합', 'Verb'), ('시', 'PreEomi'), ('다', 'Eomi')]
```

```
[('너무너무너무', 'Noun'), ('는', 'Josa'), ('아이오아이', 'Noun'), ('의', 'Josa'), ('노래', 'Noun'), ('예요', 'Josa')]
```

정리

- 데이터기반으로 스스로 **단어/명사를 추출**하고, 이를 그대로 이용하거나 KoNLPy와 결합하여 **텍스트를 처리**하는 방법에 대하여 이야기하였습니다
- 아직도 좀 더 좋은 단어/명사 추출기와 토크나이즈를 연구하고 있습니다
- 조금 더 편한 데이터분석을, 분석 준비가 아닌 분석에 더 많이 집중하는데 도움이 되었으면 좋겠습니다