

# Text Analysis–Korean 2

# 텍스트를 단어로 표현하는 방법 (tokenization)

- Subwords tokenization은 단어를 유한한 subwords units으로 표현
  - 번역, 임베딩에 기반한 document representation에 이용됩니다



‘복면가왕’ 꽃새우는 아이오아이 출신 가수 청하였다. 3일 오후 4시 50분 방송된 MBC ‘미스터리 음악쇼-복면가왕’에서는 복어아가씨와 꽃새우의 1 라운드 무대가 펼쳐졌다. 55대 44로 복어아가씨가 승리를 거뒀고, 꽃새우는 이효리의 ‘텐미닛’을 부르며 청하임을 밝혔다. 이날 청하는 아이오아이 당시 춤으로 인기가 있지 않았느냐는 질문에 “당시엔 노래보다 춤에 더 자신이 있었다. 연정과 세정이 메인보컬이라 보여줄 기회가 없었고, 이번에 보여줘서 좋다”라고 말했다.



‘복면 가왕’\_ 꽃 새우 는\_ 아이 오 아이\_ 출 신\_ 가수 청 하 였다.\_ 3 일\_ 오후  
\_ 4 시\_ 50 분\_ 방송 된\_ ...

< Word Piece Model 예시 >

# 텍스트를 단어로 표현하는 방법 (tokenization)

- 키워드 추출 / 토픽 모델링을 위해서는 **단어가 제대로 인식되어야 합니다**



‘복면가왕’ 꽃새우는 아이오아이 출신 가수 청하였다. 3일 오후 4시 50분 방송된 MBC ‘미스터리 음악쇼-복면가왕’에서는 복어아가씨와 꽃새우의 1 라운드 무대가 펼쳐졌다. 55대 44로 복어아가씨가 승리를 거뒀고, 꽃새우는 이효리의 ‘텐미닛’을 부르며 청하임을 밝혔다. 이날 청하는 아이오아이 당시 춤으로 인기가 있지 않았느냐는 질문에 “당시엔 노래보다 춤에 더 자신이 있었다. 연정과 세정이 메인보컬이라 보여줄 기회가 없었고, 이번에 보여줘서 좋다”라고 말했다.



(‘, 기호), (복면가왕, 명사), (’, 기호), (꽃새우, 명사), (는, 조사), (아이오아이, 명사), (출신, 명사), (가수, 명사), (청하, 명사), (였다, 동사), (., 기호), ...

< 품사 판별에 의한 토크나이징 예시 >

# 품사 판별과 형태소 분석

- 한국어 단어의 품사는 5언 9품사로 구성되어 있습니다.

SENT: 재공연을 했어요

POS: (재공연, 명사), (을, 조사), (했어요, 동사)

한국어 품사				
불변어	체언	명사	대명사	수사
	수식언	관형사		부사
	관계언	조사		
	독립언	감탄사		
	가변어	용언	동사	형용사

# 품사 판별과 형태소 분석

- 품사 판별은 텍스트 데이터 분석을 위한 전처리 과정 중 하나입니다

```
from konlpy.tag import Kkma  
  
kkma = Kkma()  
  
kkma.pos('오류보고는 실행환경, 에러메세지와함께 설명을 최대한상세히!^^')
```

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

# 품사 판별과 형태소 분석

- 품사 판별을 위하여 형태소 분석이 이용될 수 있습니다

SENT: 재공연을 했어요

POS: (재공연, 명사), (을, 조사), (했어요, 동사)

MORPHEMES: (재, 관형사), (공연, 명사), (을, 조사), (하, 동사), (었, 선어말어미), (어요, 종결어미)

- 형태소 분석은 단어의 구성 요소들을 분해하여 인식하는 과정입니다

## 품사 판별과 형태소 분석

- 품사 사전이 잘 구축된다면, **사전기반으로도 품사판별**을 할 수 있습니다

SENT: 재공연을 했어요

POS: (재공연, 명사), (을, 조사), (했어요, 동사)

명사사전: { ... 재공연, ... }

동사사전: { ... 했어요, 했엉, 해써용, ... }

- 품사 판별이 목적이라면 형태소분석 과정이 필수는 아닙니다

# 미등록단어 문제

- 사전 기반으로 작동하는 형태소/품사 분석은 **사전 구성이 핵심입니다**

이 예제에서는 사전을 수정해 보겠습니다. 사전 파일들은

<src/main/resources/com/twitter/penguin/korean/util/>에 있습니다.

[src/main/resources/com/twitter/penguin/korean/util/noun/wikipedia\\_title\\_nouns.txt](src/main/resources/com/twitter/penguin/korean/util/noun/wikipedia_title_nouns.txt)에 동사가 들어가 있네요. 삭제했습니다. (이런 경우가 많이 있습니다. **수작업으로 없애 주어야 하는데요 여러분의 도움을 구합니다.** 아울러 복합명사도 최대한 분리되어야 합니다. 하동청룡리석불좌상 -> 하동 청룡리 석불 좌상)



< 트위터 한국어 분석기의 contribution guide snapshot >

## 미등록단어 문제

---

- 좋은 품질을 위하여 **사용자 사전**을 추가하여 사용합니다
  - 품사판별이 제대로 이뤄지지 않은 단어를 사전에 추가

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

## soynlp

---

- soynlp 는 다음 그림처럼, 통계 기반으로 단어와 품사를 추정하는 기능이 포함되어 있습니다.

# soynlp 는 통계 기반 단어/품사 추정 기능이 포함되어 있습니다

단어 추출을 통한  
**토크나이징**

**문장**: 아이오아이는이번공연에서좋은것모습을보였습니다이빠이빠

품사 추정을 통한  
**품사 사전 업데이트**

**단어열**: [아이오아이, 는, 이번, 공연, 에서,  
좋은, 것, 모습, 을, 보였습니다, 이빠, 이빠]

품사 사전을 이용한  
**품사 판별**

**명사 사전** += [아이오아이, ... ]  
**동사 사전** += [잘했어용, ... ]

**후처리**

**품사열**: [(아이오아이, 명사), (는, 조사), (이번, 명사), (공연, 명사),  
(에서, 조사), (좋은, 형용사), (것, 명사), (모습, 명사), (을, 조사),  
(보였습니다, 동사), (이빠, 형용사), (이빠, 형용사)]

# 사전 기반으로 작동하는 품사 판별기를 만들 수 있습니다

문장: 아이오아이는이번공연에서좋은것모습을보였습니다이빠이빠

단어 추출을 통한  
**토크나이징**

단어열: [아이오아이, 는, 이번, 공연, 에서,  
좋은, 것, 모습, 을, 보였습니다, 이빠, 이빠]

품사 추정을 통한  
**품사 사전 업데이트**

명사 사전 += [아이오아이, ... ]  
동사 사전 += [잘했어용, ... ]

품사 사전을 이용한  
**품사 판별**

**품사열**: [(아이오아이, 명사), (는, 조사), (이번, 명사), (공연, 명사),  
(에서, 조사), (좋은, 형용사), (것, 명사), (모습, 명사), (을, 조사),  
(보였습니다, 동사), (이빠, 형용사), (이빠, 형용사)]

**후처리**

# 품사 판별

---

- 사전 기반 품사 판별은 세 가지 과정으로 구성되어 있습니다.

- **1 단계: 후보 생성**

- 사전을 이용하여 문장에서 가능한 품사열 후보를 만듭니다
    - 가능성이 적은 후보들을 제거한다면 계산 속도가 빨라집니다

- **2 단계: 후보 평가**

- 후보들 중에서 가장 적절한 품사열을 선택합니다

- **3 단계: 후처리**

- 사전에 포함되지 않는 단어들 처리 및 그 외의 후처리를 수행합니다

# 품사 판별

- Finite State Model 처럼 순차적으로 후보를 만들 수도 있습니다.

읽은 input  
“아이오”아이는이번공연에서좋은것모습을보였습니다이빠이빠”

후보 1: “아/명사 + 이/조사 + 오/명사” : score -0.53  
후보 2: “아이/명사 + 오/명사” : score -0.27  
후보 3: “아이오/명사” : score -0.11

k - beam

## 품사 판별

---

- 하지만 Max Score Tokenizer 와 같이, 알고 있는 단어부터 품사 판별을 수행하도록 하였습니다
  - 긴 문장이 주어지면 사람은 아는 단어부터 눈에 보입니다
  - 확신이 있는 단어부터 품사를 판별합니다

# 품사 판별

**Step 1:** 어절의 “**명사/형용사/동사/부사**”를 사전과 매칭 합니다

- 단어가 겹치더라도 가능한 모든 후보를 만듭니다

```
sent = '아이오아이는이번공연에서좋은강모습을보였습니다이빠이빠'
```

```
candidates = _initialize_L(sent)
```

```
[['아이', 'Noun', 0, 2],
```

```
['아이오', 'Noun', 0, 3],
```

```
['아이오아이', 'Noun', 0, 5],
```

```
['이오', 'Noun', 1, 3],
```

```
['아이', 'Noun', 3, 5],
```

```
['이는', 'Verb', 4, 6],
```

```
['이번', 'Noun', 6, 8],
```

```
['공연', 'Noun', 8, 10],
```

```
... ]
```

[ 단어, 품사, 시작 index, 종료 index ]

# 품사 판별

Step 2: 포함 관계에 있는 같은 품사의 단어중, 가장 긴 것만 남깁니다

```
sent = '아이오아이는이번공연에서좋은강모습을보였습니다이빠이빠'
```

```
candidates = _remove_l_subsets(candidates)
```

```
[['아이', 'Noun', 0, 2],
```

```
['아이오', 'Noun', 0, 3],
```

```
['아이오아이', 'Noun', 0, 5],
```

‘아이오아이’에 포함되는 모든 명사는 제거합니다

```
['아오', 'Noun', 1, 3],
```

```
['아이', 'Noun', 3, 5],
```

```
['이는', 'Verb', 4, 6],
```

“이는/Verb”는 ‘아이오아이’와 다른 품사이며, 포함되지 않습니다

```
['이번', 'Noun', 6, 8],
```

```
['공연', 'Noun', 8, 10],
```

```
... ]
```

# 품사 판별

## Step 3: “조사/형용사/동사”를 확장합니다

- 조사, 어미보다 명사/형용사/동사/부사를 잘 인식하는 것이 중요합니다

```
sent = '아이오아이는이번공연에서좋은강모습을보였습니다이빠이빠'
```

```
candidates = _initialize_LR(sent, candidates)
```

```
[ [('아이오아이', 'Noun'), (' ', ' '), 0, 5, 5],  
  [('아이오아이', 'Noun'), ('는', 'Josa'), 0, 5, 6],  
  [('이', 'Verb'), (' ', ' '), 4, 6, 6],  
  [('이번', 'Noun'), (' ', ' '), 6, 8, 8],  
  [('공연', 'Noun'), (' ', ' '), 8, 10, 10],  
  [('공연', 'Noun'), ('에', 'Josa'), 8, 10, 11],  
  [('공연', 'Noun'), ('에서', 'Josa'), 8, 10, 12],  
  ...  
 ]
```

# 품사 판별

## Step 4: 확장된 단어 중 같은 품사는 가장 긴 것만 남깁니다

```
sent = '아이오아이는이번공연에서좋은강모습을보였습니다이빠이빠'
```

```
candidates = _remove_r_subsets(candidates)
```

```
[ [('아이오아이', 'Noun'), (' ', ' '), 0, 5, 5],  
  [('아이오아이', 'Noun'), ('는', 'Josa'), 0, 5, 6],  
  [('이|는', 'Verb'), (' ', ' '), 4, 6, 6],  
  [('이번', 'Noun'), (' ', ' '), 6, 8, 8],  
  [('공연', 'Noun'), (' ', ' '), 8, 10, 10],  
  [(공연', 'Noun'), ('에', 'Josa'), 8, 10, 11],  
  [(공연', 'Noun'), ('에서', 'Josa'), 8, 10, 12],  
  ...  
 ]
```

# 품사 판별

## Step 4: 최종 후보입니다

```
sent = '아이오아이는이번공연에서좋은cantidad을보였습니다이빠이빠'
```

```
candidates = _initialize(sent)
```

```
[['아이오아이', 'Noun'), ('는', 'Josa'), 0, 6, 6],  
[('아이오아이', 'Noun'), ('', ''), 0, 5, 5],  
[('이는', 'Verb'), ('', ''), 4, 6, 2],  
[('이번', 'Noun'), ('', ''), 6, 8, 2],  
[('공연', 'Noun'), ('에서', 'Josa'), 8, 12, 4],  
[('공연', 'Noun'), ('', ''), 8, 10, 2],  
[('좋은', 'Adjective'), ('', ''), 12, 14, 2],  
[('모습', 'Noun'), ('을', 'Josa'), 15, 18, 3],  
[('모습', 'Noun'), ('', ''), 15, 17, 2],  
[('보였습니다', 'Verb'), ('', ''), 18, 23, 5],  
[('다이', 'Noun'), ('', ''), 22, 24, 2],  
... ]
```

# 품사 판별

## Step 5: “L + R”을 고려하여 scoring을 합니다

```
sent = '아이오아이는이번공연에서좋은.yang모습을보였습니다이빠이빠'
```

```
scores = _scoring(candidates)
```

[['아이오아이', 'Noun'], ('는', 'Josa'), 0, 6, 6,	<b>3.39]</b> ,	←	'아이오아이/명사 + 는/조사' 점수
[('아이오아이', 'Noun'), ('', ''), 0, 5, 5,	2.54],		
[('이는', 'Verb'), ('', ''), 4, 6, 2,	1.90],		
[('이번', 'Noun'), ('', ''), 6, 8, 2,	2.24],		
[('공연', 'Noun'), ('에서', 'Josa'), 8, 12, 4,	2.77],		
[('공연', 'Noun'), ('', ''), 8, 10, 2,	1.73],		
[('좋은', 'Adjective'), ('', ''), 12, 14, 2,	2.25],		
[('모습', 'Noun'), ('을', 'Josa'), 15, 18, 3,	3.26],		
[('모습', 'Noun'), ('', ''), 15, 17, 2,	2.01],		
[('보였습니다', 'Verb'), ('', ''), 18, 23, 5,	2.21],		
[('다이', 'Noun'), ('', ''), 22, 24, 2,	1.11],		
... ]			

# 품사 판별

Step 5: 점수 계산 feature를 만든 뒤, weight를 곱하여 scoring을 합니다

```
profile = OrderedDict([
    ('cohesion_l', 0.5),
    ('droprate_l', 0.5),
    ('log_count_l', 0.1),

    ('prob_l2r', 0.1),
    ('log_count_l2r', 0.1),
    ('known_LR', 1.0),

    ('R_is_syllable', -0.1),
    ('log_length', 0.5)
])
```

L의 cohesion score	* 0.5
+ L의 droprate score	* 0.5
+ L의 log 빈도수	* 0.1
+ 분석텍스트의 $P(L \rightarrow R)$	* 0.1
+ 분석텍스트의 $Freq(L \rightarrow R)$	* 0.1
+ L과 R이 모두 알려진 품사	* 1.0
+ 1음절 조사/어미	* -0.1
+ “L+R” 길이의 log	* 0.5

# 품사 판별

Step 5: 점수 계산 feature를 만든 뒤, weight를 곱하여 scoring을 합니다

```
profile = OrderedDict([
```

```
    ('cohesion_1', 0.5),
```

```
    ('droprate_1', 0.5),
```

```
    ('log_count_1', 0.1),
```

분석하려는 텍스트 도메인의  
특성을 반영하기 위한 장치

```
    ('log_count_12r', 0.1),
```

```
    ('known_LR', 1.0),
```

```
    ('R_is_syllable', -0.1),
```

```
    ('log_length', 0.5)
```

```
])
```

L의 cohesion score \* 0.5

+ L의 droprate score \* 0.5

+ L의 log 빈도수 \* 0.1

+ 분석텍스트의  $P(L \rightarrow R)$  \* 0.1

+ 분석텍스트의  $Freq(L \rightarrow R)$  \* 0.1

+ L과 R이 모두 알려진 품사 \* 1.0

+ 1음절 조사/어미 \* -0.1

+ “L+R” 길이의 log \* 0.5

# 품사 판별

## Step 6: 높은 점수의 단어부터 품사를 부여 / 겹치는 부분은 제거합니다

```
sent = '아이오아이는이번공연에서좋은쁨모습을보였습니다이빠이빠'
```

```
words = _find_best(scores)
```

```
[ [('아이', 'Noun'), ('는', 'Josa'), 0, 6, 6, 3.39],  
[('모습', 'Noun'), ('을', 'Josa'), 15, 18, 3, 3.26],  
[('공연', 'Noun'), ('에서', 'Josa'), 8, 12, 4, 2.77],  
[('아이', 'Noun'), ('이', 'Josa'), 0, 5, 5, 2.54],  
[('좋은', 'Adjective'), (' ', ' '), 12, 14, 2, 2.25],  
[('이번', 'Noun'), (' ', ' '), 6, 8, 2, 2.24],  
[('보였습니다', 'Verb'), (' ', ' '), 18, 23, 5, 2.21],  
[('모습', 'Noun'), (' ', ' '), 15, 17, 2, 2.01],  
[('아', 'Noun'), ('는', 'Verb'), (' ', ' '), 4, 6, 2, 1.90],  
[('공연', 'Noun'), (' ', ' '), 8, 10, 2, 1.73],  
[('다', 'Noun'), ('이', 'Verb'), (' ', ' '), 22, 24, 2, 1.11],  
... ]
```

# 품사 판별

## Step 7: 사전에 등록되지 않은 단어는 아직 인식되지 않았습니다

```
sent = '아이오아이는이번공연에서좋은#모습을보였습니다이빠이빠'
```

```
[('아이오아이', 'Noun'),  
 ('는', 'Josa'),  
 ('이번', 'Noun'),  
 ('공연', 'Noun'),  
 ('에서', 'Josa'),  
 ('좋은', 'Adjective'),  
 ('모습', 'Noun'),  
 ('을', 'Josa'),  
 ('보였습니다', 'Verb'),  
 ('이빠', 'Adjective'),  
 ('이빠', 'Adjective')]
```

# 품사 판별

## Step 7: 사전에 없는 단어로 구성된 sub-sentence를 **후처리** 합니다

```
sent = '아이오아이는이번공연에서좋은깡모습을보였습니다이빠이빠'
```

```
[('아이오아이', 'Noun'),  
 ('는', 'Josa'),  
 ('이번', 'Noun'),  
 ('공연', 'Noun'),  
 ('에서', 'Josa'),  
 ('좋은', 'Adjective'),  
 ('깡', None),  
 ('모습', 'Noun'),  
 ('을', 'Josa'),  
 ('보였습니다', 'Verb'),  
 ('이빠', 'Adjective'),  
 ('이빠', 'Adjective')]
```

# 품사 판별

## Step 7: 길이가 긴 부분은 Max Score Tokenizer로 토크나이징까지 합니다

```
sent = '아이오아이는이번공연에서좋은모습을보였습니다양순이들이죠아'
```

```
[('아이오아이', 'Noun'),  
 ('는', 'Josa'),  
 ('이번', 'Noun'),  
 ('공연', 'Noun'),  
 ('에서', 'Josa'),  
 ('좋은', 'Adjective'),  
 ('모습', 'Noun'),  
 ('을', 'Josa'),  
 ('보였습니다', 'Verb'),  
 ('양순이들', None),  
 ('이', None),  
 ('죠아', None)]
```

- 품사 사전에 ['양순이들', '죠아']가 등록되어 있지 않더라도 가능한 단어로 나눠듭니다
- "양순이들 + 이"에서 조사 사전을 바탕으로 "양순이들"의 품사를 추정하는 것은 현재 개발중입니다

# 품사 판별 성능: vs. 트위터 한국어 분석기

twitter.pos(sent)

Process time: 102 ms

```
[('아이오', 'Noun'),  
 ('아이', 'Noun'),  
 ('는', 'Josa'),  
 ('이번', 'Noun'),  
 ('공연', 'Noun'),  
 ('에서', 'Josa'),  
 ('좋', 'Adjective'),  
 ('은', 'Eomi'),  
 ('캉', 'Noun'),  
 ('모습', 'Noun'),  
 ('을', 'Josa'),  
 ('보였', 'Verb'),  
 ('습니다', 'Eomi'),  
 ('이뻐', 'Adjective'),  
 ('이뻐', 'Adjective')]
```

proposed.pos(sent)

Process time: 3.05 ms

```
[('아이오아이', 'Noun'),  
 ('는', 'Josa'),  
 ('이번', 'Noun'),  
 ('공연', 'Noun'),  
 ('에서', 'Josa'),  
 ('좋은', 'Adjective'),  
 ('캉', None),  
 ('모습', 'Noun'),  
 ('을', 'Josa'),  
 ('보였습니다', 'Verb'),  
 ('이뻐', 'Adjective'),  
 ('이뻐', 'Adjective')]
```

- 
- 알고리즘은 예외가 발생하며, 사용자는 예외를 쉽게 수정하고 싶어합니다
    - 사전의 단어 추가 및 삭제
    - 반드시 보존하고 싶은 단어의 손쉬운 보호

# 사전의 단어 추가 및 삭제

---

- 컴파일을 다시 하지 않으면서 단어를 추가/삭제 해야 합니다

```
from soynlp.pos import LRMaxScoreTagger

my_dictionary_folders=['folder1', 'folder2']

tagger = LRMaxScoreTagger(my_dictionary_folders)

tagger.add_words_into_dictionary( ['아이오아이'], 'Noun')

tagger.remove_words_from_dictionary( ['아이오아이'], 'Noun')
```

# 반드시 보존하고 싶은 단어의 손쉬운 보호

- 도매인의 키워드, 혹은 중요한 단어들은 선호도를 설정합니다

```
tagger.set_word_preference(['아이오아이', '너무너무너무'], 'Noun', 10)
```

```
[(('아이오아이', 'Noun'), ('는', 'Josa'), 0, 6, 6,  
 3.39],  
 [('아이오아이', 'Noun'), ('', ''), 0, 5, 5,  
 2.54],  
 [('이는', 'Verb'), ('', ''), 4, 6, 2, 1.90],  
 ... ]
```

```
[(('아이오아이', 'Noun'), ('는', 'Josa'), 0, 6, 6,  
 13.39],  
 [('아이오아이', 'Noun'), ('', ''), 0, 5, 5,  
 12.54],  
 [('이는', 'Verb'), ('', ''), 4, 6, 2, 1.90],  
 ... ]
```



다른 단어로 인식될 가능성을  
원천 봉쇄

# unigram

---

```
tokenize('라라랜드는 재미있는 영화입니다')
```

```
라라랜드, 는, 재미, 있는, 영화, 입니다
```

- 독립된 하나의 단어를 unigram이라 합니다.
- 위 문장은 6개의 단어로 이뤄져 있습니다.

# bigram

---

```
tokenize('라라랜드는 재미있는 영화입니다')
```

```
라라랜드, 는, 재미, 있는, 영화, 입니다
```

- bigram은 두 개의 단어 조합을 하나의 단어로 취급합니다
  - 연결 부분을 표현하기 위하여 '-'을 이용합니다.
    - "재미 - 있는"
    - 두 단어는 반드시 연속될 필요는 없습니다
      - "재미 - 영화 "

# bigram

---

```
tokenize('라라랜드는 재미있는 영화입니다')
```

라라랜드, 는, [재미, 있는], 영화, 입니다

- bigram은 문맥의 표현력이 좋습니다
  - '재미'라는 단어 만으로는 이 문장의 긍/부정을 알기 어렵습니다
  - '있는' 만으로는 어떤 의미인지 알기 어렵습니다
  - '재미 – 있는'은 긍정적인 문맥을 표현합니다.

# bigram

---

- document classification은 bigram + linear model 이면 충분합니다
  - 많은 연구들에서도 sentiment/category classification에서는 bigram features 이면 logistic regression과 같은 모델이어도 분류가 잘된다 알려졌습니다 [1,2]
  - 하지만 unigram 보다 bigram을 이용하는 것은 큰 도움이 됩니다

# n-gram

---

```
tokenize('라라랜드는 재미있는 영화입니다')
```

라라랜드, 는, [재미, 있는, 영화], 입니다

- n-gram은 세 개 이상의 단어 조합을 하나의 단어로 취급합니다
  - Phrase 혹은 “바람 – 의 – 나라”와 같은 단어가 되기도 합니다

## n-gram

---

- n-gram의 추출 방법은 다양합니다
  - 가장 좋은 것은 없습니다
  - 하지만, 계산 과정에서 많은 메모리가 필요할 수 있습니다

# n-gram

---

- By counting
  - 가장 간단한 방법은 모든 n-gram에 대하여 빈도수를 계산하는 것입니다
  - 'Josa + Verb + Noun'과 같은 형태의 ngram이 추출될 수도 있습니다

[('열린/Verb', '영화/Noun'), 476),  
 ('에서/Noun', '열린/Verb', '영화/Noun'), 288),  
 ('코미디/Noun', '영화/Noun'), 204),  
 ('국제/Noun', '영화제/Noun'), 200),  
 ('에서/Josa', '열린/Verb', '영화/Noun'), 185)]

< 마지막 단어가 영화인 bi/trigram >

[('재/Noun', '배포/Noun', '금지/Noun'), 20436),  
 ('및/Noun', '재/Noun', '배포/Noun'), 14687),  
 ('전재/Noun', '및/Noun', '재/Noun'), 14340),  
 ('무단/Noun', '전재/Noun', '및/Noun'), 14340),  
 ('무단/Noun', '전재/Noun', '재/Noun'), 5178)]

< 뉴스기사의 빈도수 기준 상위 5개의 trigram >

# n-gram

---

- By Point Mutual Information (PMI) – like
  - Mikolov는 PMI를 조금 바꾼 간단한 bigram 점수를 만들었습니다<sup>[1]</sup>

$$score(w_i, w_j) = \frac{count(w_i, w_j) - \delta}{count(w_i) \times count(w_j)}$$

('허심/Noun', '탄회/Noun'), ('무라카미/Noun', '하루키/Noun'), ('로웰/Noun', '패독/Noun')

...

('가습기/Noun', '살균제/Noun'), ('자유로이/Adverb', '접근할/Verb'), ('새판/Noun', '짜기/Verb')

# n-gram

---

- Extending Point Mutual Information (PMI)
  - PMI는 2개의 items에 대하여 정의되어 있습니다
  - n 개의 items에 대한 확장방법은 다양하며, 절대적인 정답은 없습니다
  - bigram이 더 정확한 문맥을 나타내므로, 이를 이용하여 PMI를 확장합니다

$$score(w_i, w_j, w_k) = \frac{count(w_i, w_j, w_k) - \delta}{count(w_i, w_j) \times count(w_j, w_k)}$$

# Korean n-gram

---

```
tokenize('라라랜드는 재미있는 영화입니다')
```

```
라라랜드, 는, 재미, 있는, 영화, 입니다
```

- 한국어에서 의미있는 n-gram은 품사 정보를 이용하는 것이 좋습니다
  - “있는 – 영화”는 유의미한 n-gram이 아닙니다
  - 첫 단어가 명사이거나, 조사/어미가 아닌 n-gram을 선택할 수 있습니다
    - “재미 – 있는”

# Korean n-gram

---

```
tokenize('라라랜드는 재미있는 영화입니다')
```

```
라라랜드/명사, 는/조사, 재미/명사, 있는/형용사, 영화/명사, 입니다/형용사
```

- 의미있는 n-gram을 추출하기 위하여 templates을 이용해도 좋습니다
  - (명사, 조사) / (명사, 명사) 처럼 미리 정의한 templates에 품사가 매칭되는 n-grams 을 추출할 수 있습니다

# Korean n-gram

---

```
tokenize('라라랜드가 개봉 했습니다')
```

```
라라랜드/명사, 가/조사, 개봉/명사, 했습니다/동사
```

- 조사/어미를 skip 하는 templates도 유용합니다
  - (명사, [조사], 명사)를 이용하면

# Korean n-gram

---

- 조사/어미를 skip 하는 templates도 유용합니다
  - (명사, [조사], 명사)를 이용하면 아래 두 문장에서 모두 “라라랜드 – 개봉” 을 추출할 수 있습니다

라라랜드/명사, 가/조사, 개봉/명사, 했습니다/동사

진짜/부사, ?/기호, 라라랜드/명사, 개봉/명사, 했어/동사, ?/기호

# (Positive) Point Mutual Information

PMI, PPMI

- 
- Word2Vec 같은 word embedding 방법이 등장하기 전부터 semantics 을 학습하기 위한 연구들이 제안되었습니다.
  - Point Mutual Information (PMI) 는 bow models 같은 sparse vector representation 에서 semantics 을 학습하기 위해 이용된 방법입니다.

# Point Mutual Information (PMI)

---

- 확률 이론에서는 두 확률이 서로 독립인지 판단하는 방법을 제공합니다.
  - 전체 공간에서의  $p(y)$  와  $x$  조건에서의  $p(y|x)$  가 같으면  $x, y$  는 독립입니다.

$$\frac{p(x,y)}{p(x) \times p(y)} = \frac{p(y|x)}{p(y)} = 1$$

# Point Mutual Information (PMI)

- 확률 이론에서는 두 확률이 서로 독립인지 판단하는 방법을 제공합니다.
  - 두 확률이 독립이면 다음 조건이 성립합니다.

$$\frac{p(x,y)}{p(x) \times p(y)} = 1 , \frac{p(\text{안경 } o, \text{저녁 } o)}{p(\text{안경 } o) \times p(\text{저녁 } o)} = \frac{\frac{1}{12}}{\frac{3}{12} \times \frac{4}{12}} = 1$$

	저녁을 먹었다	저녁을 먹지 않았다	Prob.
안경을 썼다	100	200	3 / 12
안경을 쓰지 않았다	300	600	9 / 12
Prob	4 / 12	8 / 12	

# Point Mutual Information (PMI)

---

- 서로 양의 상관성이 있으면  $\frac{p(x,y)}{p(x) \times p(y)}$  이 1보다 큽니다.

$$\frac{p(x,y)}{p(x) \times p(y)} = 1 , \frac{p(\text{안경 } o, \text{저녁 } o)}{p(\text{안경 } o) \times p(\text{저녁 } o)} = \frac{\frac{2}{12}}{\frac{5}{12} \times \frac{3}{12}} = 1.2$$

	저녁을 먹었다	저녁을 먹지 않았다	Prob.
안경을 썼다	200	100	3 / 12
안경을 쓰지 않았다	300	600	9 / 12
Prob	5 / 12	7 / 12	

# Point Mutual Information (PMI)

---

- PMI 는 두 경우의 상관성을 표현하는 index 입니다.

$$PMI(x, y) = \log \frac{p(x, y)}{p(x) \times p(y)}$$

- 양의 상관관계라면 0 보다 큰 값을 반대라면 0 보다 작은 값을 지닙니다.
- 값의 방향성에 해석력이 있습니다.

## Positive PMI (PPMI)

---

- 자연어처리에서의 semantic에서는 음의 상관관계에 큰 의미가 없습니다.
  - 양의 상관관계의 패턴을 강조하기 위해 0 보다 작은 값을 0 으로 변환합니다.

$$PPMI(x, y) = \max(0, PMI(x, y))$$

# Smoothing PMI

---

- PMI (PPMI) 는 infrequent 에 민감합니다.

$$PMI(x, y) = \log \frac{p(x, y)}{p(x) \times p(y)} = \log \frac{p(y | x)}{p(y)}$$

- $p(y)$  가 지나치게 작으면, 대부분의  $y$  가  $x$ 에서 발생할 가능성이 있습니다.

## Smoothing PMI

---

- 한 가지 해결책으로  $p(y)$  에 일정한 값  $\alpha$  를 더합니다.
  - $p(y|x)$  가  $\alpha$  이상인 경우에만  $PMI(x,y)$  를 계산하는 효과가 있습니다.
  - $\alpha$  는 threshold 역할을 합니다.

$$PMI(x,y) = \log \frac{p(x,y)}{p(x) \times (p(y) + \alpha)} = \log \frac{p(y|x)}{p(y) + \alpha}$$

# Smoothing PMI

---

- $\alpha$  역시  $x$ 에 따라 다르게 적용되어야 합니다.
  - $x$ 에 따라  $\log \frac{p(y|x)}{p(y)+\alpha}$ 의 경향은 달라집니다.
  - Universal parameter  $\alpha$ 는 없습니다.

$$PMI(x, y) = \log \frac{p(x, y)}{p(x) \times (p(y) + \alpha)} = \log \frac{p(y|x)}{p(y) + \alpha}$$

# Defining contexts

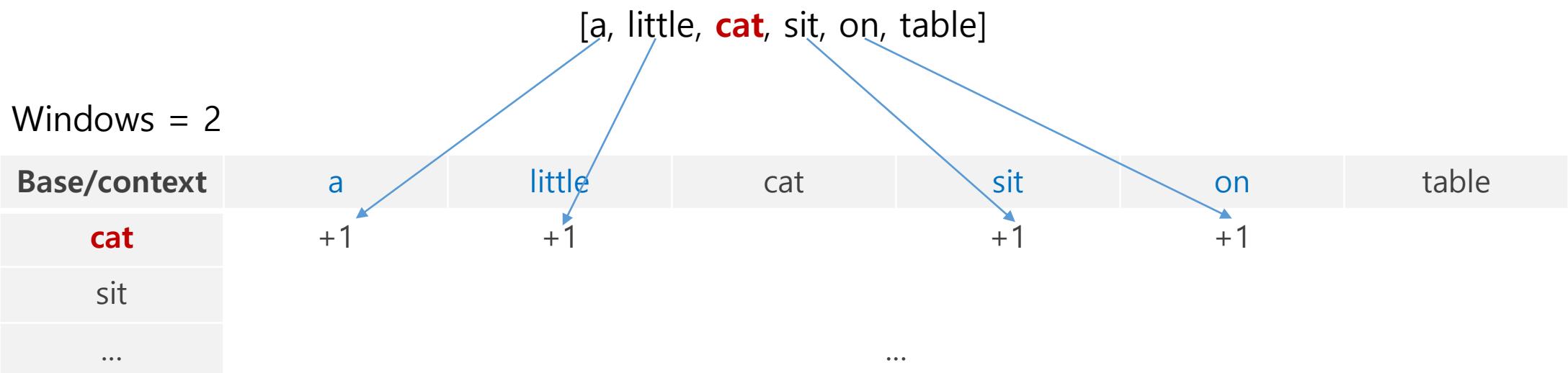
---

- Semantic 을 표현할 대상과 이를 설명하는 정보들을 설정합니다.
  - $x$  는 semantic 을 표현할 대상입니다.
  - $y$  는  $x$  를 설명하는 정보 (context) 입니다.

$$PMI(x, y) = \log \frac{p(y|x)}{p(y)}$$

# Defining contexts

- (term, context terms) 을  $(x, y)$  로 표현할 수 있습니다.



# Defining contexts

- (term, context terms) 을  $(x, y)$  로 표현할 수 있습니다.

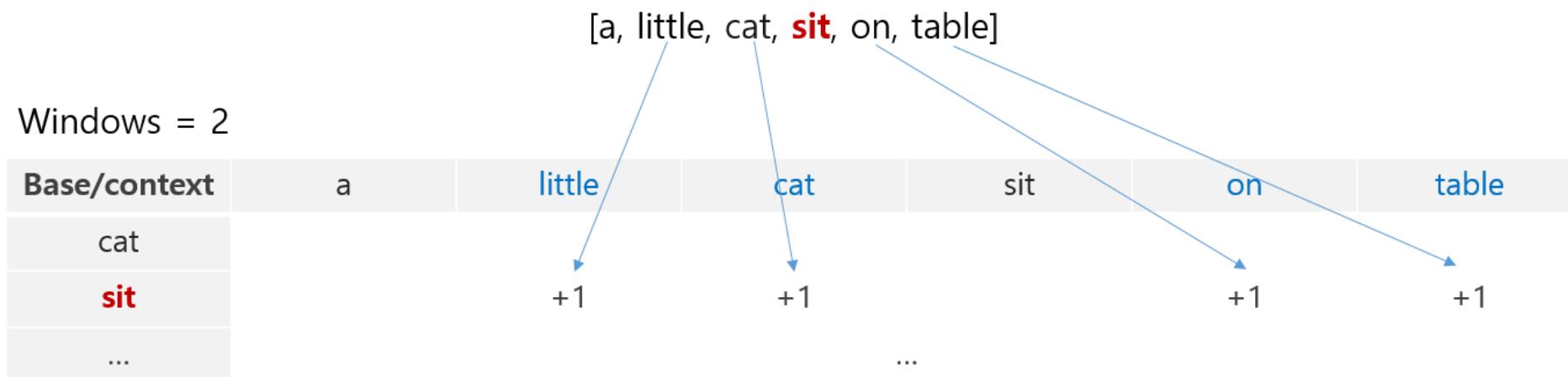
Windows = 2

Base/context	a	little	cat	sit	on	table
cat		+1	+1			
<b>sit</b>					+1	+1
...				...		

[a, little, cat, **sit**, on, table]

# Defining contexts

- (word – context) pair 은 Word2Vec 의 개념과 유사합니다.
  - Levy & Goldberg (2014, NIPS) 에서 Word2Vec 은 PMI 행렬에 차원축소 기법을 적용한 것과 비슷하다는 사실이 밝혀졌습니다.



# Defining contexts

---

- (term, context terms) 을 ( $x, y$ ) 로 표현할 수 있습니다.

from message = ['지금', '어디', '야']  
response = ['신도림', '이야']

Base/context	지금	어디	야	신도림	이야	...
지금	↑					
어디				↑		
야						

# Packages

---

- <https://github.com/lovit/soynlp> : PMI computing modules

```
from soynlp.word import pmi
from soynlp.word import sent_to_word_context_matrix

# create word - context matrix
x = sent_to_word_context_matrix(sents, windows=3, min_tf=10, tokenizer=lambda x:x.split(), verbose=True)

# computing pmi score
pmi_dok_matrix = pmi(x, min_pmi=0, verbose=True)
```

# Packages

---

- <https://github.com/lovit/soynlp> : PMI computing modules

```
from soynlp.word from PMI

pmi_extractor = PMI(windows=3, min_tf=10, verbose=True,
    tokenizer=lambda x:x.split(), min_pmi=0, alpha=0.0001)

pmi_extractor = pmi_extractor.train(corpus)
pmi_extractor.most_similar_words(query)
pmi_extractor.most_related_contexts(query)
```

# Keyword with Proportion ratio

# 키워드란?

---

- 키워드 개념적으로는 이해되지만, 명확히 정의되지 않았습니다.
  - 키워드를 추출하기 전에, **키워드가 무엇인지 정의부터** 해야 합니다.
- 자주 등장한 단어는 키워드가 아닐 수 있습니다.
  - 뉴스에서는 '오늘', '뉴스', '기자'라는 단어는 늘 등장합니다.

# 키워드란?

---

- 키워드 관련 논문들에서 공통적으로 언급되는 키워드의 기준입니다.
  - Saliency (coverage)
    - 한 집합을 대표하는 키워드는 그 집합의 문서에 자주 등장합니다
  - Distinctiveness (discriminative power)
    - 키워드를 이용하면 그 집합과 다른 집합을 구분할 수 있습니다.

# 키워드란?

---

- 키워드를, “한 관점에서 유독 자주 등장하는 단어”로 정의할 수 있습니다.
  - (예시) “오늘”的 키워드, “인물 별” 키워드
  - (예시) 여름 철 평상시 뉴스에서 ‘폭우’가 0.1% 등장합니다. 하지만 오늘 뉴스에서는 ‘폭우’가 1% 등장하였다면, 평상시보다 10 배 더 언급된 단어입니다. ‘폭우’는 오늘 뉴스의 키워드라 할 수 있습니다.

# 상대적 출현 비율을 이용한 키워드 추출

---

- 한 단어를 기준으로, 관심있는 문서 집합 ( $D_T$ )에서의 단어 등장 비율과 비교 대상 문서 집합 ( $D_R$ )에서의 등장 비율을 이용하여 키워드 점수를 정의합니다.

$$score_{keyword}(w) = \frac{P(w|D_T)}{P(w|D_T) + P(w|D_R)}$$

$P(w|D_T)$ : 단어  $w$ 의 관심있는 문서 집합에서의 등장 비율

$P(w|D_R)$ : 단어  $w$ 가 비교 문서 집합 (평상시)에서의 등장 비율

# 상대적 출현 비율을 이용한 키워드 추출

---

- 한 단어를 기준으로, 관심있는 문서 집합 ( $D_T$ )에서의 단어 등장 비율과 비교 대상 문서 집합 ( $D_R$ )에서의 등장 비율을 이용하여 키워드 점수를 정의합니다.

$$\begin{aligned} score_{keyword}('폭우') &= \frac{P('폭우'|D_T)}{P('폭우'|D_T) + P('폭우'|D_R)} \\ &= \frac{1 \%}{1 \% + 0.1 \%} = \frac{1}{1.1} = 0.909 \end{aligned}$$

# 상대적 출현 비율을 이용한 키워드 추출

- 제안된 방법은 **해석력**이 있습니다.
  - 제안된 점수는 [0, 1] 범위 안의 keyword score 를 가질 수 있습니다.

$$score_{keyword}(w) = \frac{P(w|D_T)}{P(w|D_T) + P(w|D_R)}$$

단어 $w$ 가 $D_T$ 에만 등장한 경우	단어 $w$ 가 $D_T$ 와 $D_R$ 에 동일하게 등장한 경우	단어 $w$ 가 $D_R$ 에만 등장한 경우
$\frac{0.01}{0.01 + 0} = 1$	$\frac{0.01}{0.01 + 0.01} = 0.5$	$\frac{0}{0 + 0.01} = 0$

# 상대적 출현 비율을 이용한 키워드 추출

- 레이블링을 할 문서 군집을  $D_T$ 로, 그 외의 문서 집합을  $D_R$ 로 정의합니다.
  - 앞선 방법을 이용하면 군집을 해석할 수 있는 labels를 추출할 수 있습니다.

no.	meaning	Keywords
1	렌트카 광고	제주렌트카, 부산출발제주도, 제주신, 이끌림, 제주올레, 왕복항공, 불포함, 제주도렌트카, 064, 롯데호텔, 자유여행, 객실, 제주여행, 특가, 해비치, 제주시, 제주항, 티몬, 2박3일, 올레, 유류, 항공권, 조식, 제주도여행, 제주공항, 2인
2	중고차 매매	최고급형중고, 최고급, 프리미어, 프라임, 2011년식, YF소나타TOP, 2010년식, 풀옵션, 2011년, YF소나타PR, 1인, Y20, 2010년, 완전무사고, 판매완료, 군포, 검정색, YF쏘나타, 2011, 하이패스, 2010, 무사고, 등급, 파노라마, 허위매물
3	클래식 음악	금관악기, 아이엠, Tru, 트럼펫, 트럼, 나팔, 금관, 텔레만, Eb, 호른, 오보에, Tr, Concerto, 하이든, 협주곡, Ha, 악기, 연주하는, 오케, 오케스트라, 독주, 악장, 작곡가, 곡
4	아이비 "유혹의 소나타 "	Song, 공부할, 부른, 노래, 가사, 부르는, 가수, 보컬, 목소리, 발라드, 명곡, 신나, 들으면, 듣기, 유혹의, 앨범, 아이비, 제목
5	광염 소나타 및 일제강점기 소설들	백성수, 발가락, 현진, 이광수, 김유, 자연주의, 친일, 평양, 운수, 유미, 저지르, 야성, 탐미, 김동인, 복녀, 광염, 맑았다, 사실주의, 광기, 저지, 1920, 단편소설, 범죄, 감자, 동인, 한국문학

# 연관어 분석

---

- 기준 단어가 등장한 문서 집합의 키워드는 기준 단어의 연관어입니다.

$$S(w) = \frac{P(w|D_S)}{P(w|D_S) + P(w|\widetilde{D}_S)}$$

$P(w|D_S)$ : 기준 단어 S가 등장한 문서 집합  $D_S$ 에서 단어 w의 등장 비율

$P(w|\widetilde{D}_S)$ : 기준 단어 S가 등장하지 않은 문서 집합  $\widetilde{D}_S$ 에서 단어 w의 등장 비율

# 키워드 / 연관어 분석

- 2016-10-20, 하루치 뉴스에서의 연관어 (명사) 분석 결과 (빈도수, 점수)

Seed word: 아이오아이	엠카운트다운 (221, 1.00)	잠깐 (162, 0.99)	타이틀곡 (311, 0.99)
방탄소년단 (638, 0.98)	키미 (297, 0.98)	보컬 (155, 0.98)	에이핑크 (237, 0.98)
유정 (161, 0.98)	파워풀 (152, 0.98)	형은 (311, 0.98)	프로듀스 (185, 0.98)
샤이니 (299, 0.98)	불독 (1212, 0.98)	다이아 (182, 0.98)	음반 (204, 0.98)
컴백 (536, 0.98)	세이 (267, 0.98)	순위 (259, 0.98)	콘셉트 (320, 0.98)
멤버들 (504, 0.98)	소라 (262, 0.97)	무대 (1332, 0.97)	발랄 (250, 0.97)
언니 (172, 0.97)	진영 (304, 0.97)	뮤직 (195, 0.97)	서바이벌 (203, 0.97)
싱글 (432, 0.97)	당당 (242, 0.97)	걸그룹 (1060, 0.96)	사운드 (189, 0.96)
각오 (168, 0.96)	강렬 (352, 0.96)	101 (341, 0.96)	실감 (167, 0.96)
쇼케이스 (549, 0.95)	작사 (230, 0.95)	1위 (1357, 0.95)	데뷔 (1365, 0.95)
미니앨범 (197, 0.95)	멤버 (624, 0.95)	프로듀서 (223, 0.95)	신곡 (400, 0.94)
신인 (328, 0.94)	일산 (194, 0.94)	뉴스1스타 (357, 0.94)	롤링 (391, 0.94)

# 키워드 / 연관어 분석

- 2016-10-20, 하루치 뉴스에서의 연관어 (명사) 분석 결과 (빈도수, 점수)

Seed word: 트와이스	가온차트 (191, 1.00)	스트리밍 (329, 1.00)	미니앨범 (197, 1.00)
1주년 (201, 1.00)	티저 (289, 0.99)	두번째 (201, 0.99)	뮤직비디오 (553, 0.99)
타이틀 (270, 0.99)	누적 (799, 0.99)	아이돌 (301, 0.98)	유튜브 (473, 0.98)
컴백 (536, 0.98)	0시 (190, 0.98)	1위 (1357, 0.98)	프로모션 (198, 0.98)
음반 (204, 0.98)	1억 (415, 0.98)	타이틀곡 (311, 0.97)	93 (181, 0.97)
데뷔 (1365, 0.97)	코너 (239, 0.97)	맞은 (316, 0.97)	맞이 (293, 0.97)
걸그룹 (1060, 0.97)	한류 (297, 0.97)	앞둔 (370, 0.97)	판매량 (231, 0.96)
2016년 (1337, 0.96)	대만 (251, 0.96)	페스티벌 (334, 0.96)	팬들 (999, 0.96)
신곡 (400, 0.96)	아이오아이 (270, 0.96)	잠실 (188, 0.95)	24일 (1136, 0.95)
콘서트 (463, 0.95)	입증 (297, 0.95)	6개월 (382, 0.95)	음원 (318, 0.95)
4월 (823, 0.95)	콘셉트 (320, 0.95)	축하 (189, 0.95)	73 (246, 0.94)
블랙핑크 (190, 0.94)	번째 (1158, 0.94)	돌파 (569, 0.94)	문구 (152, 0.94)

# 키워드 / 연관어 분석

- 2016-10-20, 하루치 뉴스에서의 연관어 (명사) 분석 결과 (빈도수, 점수)

Seed word: 박근혜	수석비서관회의 (208, 1.00)	재단들 (152, 1.00)	연설문 (204, 0.99)
누구라 (178, 0.99)	불법행위 (240, 0.99)	퇴임 (188, 0.98)	엄정 (388, 0.98)
창조경제 (226, 0.98)	처벌받 (227, 0.98)	미르 (604, 0.98)	스포츠재단 (676, 0.97)
더블루케이 (194, 0.97)	최씨 (695, 0.97)	재단 (1690, 0.97)	자유학기제 (201, 0.97)
비선실세 (219, 0.97)	최순실씨 (520, 0.97)	미르재단 (247, 0.96)	게이트 (303, 0.96)
대통령 (5682, 0.96)	모녀 (223, 0.96)	행복교육 (227, 0.95)	실세 (309, 0.95)
비선 (288, 0.95)	최순실 (1318, 0.95)	의혹 (3602, 0.95)	고양 (278, 0.95)
국정 (185, 0.94)	청와대 (2112, 0.94)	지지층 (151, 0.94)	킨텍스 (332, 0.94)
체육 (221, 0.94)	재계 (152, 0.93)	민생 (164, 0.93)	2002년 (186, 0.93)
정권 (596, 0.93)	가중 (175, 0.93)	유용 (359, 0.93)	전경련 (348, 0.93)
주재 (459, 0.93)	국민들 (441, 0.93)	백승 (216, 0.92)	갤러리 (271, 0.92)
기업들 (808, 0.92)	지지율 (336, 0.92)	확산 (800, 0.91)	철저히 (327, 0.91)

# 키워드 / 연관어 분석

- 2016-10-20, 하루치 뉴스에서의 연관어 (명사) 분석 결과 (빈도수, 점수)

Seed word: 최순실	게이트 (303, 1.00)	정유라 (329, 1.00)	연설문 (204, 0.99)
모녀 (223, 0.99)	비선 (288, 0.99)	더블루케이 (194, 0.98)	실세 (309, 0.98)
스포츠재단 (676, 0.98)	최씨 (695, 0.98)	최경희 (223, 0.98)	이화여대 (651, 0.98)
특혜 (532, 0.98)	미르재단 (247, 0.98)	학점 (191, 0.98)	비선실세 (219, 0.98)
이대 (419, 0.97)	미르 (604, 0.97)	재단 (1690, 0.97)	정유라씨 (200, 0.97)
엄정 (388, 0.97)	사퇴 (463, 0.96)	의혹 (3602, 0.96)	누구라 (178, 0.96)
사임 (245, 0.96)	교수들 (183, 0.96)	입학 (356, 0.96)	창조경제 (226, 0.96)
최순실씨 (520, 0.95)	수석비서관회의 (208, 0.95)	총장 (1215, 0.95)	문체부 (268, 0.95)
국정 (185, 0.95)	색깔론 (160, 0.95)	침묵 (223, 0.95)	불법행위 (240, 0.95)
모금 (238, 0.95)	재단들 (152, 0.95)	처벌받 (227, 0.95)	본관 (204, 0.95)
비리 (427, 0.94)	청와대 (2112, 0.94)	박근혜 (1445, 0.94)	퇴임 (188, 0.94)
개입 (473, 0.93)	설립 (1522, 0.93)	전경련 (348, 0.93)	더블 (225, 0.93)

## 상대적 출현 비율 vs PMI

---

- 상대적 출현 비율을 이용한 방법은 PMI 와 비슷합니다.
  - 키워드 추출 방법은 (단어, 문서집합) 간의 co-occurrence 를 이용합니다.
  - 연관어 추출 방법은 (단어, 단어) 간의 co-occurrence 를 이용합니다.
- 해석력을 얻기 위해 co-occurrence 를 다른 방식으로 이용하였습니다.

# LASSO regression for keyword extraction

---

- LASSO 선택하는 모델은 두 가지 조건을 만족합니다
  - (1) 분별력이 좋으면서
  - (2) 많은 문서에서 등장한 단어를 우선적으로 선택
- 몇 번 등장하지 않은 단어는 분별력은 좋을 수 있지만, L1 cost를 높입니다
- 동일한 분별력을 가질 때에는 좀 더 자주 등장한 단어를 선택합니다  
(문서에 대한 coverage가 높은 단어가 선택될 가능성이 더 높습니다)

# LASSO regression for keyword extraction

---

- 문서 종류를 명확히 구분하면서도 그 종류의 문서에서 자주 등장한 단어를 키워드로 정의할 수도 있습니다
  - 변별력이 좋고 설명하려는 문서의 다수에서 등장하는 단어 집합을 이용하면, 적은 수의 단어로 해당 문서 집합을 표현할 수 있습니다.
  - 이는 LASSO가 추출하는 features와 keywords의 정의가 일치합니다
- LASSO 는 correlation 이 높은 단어셋이 있다면, 그 중 대표 단어를 선택하여 중복적인 키워드를 제거합니다.
  - 하지만, ['바락', '오바마']가 늘 함께 등장한다면, 하나만 키워드로 선택될 수 있습니다.

# L1 Regularization

T11, T13, T14를 이용하면  $y=1$ 을 완벽히 인식할 수 있고,  
이 문제는 {T1, T2, T3, T11, T13, T14}만 이용해도 잘 풀림

y	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
0	5	3													
0	3	2		5											
0	2	4		4											
0	5		2												
0	1		1												
0	4			1											
0	2				2										
1															
1															
1															
1															
1															
1															
1															

# Packages

- <https://github.com/lovit/soykeyword> : Proportion ratio for keyword

```
from soykeyword.proportion import CorpusbasedKeywordExtractor

corpusbased_extractor = CorpusbasedKeywordExtractor(min_tf=20, min_df=10, tokenizer= lambda x:x.split())
corpusbased_extractor.train(Corpus(tokenized_corpus_fname))
lassobased_extractor.extract_from_word('아이오아이', minimum_number_of_keywords=30)
```

```
[KeywordScore(word='아이오아이', frequency=270, score=1.0),
 KeywordScore(word='엠카운트다운', frequency=221, score=0.997897148491129),
 KeywordScore(word='펜타곤', frequency=104, score=0.9936420169665052),
 KeywordScore(word='잠깐', frequency=162, score=0.9931809154109712),
 KeywordScore(word='엠넷', frequency=125, score=0.9910325251765126),
 KeywordScore(word='걸크러쉬', frequency=111, score=0.9904705029926091),
 KeywordScore(word='타이틀곡', frequency=311, score=0.987384461584851),
 KeywordScore(word='코드', frequency=105, score=0.9871835929954923),
 KeywordScore(word='본명', frequency=105, score=0.9863934667369743),
 ... ]
```

# Packages

---

- <https://github.com/lovit/soykeyword> : Lasso for keywords

```
from soykeyword.lasso import LassoKeywordExtractor

lassobased_extractor = LassoKeywordExtractor(min_tf=20, min_df=10)
lassobased_extractor.train(x, index2word)
lassobased_extractor.extract_from_word('아이오아이', minimum_number_of_keywords=30)
```

```
[KeywordScore(word='너무너무너무', frequency=86, coefficient=3.8159005957233778),
 KeywordScore(word='선의', frequency=40, coefficient=3.2584820410431181),
 KeywordScore(word='산들', frequency=90, coefficient=2.4407245228574896),
 KeywordScore(word='엠카운트다운', frequency=221, coefficient=1.7601587420428146),
 KeywordScore(word='챔피언', frequency=105, coefficient=1.4864913827165669),
 KeywordScore(word='사나', frequency=46, coefficient=1.4183641861333143),
 KeywordScore(word='드림', frequency=119, coefficient=1.3338856375792103),
 KeywordScore(word='뮤직', frequency=195, coefficient=1.1767179765646125),
 KeywordScore(word='먹고', frequency=216, coefficient=1.1632972589808017),
 KeywordScore(word='완전체', frequency=77, coefficient=1.121112062888608),
 KeywordScore(word='일산', frequency=194, coefficient=0.96056172786240313),  
...]
```

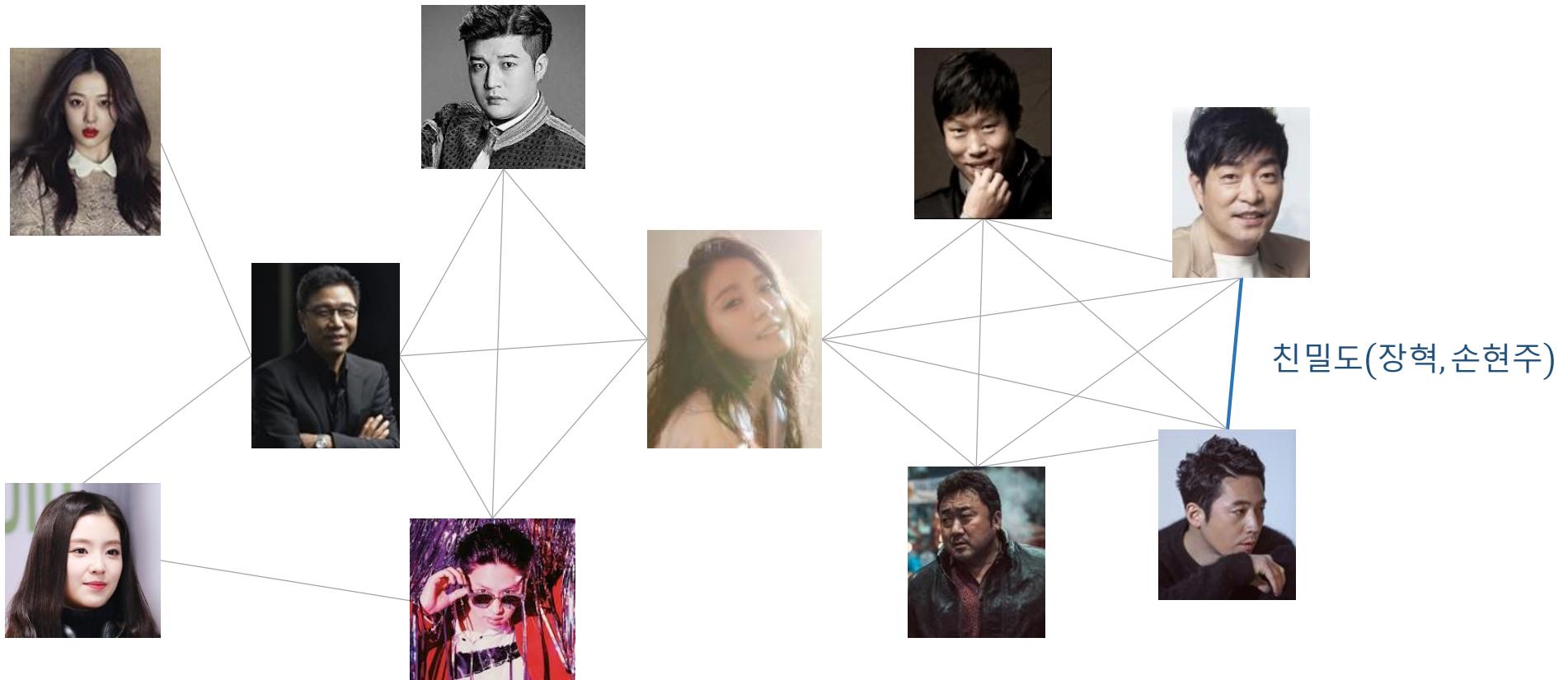
# Introduction

---

- 그래프는 데이터를 표현하는 형식입니다.
  - 그래프는 마디(node, V) 와 호(edge, E)로 이루어져 있습니다.
  - $G = (V, E)$
- 벡터 공간보다 더 자유로운 표현이 가능합니다.
  - $(N1, N2), (N1, N3)$  는 가깝지만,  $(N2, N3)$  이 매우 멀 수 있습니다.

# Introduction

- “Social media user networks”



# Introduction

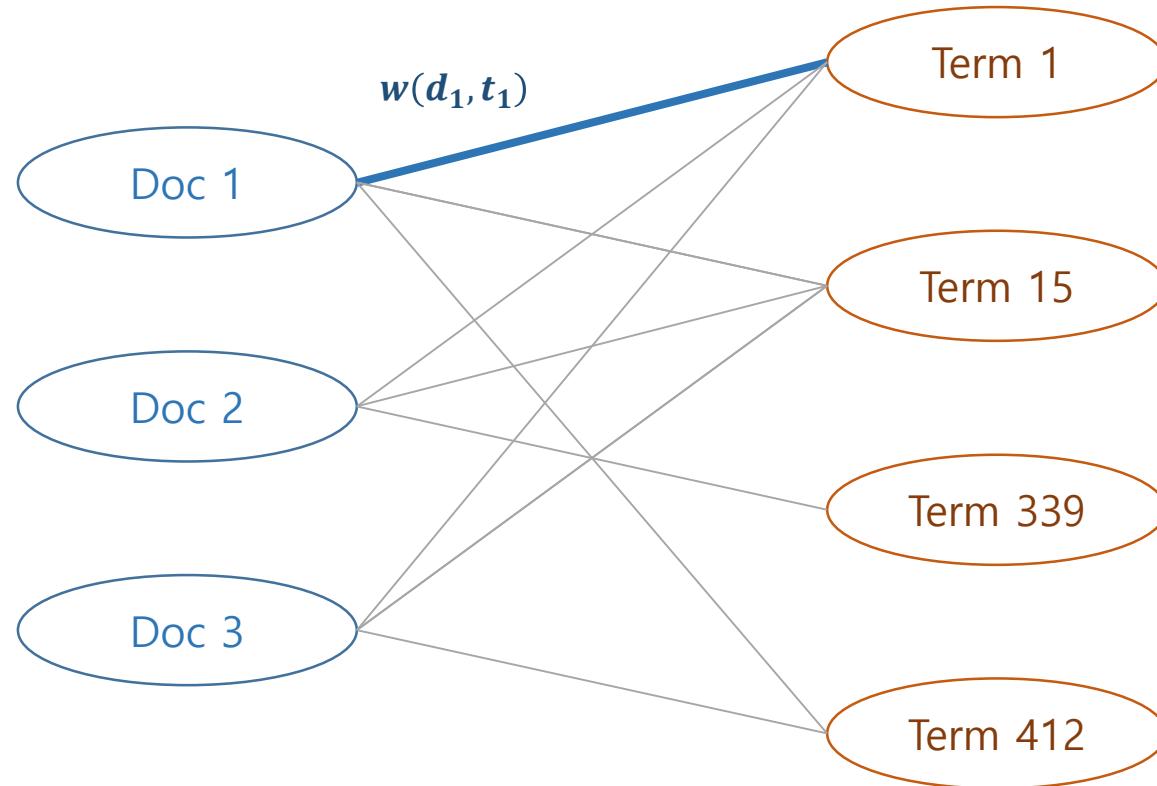
- “영화 – 배우” 그래프



# Introduction

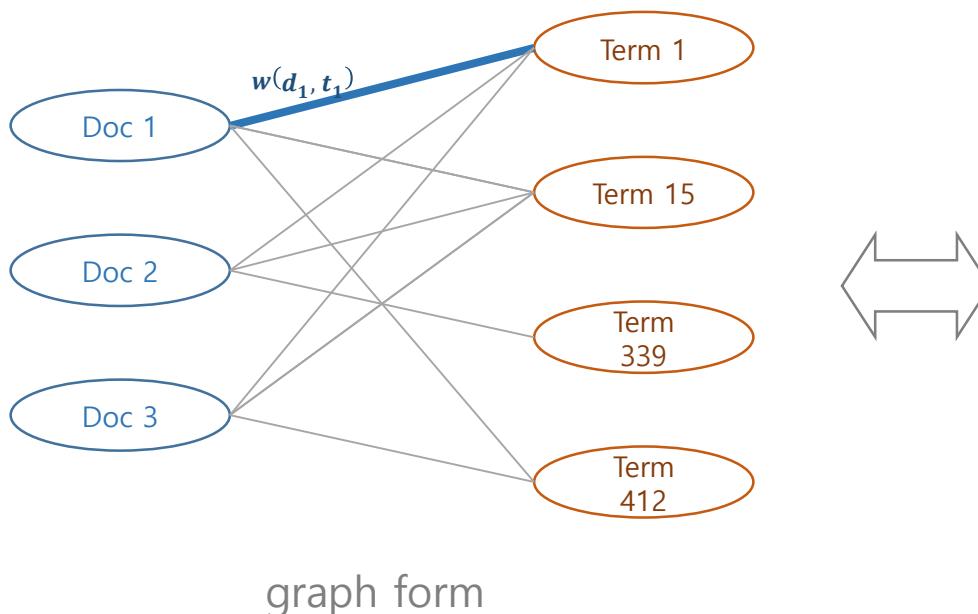
---

- Term document graph



# Introduction

- (row, column) 을 edge 의 시작점과 끝점으로 설정하면 그래프는 행렬로 표현됩니다.
  - Edge 는 두 마디 간의 거리 / 유사도 / 가중치 등의 값을 부여할 수 있습니다.



	1	2	...	15	...	339	412	...
D1	4	-	...	3	...	-	2	...
D2	2	-	...	6	...	4	-	...
D3	5	-	...	4	...	-	5	...
D4	-	2	...	-	...	-	3	...

matrix form

# Introduction

---

- “영화 – 배우”나 “문서 – 단어”처럼 마디의 종류가 두 가지로 나뉘어지고, 서로 다른 종류의 마디끼리만 연결이 된 경우, “bipartite graph”라 합니다.

# Introduction

---

- 그래프 형식으로 표현된 데이터에 대한 대표적인 질문은
  - Which nodes are **important**?
  - Which nodes are **similar** with given node?

Which nodes are **important**?

# Which nodes are important?

---

- 마디의 중요도를 정의하는 다양한 statistics 이 있습니다.
  - 연결된 edge 의 개수
  - 연결된 edge 의 weights 의 총합
  - ...
- PageRank 는 그래프의 구조를 바탕으로 마디의 중요도를 정의합니다.

# PageRank

---

- PageRank<sup>[1]</sup> 는 Google 의 검색 결과의 랭킹을 위하여 개발된 방법입니다.
  - 질의어를 포함한 웹문서는 수십만개입니다.
  - 그 중 가장 적절한 웹문서를 선택하기 위하여 웹페이지의 중요도를 정의해야 했습니다.

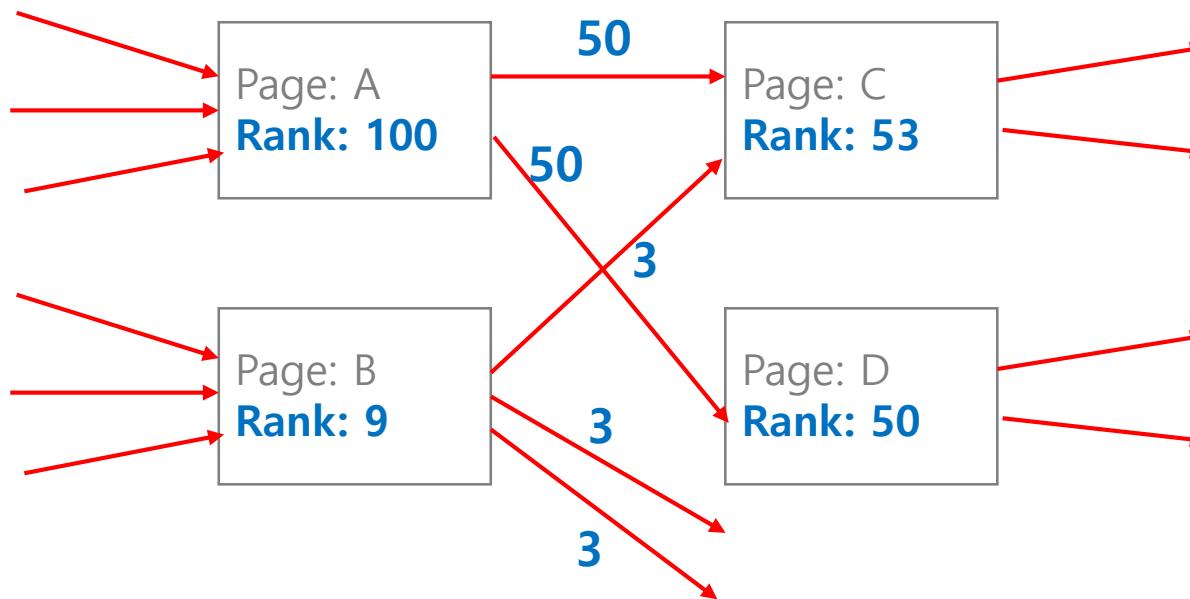
# PageRank

---

- PageRank 는 웹공간의 hyper link 구조를 이용하여 중요도를 계산합니다.
  - 많은 back-links (자신으로 유입되는 링크)를 가진 페이지는 중요할 가능성이 높습니다.
  - 중요한 페이지의 hyper link 는 그렇지 않은 페이지보다 더 영향력이 있어야 합니다.

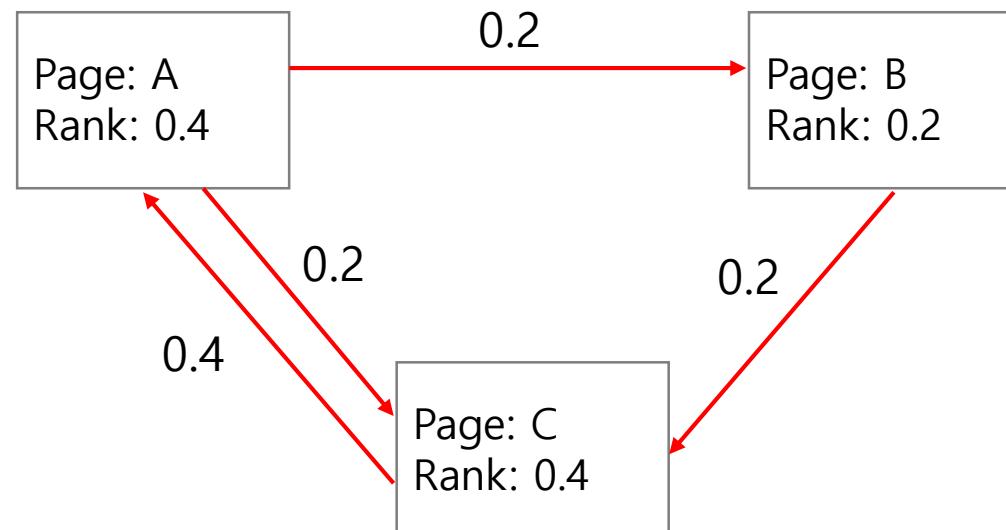
# PageRank

- PageRank 는 각 페이지의 중요도를, 연결된 페이지에 골고루 나눠줍니다.
  - 연결된 웹페이지에 투표를 하는 것과 같습니다.
  - 100 의 중요도를 지닌 페이지의 중요도는 두 페이지에 50 씩 나눠집니다.



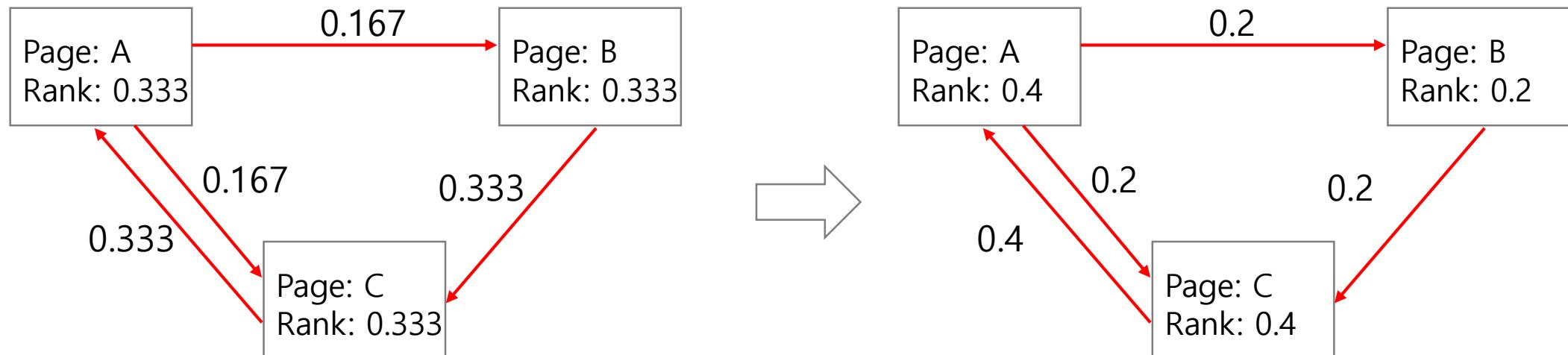
# PageRank

- PageRank 는 iterative 한 방법으로 각 페이지의 중요도를 계산합니다.
  - Rank 의 균형이 맞는 상태가 존재합니다.
  - 그러나 이 균형을 처음부터 알지는 못합니다.



# PageRank

- PageRank 는 iterative 한 방법으로 각 페이지의 중요도를 계산합니다.
  - 초기화 때 모든 페이지에 같은 rank 를 부여합니다.
  - $PR(u) = c \sum_{v \in B_u} \frac{PR(v)}{N_v}$  를 이용하여 모든 웹페이지  $u$  의 중요도를 계산합니다.



# PageRank

---

- PageRank 는 **iterative** 한 방법으로 각 페이지의 중요도를 계산합니다.
  - Ant (random surfer) models 로 직관적인 설명이 가능합니다.
  - 개미들이 그래프 위의 마디를 무한히 이동하면 많은 개미가 몰리는 마디가 생겨납니다.
  - 몰려있는 개미의 상대적인 양이 마디들의 중요도입니다.
  - 웹공간에서는 각 페이지에 사용자들이 머무는 기대시간이기도 합니다.

# PageRank

---

- Out links 가 없는 마디는 PageRank 가 제대로 작동하지 않게 만듭니다.
  - Out links 가 없으면 Iteration 이 지속될수록 누적된 개미의 양이 커집니다.
  - 웹페이지에서는 링크를 타고 유입된 사람이 다른 페이지를 더 이상 보지 않는 것과 같습니다.

# PageRank

---

- Random jump 를 이용하여 문제를 해결합니다.

$$\bullet \ PR(u) = c \sum_{v \in B_u} \frac{PR(v)}{N_v} + (1 - c) \frac{1}{N}$$

- 각 페이지에서  $c$  만큼의 rank 만 out links 를 이용하고,  $1 - c$  만큼은 모든 페이지에 임의로 이동합니다.

- $(1 - c) \frac{1}{N}$  만큼 모든 페이지로부터 유입되었다는 의미입니다.

- $c$  는 survival rate 입니다.  $0 < c < 1$  의 값을 이용합니다. (default = 0.85)

# PageRank

---

- Algorithm

Input: [Graph  $G$ , weight  $c$ ]

Output: [PageRank  $PR(u)$ ]

---

1. Initialize  $PR(u)$  with  $\frac{1}{N}$
2. While not converged, iterate

$$PR(u) = c \sum_{v \in B_u} \frac{PR(v)}{N_v} + (1 - c) \frac{1}{N}$$

# PageRank

---

- $PR(u)$  의 수렴은 매우 빠릅니다.
  - Iteration 마다  $PR(u)$  의 차이는 지수승으로 줄어듭니다.
  - 322 M links 인 웹그래프에서도 iteration = 50 이면 충분합니다.

# Personalized PageRank

---

- PageRank 는 웹페이지의 **global importance** 를 계산합니다.
  - 개인의 검색 이력 등의 정보를 이용하지 않은 중요도입니다.
- Bias 는 각 페이지에 대한 개인의 선호 (**preference**) 입니다.
  - Bias,  $PR(u) = c \sum_{v \in B_u} \frac{PR(v)}{N_v} + (1 - c) \frac{1}{N}$  를 조절하면 개인의 성향 / 상황을 고려한 ranking 이 가능합니다.
  - $(1 - c) \frac{1}{N}$  은 선호가 없음을 의미합니다.

# PageRank

---

- PageRank 에서 기억해야 할 점은 마디의 중요도를 정의하는 방식입니다.

중요한 마디로부터 많은 투표 (back-links)를 받는 마디가 중요한 마디

# PageRank

---

- PageRank 의 구현체는 다양합니다.
- soygraph 는 텍스트 데이터의 handling 과 효율적인 계산을 위해 제안된 방법들을 구현하려는 프로젝트입니다.

# PageRank

```
from soygraph import dict_to_matrix

dd = {0: {1: 0.037, 55:0.025}, 1: {83:32, ... }, ... }

x = dict_to_matrix(dd)
print(type(x)) # <class 'scipy.sparse.csr.csr_matrix'>
print(x.shape) # (15097, 15097)

from soygraph.ranking import PageRank

pagerank = PageRank(
    damping_factor=0.85, max_iter=30, ranksum=1.0,
    verbose=True, converge_threshold=0.0001
)

rank_value = pagerank.rank(x)
rank_value = pagerank.rank(dd)
```

# HITS algorithm

---

- HITS<sup>[1]</sup> 는 웹검색의 ranking 을 위해 제안된 알고리즘입니다.
  - PageRank 와 비슷한 방식입니다.
  - 논문<sup>[1]</sup>에 기술된 알고리즘 전체의 디테일은 다릅니다만, ranking 방식은 비슷합니다.
  - HITS 는 각 마디(웹페이지)에 대하여 authority, hub 의 ranking 을 계산합니다.

# HITS algorithm

---

- HITS 는 authority 와 hub 를 재귀적(recursive)으로 정의합니다.
  - $authority(u) = \sum_{v \in INB_u} hub(v)$
  - $hub(u) = \sum_{v \in OUTB_u} authority(v)$
- Authority ranking 은 inbound nodes 의 hub ranking 의 합입니다.
- Hub ranking 은 outbound nodes 의 authority ranking 의 합입니다.

# HITS algorithm

---

- 매 iteration마다 **normalization**이 필요합니다.
  - $\sum_{v \in INB_u} hub(v)$ ,  $\sum_{v \in OUTB_u} authority(v)$  을 반복하면 그래퍼 전체의 rank sum이 증가합니다.
  - 매 iteration마다 전체의 rank sum이 유지되도록 정규화를 수행합니다.

# HITS algorithm

---

- PageRank 는 한 마디의 중요도 (ants) 가 연결된 마디들의 개수만큼 나누어져 다른 마디로 이동하는 개념입니다.
- HITS 는 연결된 마디의 개수만큼 증식하여 전파되는 개념입니다.

# HITS algorithm

```
from soygraph.ranking import HITS

hits = HITS(
    damping_factor=0.85, max_iter=30, ranksum=1.0,
    verbose=True, converge_threshold=0.0001
)

authority, hub = hits.rank(x)
authority, hub = hits.rank(dd)
```

- 
- PageRank 와 HITS 는 그래프의 마디에 대한 중요도를 정의 / 학습하는 방법입니다.
    - 두 알고리즘 모두 검색 결과의 ranking 을 위하여 제안되었지만, 다양한 분야에서 응용이 되고 있습니다.
    - 학습에 이용할 데이터가 목적에 적합한 그래프의 형태로 표현되기만 하면 알고리즘을 적용할 수 있습니다.

# TextRank

---

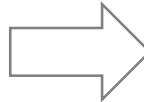
- TextRank<sup>[1]</sup> 은 PageRank 를 이용하여 키워드 / 핵심문장을 추출합니다.
  - 텍스트로부터 단어 / 문장 그래프를 만든 뒤, PageRank를 적용합니다.
- Co-occurrence 가 있는 두 단어는 edge 로 연결됩니다.
  - Co-occurrence 는 window 안에 포함된 경우입니다.
  - Edge 의 값은 co-occurrence 입니다.

# TextRank

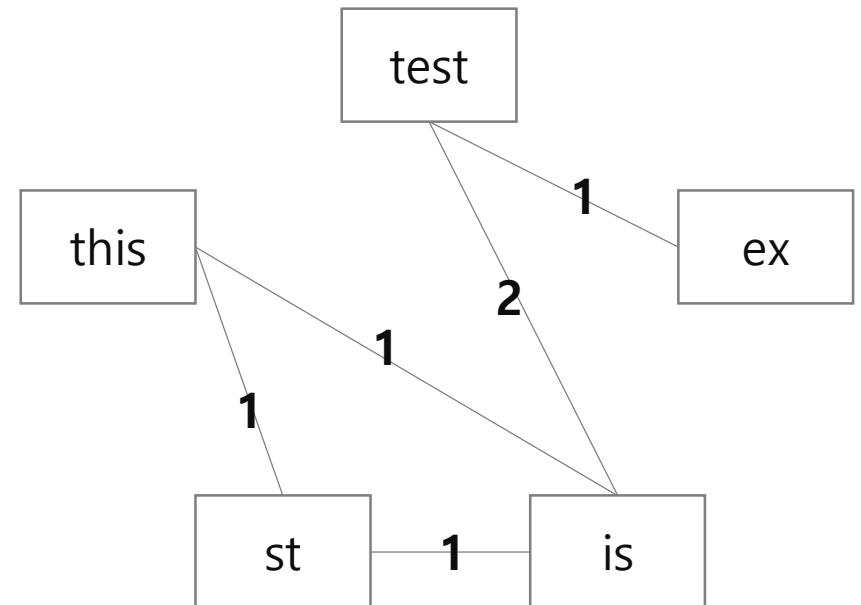
---

window = 1

doc = [[this, is, test, ex],  
[this, st, is, test]]



texts



word graph

# TextRank

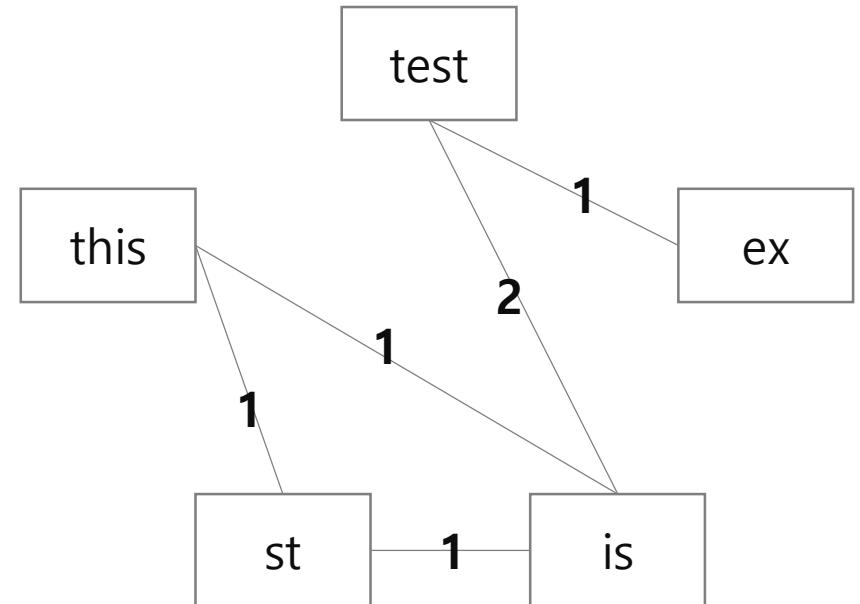
---

- $R(V_i) = (1 - d) + d \times \sum_{v_j \in In(V_i)} \frac{w_{ji}}{\sum_{v_k \in Out(V_j)} w_{jk}} R(V_j)$

- Weighted PageRank version

- $R(this) = 0.1 + 0.9 \times \left( \frac{1}{1} \times R(st) + \frac{1}{3} \times R(is) \right)$

with  $d = 0.9$



# TextRank

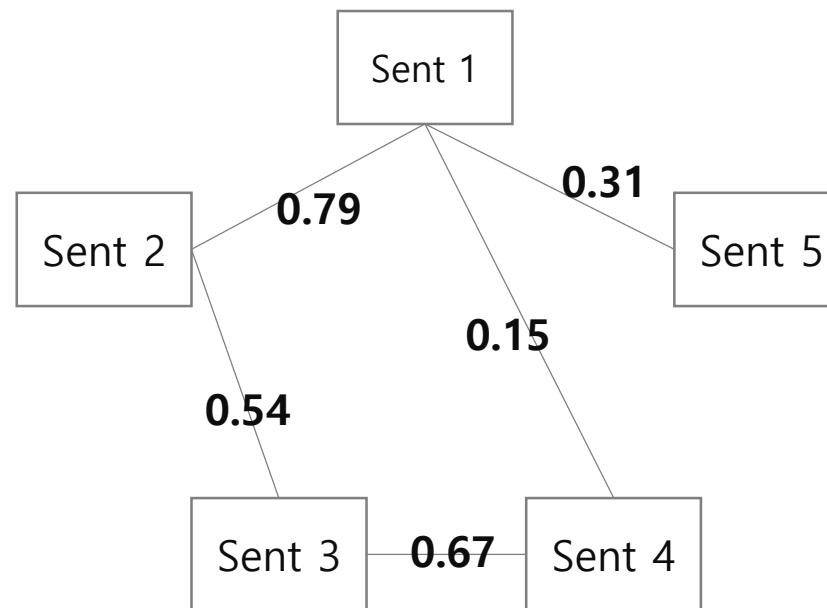
---

- “중요한 단어 주위에 있는 단어는 중요하다” 라는 가정을 바탕으로 word graph 를 구성합니다.
- 빈도수가 많은 단어가 높은 rank 를 가질 가능성이 높습니다.

# TextRank

---

- Sentence graph 는 모든 문장 간의 유사도를 edge weight 로 구성합니다.
  - 문장 간의 유사도는 임의의 유사도를 이용할 수 있습니다.



# TextRank

---

- Sentence graph 는 모든 문장 간의 유사도를 edge weight 로 구성합니다.
  - 각 문장  $S_i = w_1^i, w_2^i, \dots, w_{N_i}^i$  를  $N_i$  개의 단어 집합으로 생각합니다.
  - 문장 간 유사도는 아래의 식으로 정의합니다.

$$\text{similarity}(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \& w_k \in S_j\}|}{\log(|S_i|) + \log(|S_j|)}$$

## TextRank

---

- 많은 문장들과 높은 유사도로 연결이 되어 있는 문장은, 빈번히 등장하는 단어들을 다수 포함할 가능성이 높습니다.
- Sentence graph 를 이용하는 TextRank 는 빈번한 단어를 다수 포함한 문장을 핵심 문장으로 선택합니다.

# TextRank

- 문장으로부터 word / sentence graph 를 만들어 PageRank 를 적용합니다.

```
from textrank import summarize_as_keywords

keywords = summarize_as_keywords(sents, topk=50,
    tokenizer=lambda s:s.split(), min_count=10,
    min_cooccurrence=3, verbose=True, debug=True
)
```

```
[('재배포', 0.538140850221495),
 ('무단', 0.46747526750507146),
 ('금지', 0.3797005381404317),
 ('뉴시스', 0.1889406802065108),
 ('공감', 0.10557622894724966),
 ('저작권자', 0.07848823486967427),
 ... ]
```

# TextRank

---

- 문장으로부터 word / sentence graph 를 만들어 PageRank 를 적용합니다.

```
from textrank import summarize_as_sentences
```

```
keywords = summarize_as_sentences (sents)
```

# TextRank (gensim)

---

- Gensim 은 TextRank 를 이용한 summarizer 를 제공합니다.
  - Gensim 은 여러 문장에서 핵심 문장을 찾는 함수와
  - 여러 문서에서 핵심 문서를 찾는 함수를 제공합니다.

```
from gensim.summarization.summarizer import summarize  
from gensim.summarization.summarizer import summarize_corpus
```

```
text = """Rice Pudding - Poem by Alan Alexander Milne  
... What is the matter with Mary Jane?  
... She's crying with all her might and main,  
... And she won't eat her dinner - rice pudding again -  
... What is the matter with Mary Jane? ..."""
```

```
print(summarize(text))
```

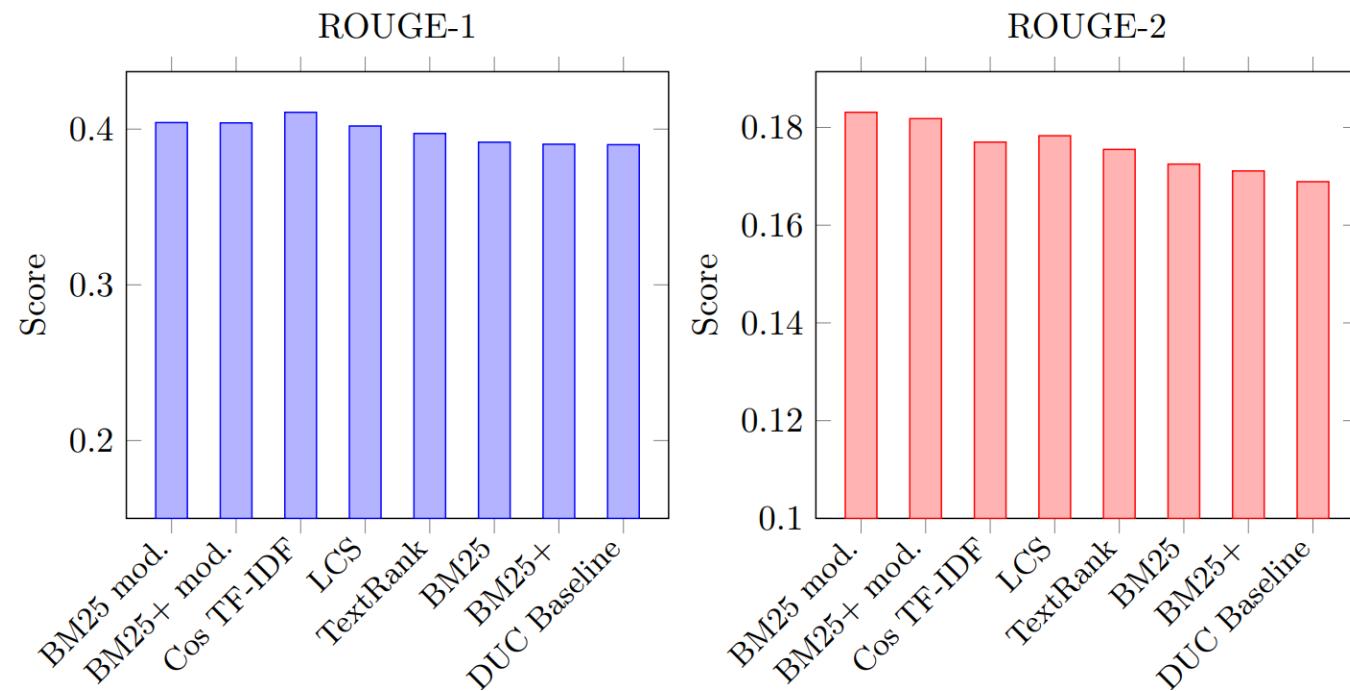
## TextRank (gensim)

---

- Gensim 은 TextRank 를 이용한 summarizer 를 제공합니다.
  - Gensim 에 구현된 summarizer 는 TextRank 의 변형입니다.
  - Barrios et al., (2016) 은 TextRank 의 graph 를 만들 때, BM25+ 점수를 edge weight 로 이용하였습니다.

# TextRank (gensim)

- TextRank 의 원형과 gensim summarizer 의 성능은 2002 Document Understanding Conference (DUC) 를 이용하여 ROGUE score 로 평가하였습니다.



# LexRank

---

- Sentence graph 구성 시, 디테일이 TextRank 와 다릅니다.
  - 문장 간 유사도를 TF-IDF cosine 을 이용합니다.
- 문장 간 유사도의 최소값의 threshold 를 이용합니다.
  - 적절한 threshold 를 설정하면 sparse graph 를 만들 수 있으며,
  - 핵심 문장이 잘 추출됩니다.

# WordRank

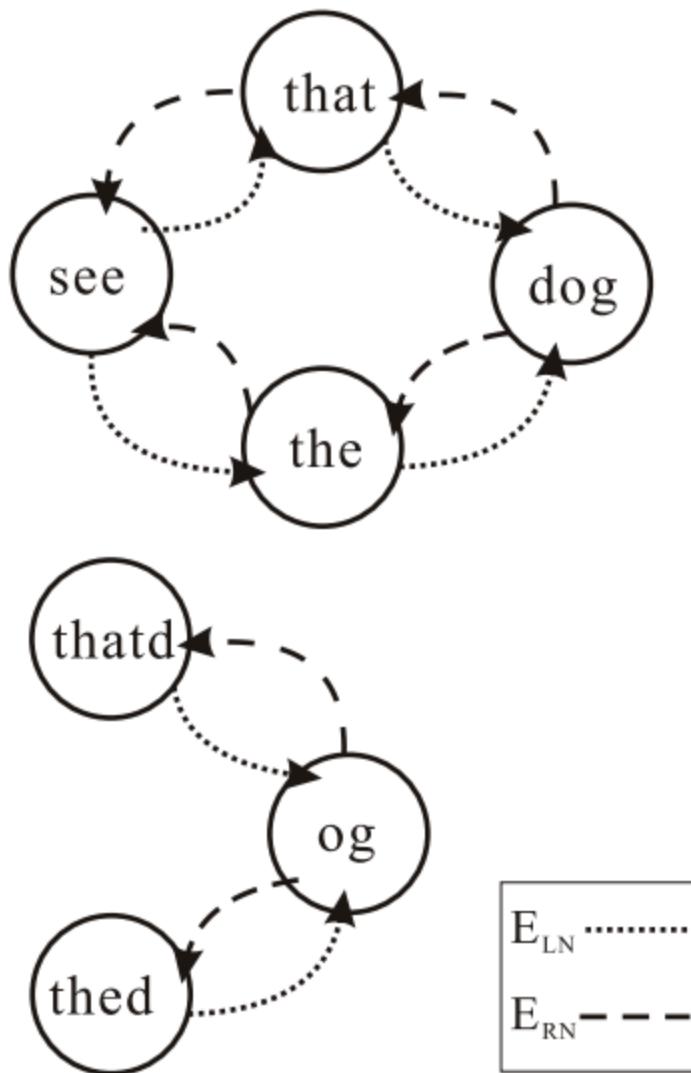
---

- TextRank 는 단어열로 분해된 문장으로부터 키워드를 추출합니다.
- Substring graph 를 이용하면 데이터 기반으로 단어 추출도 가능합니다.
  - WordRank 는 중국/일본어에서 단어를 추출하기 위하여 제안된 방법입니다.
  - “**단어의 이웃은 단어**이며, 단어가 아닌 substring 의 이웃은 잘못된 substring 이다”는 가정을 이용합니다.

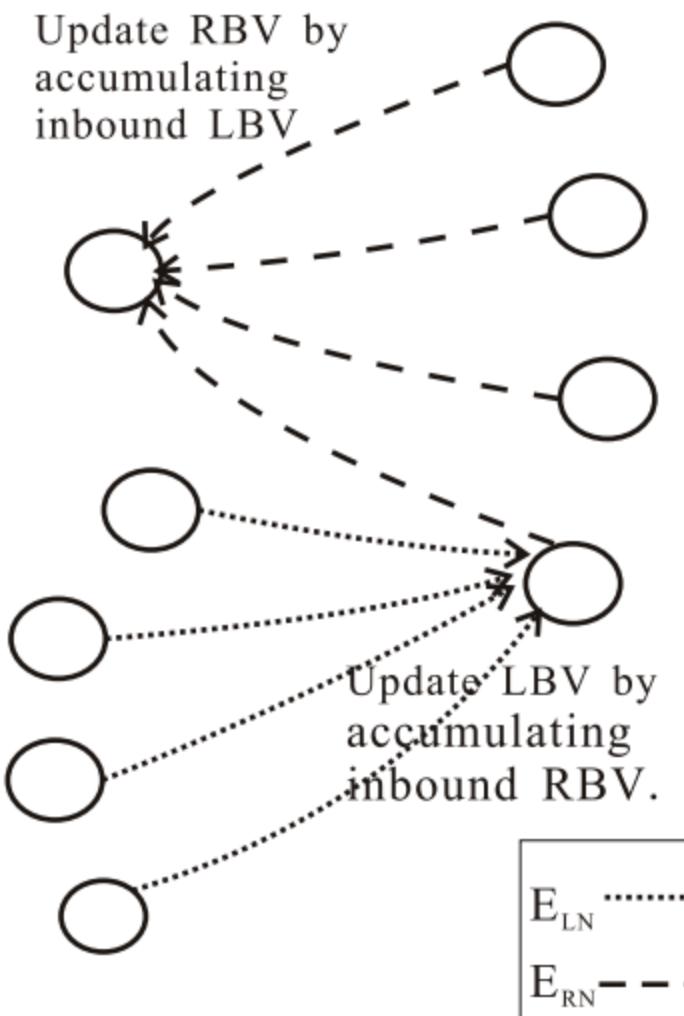
do you see that dog  
i dont see the dog  
there is the dog

doyouseethatdog  
idontseethedog  
thereisthedog

e:	8	the:	3
o:	6	edo:	2
do:	5	hedo:	2
og:	3	seeth:	2
he:	3	hedog:	2
dog:	3	...	



(a) Segmented, unsegmented corpus and (b) Illustration of the link structure (partial) valid word hypotheses



(c) Illustration of iterative updating

# KR-WordRank

---

- 한국어 텍스트에는 WordRank 적용이 어렵습니다.
  - 한국어에서 사용되는 글자 수가 적기 때문에 1음절 단어가 주요 단어로 추출
  - 띄어쓰기 정보를 무시하면 잘못된 단어후보가 단어로 추출될 수 있습니다.
    - “너는지난주” vs “너는 지난주”
  - 추출해야 하는 단어는 어절의 왼쪽에 위치하는 substring 뿐입니다.
    - “트와이스는”

# KR-WordRank

---

- Packages
  - <https://github.com/lovit/kr-wordrank>
  - > pip install krwordrank

```
from krwordrank.word import KRWordRank

min_count = 5    # 단어의 최소 출현 빈도수 (그래프 생성 시)
max_length = 10 # 단어의 최대 길이
wordrank_extractor = KRWordRank(min_count, max_length)

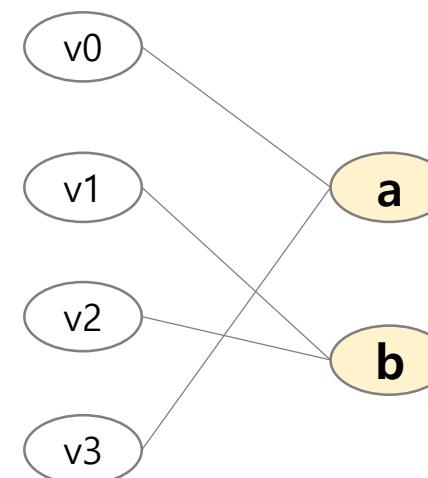
texts = ['예시 문장 입니다', '여러 문장의 list of str 입니다', ... ]
keywords, rank, graph = wordrank_extractor.extract(texts)
```

Which nodes are **similar**?

# Which nodes are **similar**?

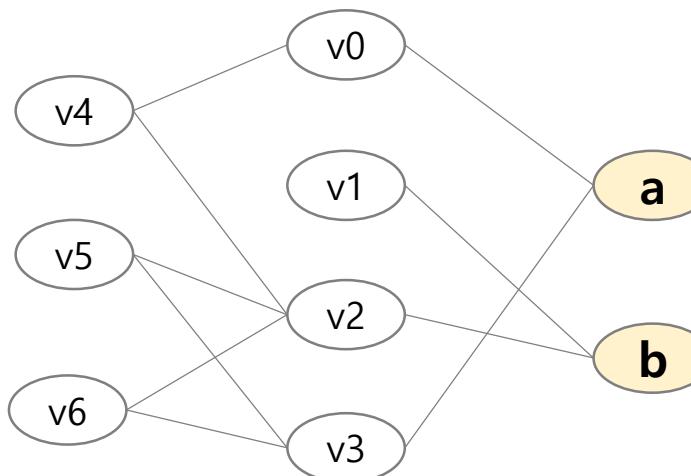
---

- 두 마디 (a, b)의 이웃이 유사하면 (a, b)는 비슷하다 정의할 수 있습니다.
  - (a, b) 의 이웃 벡터에 대한 Cosine / Jaccard similarity 는 0입니다.
  - (a, b) 를 문서, (v0, ... v3) 을 단어로 생각하면 (a, b) 는 동일한 단어를 공유하지 않습니다.



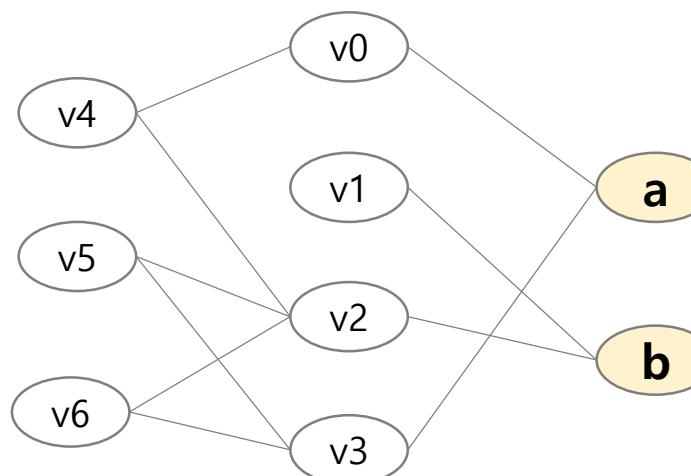
# Which nodes are similar?

- 두 마디 (a, b)의 이웃이 유사하면 (a, b)는 비슷하다 정의할 수 있습니다.
  - ( $v_2, v_3$ ) 는 공유하는 이웃이 많기 때문에 비슷한 마디입니다.
  - ( $v_2, v_3$ ) 의 유사성을 고려하면 (a, b) 도 비슷한 마디여야 합니다.
  - 이웃의, 이웃의, ... 이웃을 고려합니다.



# SimRank

- SimRank<sup>[1]</sup> 는 그래프의 구조를 반영하여 마디 간 유사도를 정의합니다.



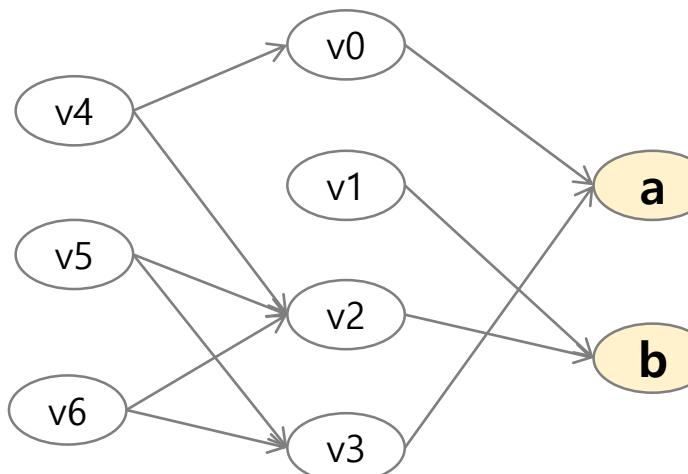
# SimRank

- SimRank 도 재귀적으로 마디 간 유사도를 정의합니다.

$$S(a, b) = \frac{C}{|I(a)||I(b)|} \sum_{i \in I(a)} \sum_{j \in I(b)} S(i, j)$$

$I(a)$ : 마디  $a$  의 inbounds

$I(b)$ : 마디  $b$  의 inbounds

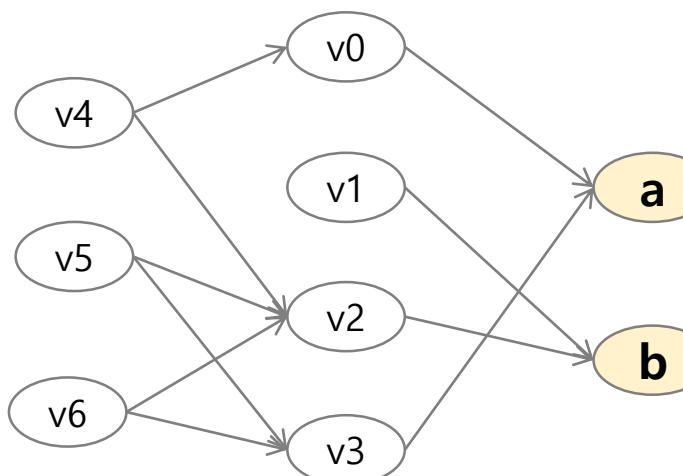


# SimRank

- SimRank 의 학습 역시 iterative 하게 진행됩니다.

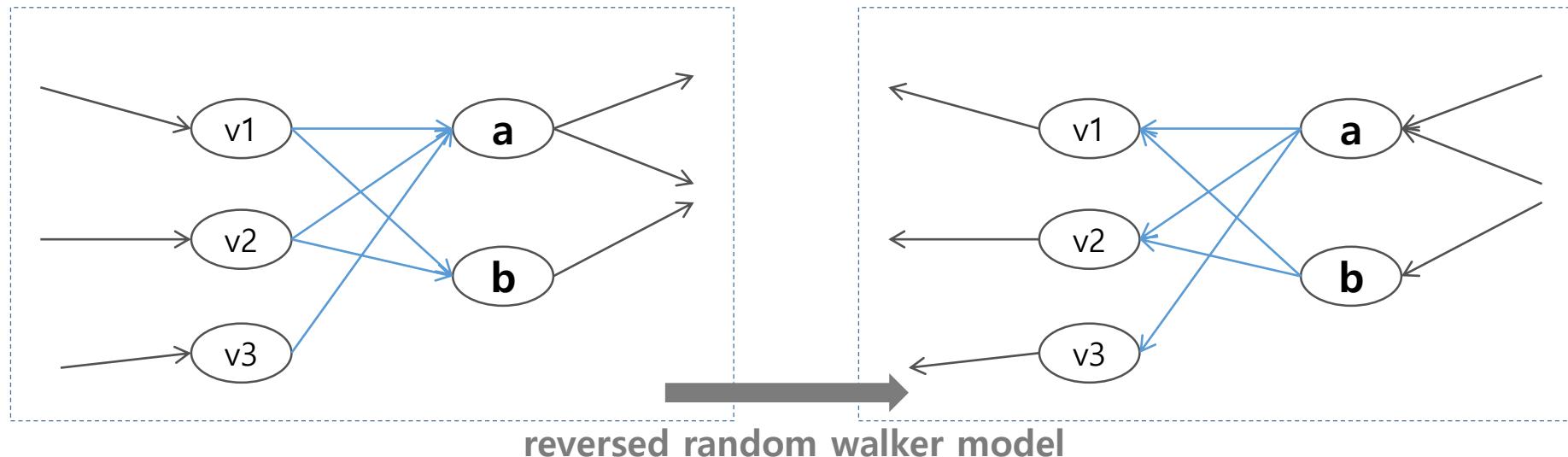
$$S_{k+1}(a, b) = \frac{C}{|I(a)||I(b)|} \sum_{i \in I(a)} \sum_{j \in I(b)} S_k(i, j)$$

$I(a)$ : 마디  $a$  의 inbounds  
 $I(b)$ : 마디  $b$  의 inbounds  
 $S_0(a, b) = 1$  if  $a = b$  else 0



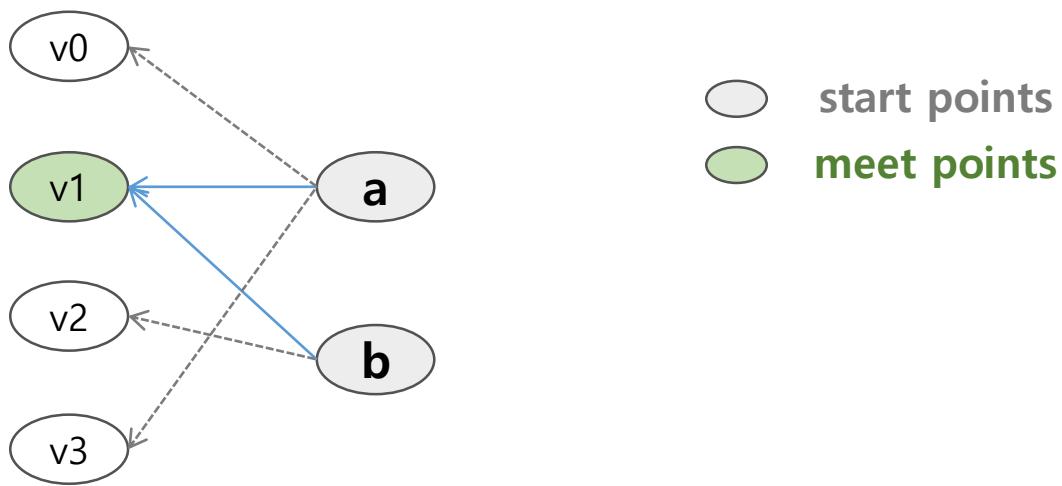
# SimRank 의 의미

- SimRank 는 reversed graph 에서의 random walker 로 해석합니다.
  - k 번의 iteration 을 통하여 학습된 SimRank,  $S_k(a, b)$  는 reversed graph 에서 ( $a, b$ ) 를 출발한 random walker 가 k step 안에 만날 확률입니다.



# SimRank 의 의미

$$Sim_{k+1}(a, b) = \frac{C|_{c=1}}{|I(a)||I(b)|} \sum_{i \in I(a)} \sum_{j \in I(b)} Sim_k(i, j)$$



## At 1 step iteration,

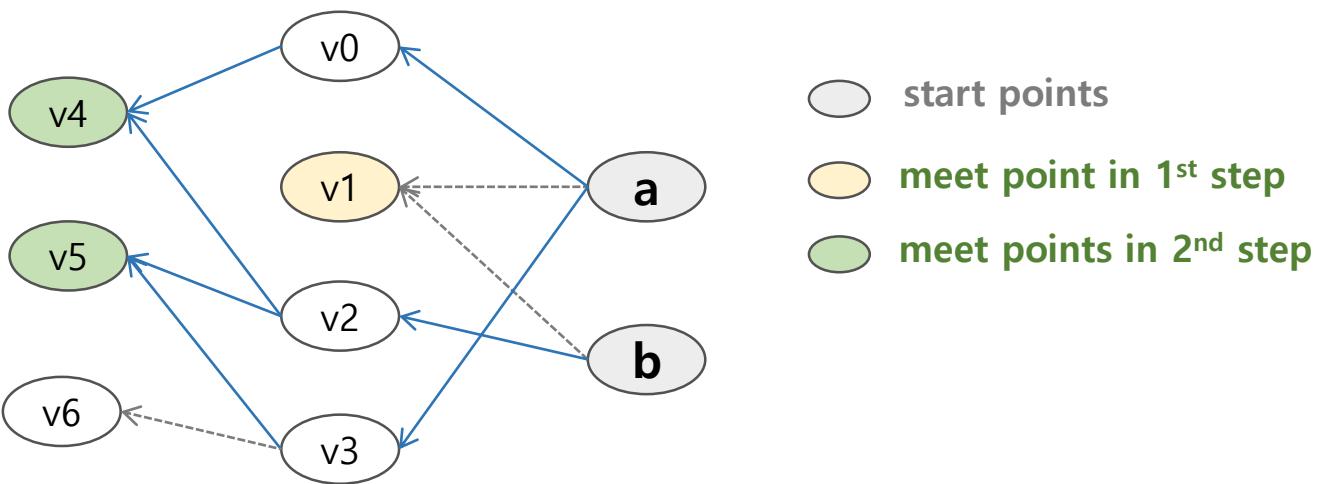
'a' has 3 paths, 'b' has 2 paths, (1 commons)

- Prob. of meeting at  $v_1 = \frac{1}{3} \times \frac{1}{2} = \frac{1}{6}$

- Prob. of meeting within 1 step =  $\frac{1}{6}$

# SimRank 의 의미

$$Sim_{k+1}(a, b) = \frac{C|_{c=1}}{|I(a)||I(b)|} \sum_{i \in I(a)} \sum_{j \in I(b)} Sim_k(i, j)$$



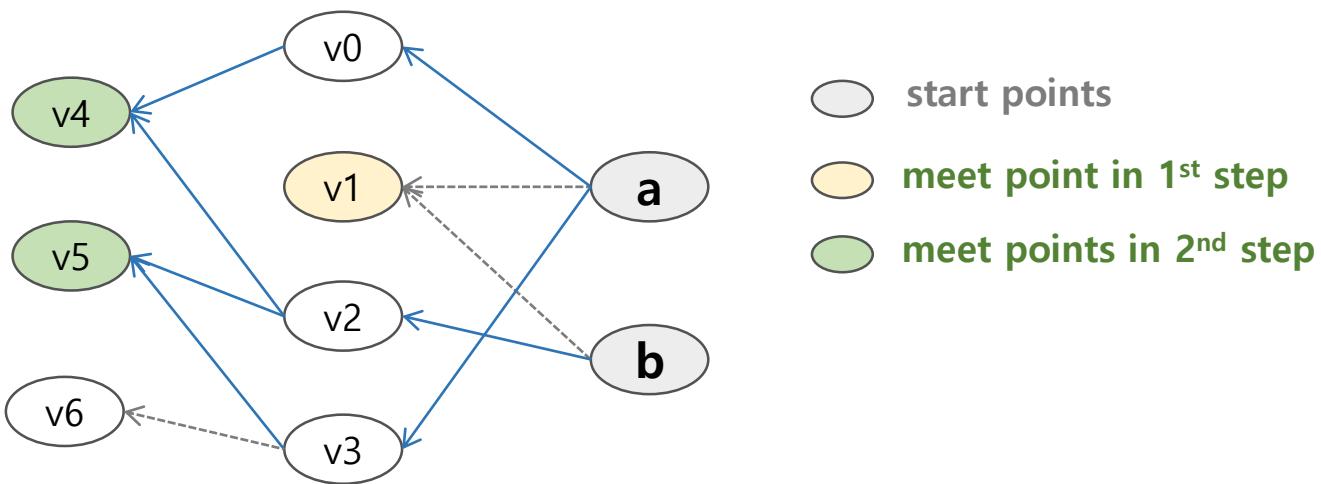
## At 2 step iteration,

- Prob. of meeting at v4 | {a - v0 - v4 // b - v2 - v4 } =  $\left(\frac{1}{3} \times 1\right) \times \left(\frac{1}{2} \times \frac{1}{2}\right) = \frac{1}{12}$
- Prob. of meeting at v5 | {a - v3 - v5 // b - v2 - v5 } =  $\left(\frac{1}{3} \times \frac{1}{2}\right) \times \left(\frac{1}{2} \times \frac{1}{2}\right) = \frac{1}{24}$
- Prob. of meeting at 2 steps =  $\frac{3}{24}$
- Prob. of meeting within 2 steps =  $\frac{1}{6} + \frac{3}{24} = \frac{7}{24}$

# SimRank 의 의미

- Survival rate  $C$  를 고려합니다.

$$Sim_{k+1}(a, b) = \frac{C|_{c=1}}{|I(a)||I(b)|} \sum_{i \in I(a)} \sum_{j \in I(b)} Sim_k(i, j)$$

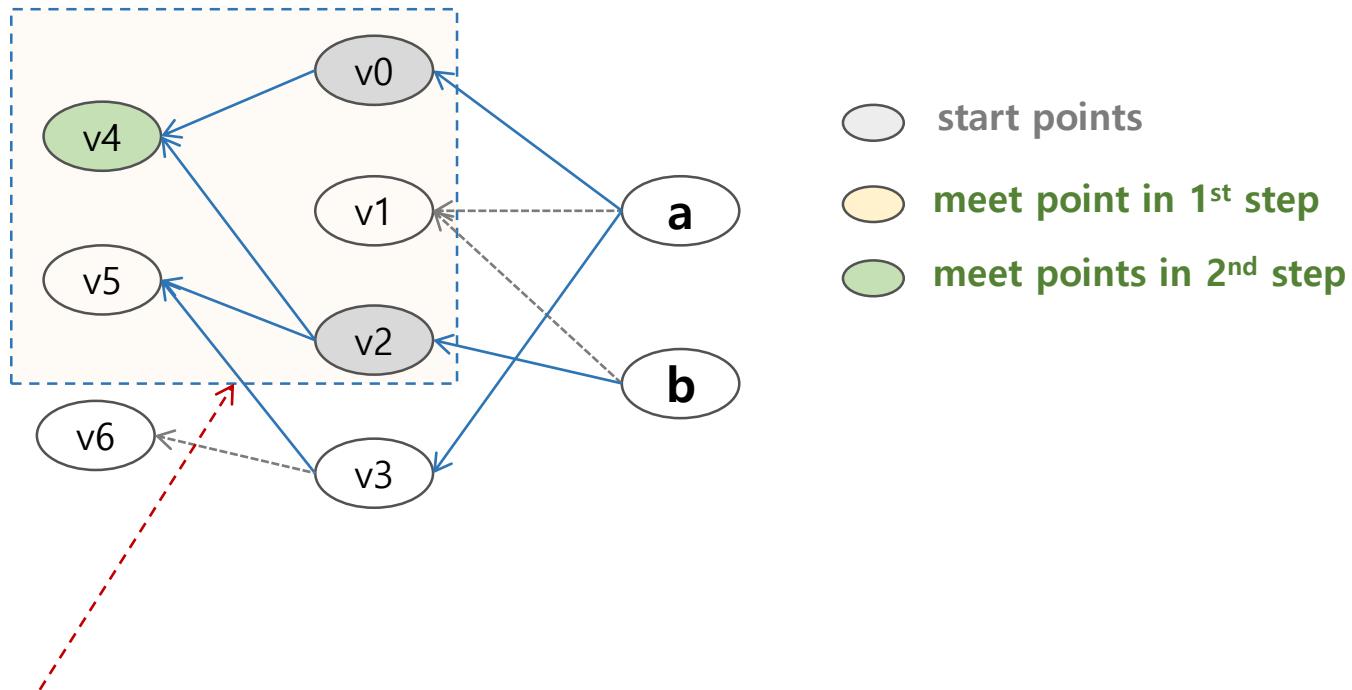


## At 2 step iteration with decaying factor

- Prob. of meeting within 2 steps =  $c \frac{1}{6} + c^2 \frac{3}{24}$

# SimRank 의 의미

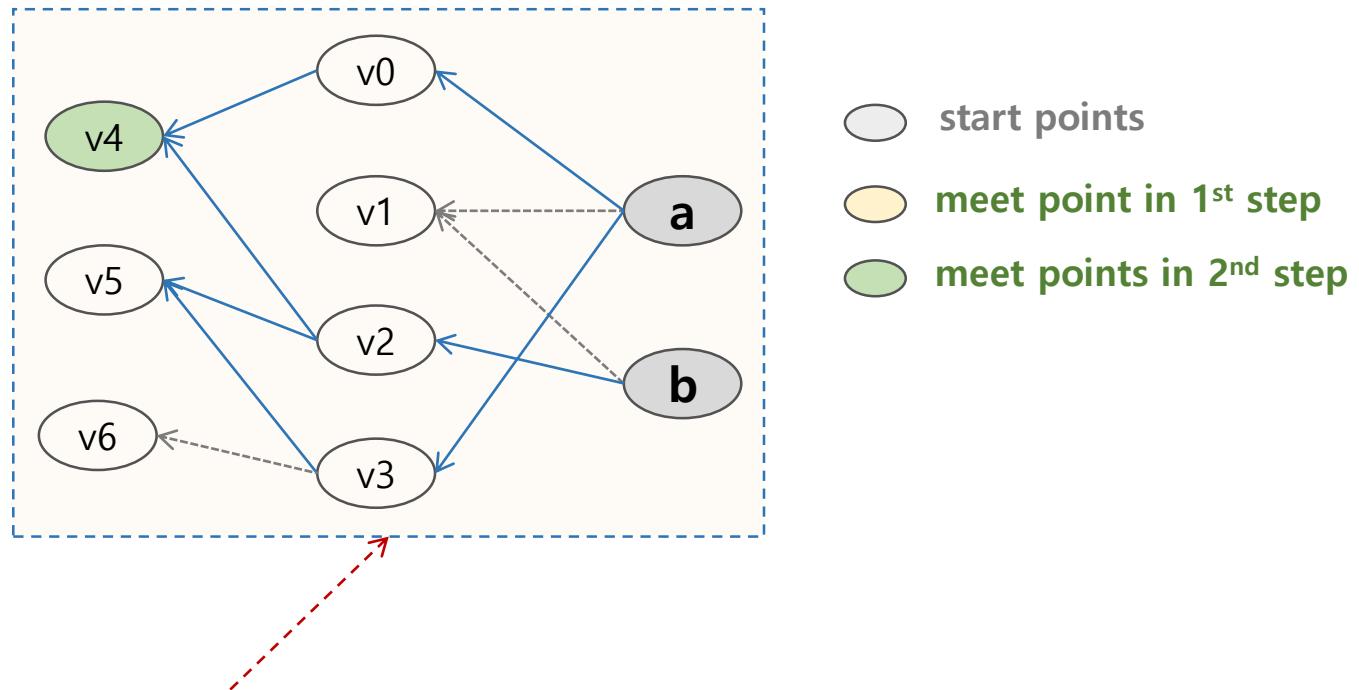
$$Sim_1(a, b) = \frac{C|_{c=1}}{|I(a)||I(b)|} \sum_{i \in I(a)} \sum_{j \in I(b)} Sim_0(i, j)$$



Iteration 1에서  $Sim_1(v0, v2)$  가 업데이트되지만,  
 $Sim_1(v0, v2)$  가  $Sim_1(a, b)$  까지 전달되지는 않습니다

# SimRank 의 의미

$$Sim_1(a, b) = \frac{C|_{c=1}}{|I(a)||I(b)|} \sum_{i \in I(a)} \sum_{j \in I(b)} Sim_0(i, j)$$



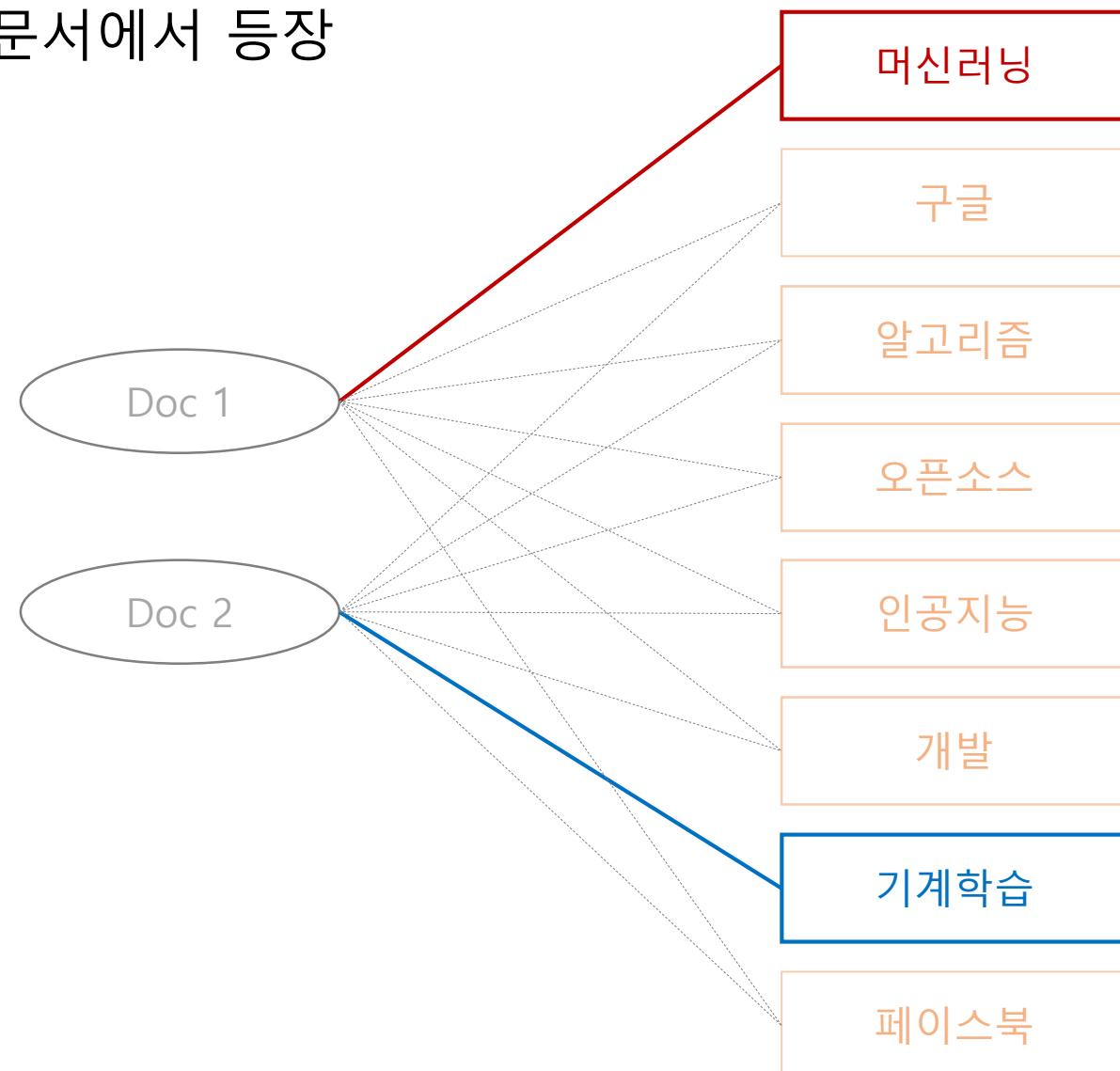
Iteration 2에서  $Sim_1(v_0, v_2)$  고려되어  $Sim_2(a, b)$ 에 업데이트 됩니다.

# SimRank 의 의미

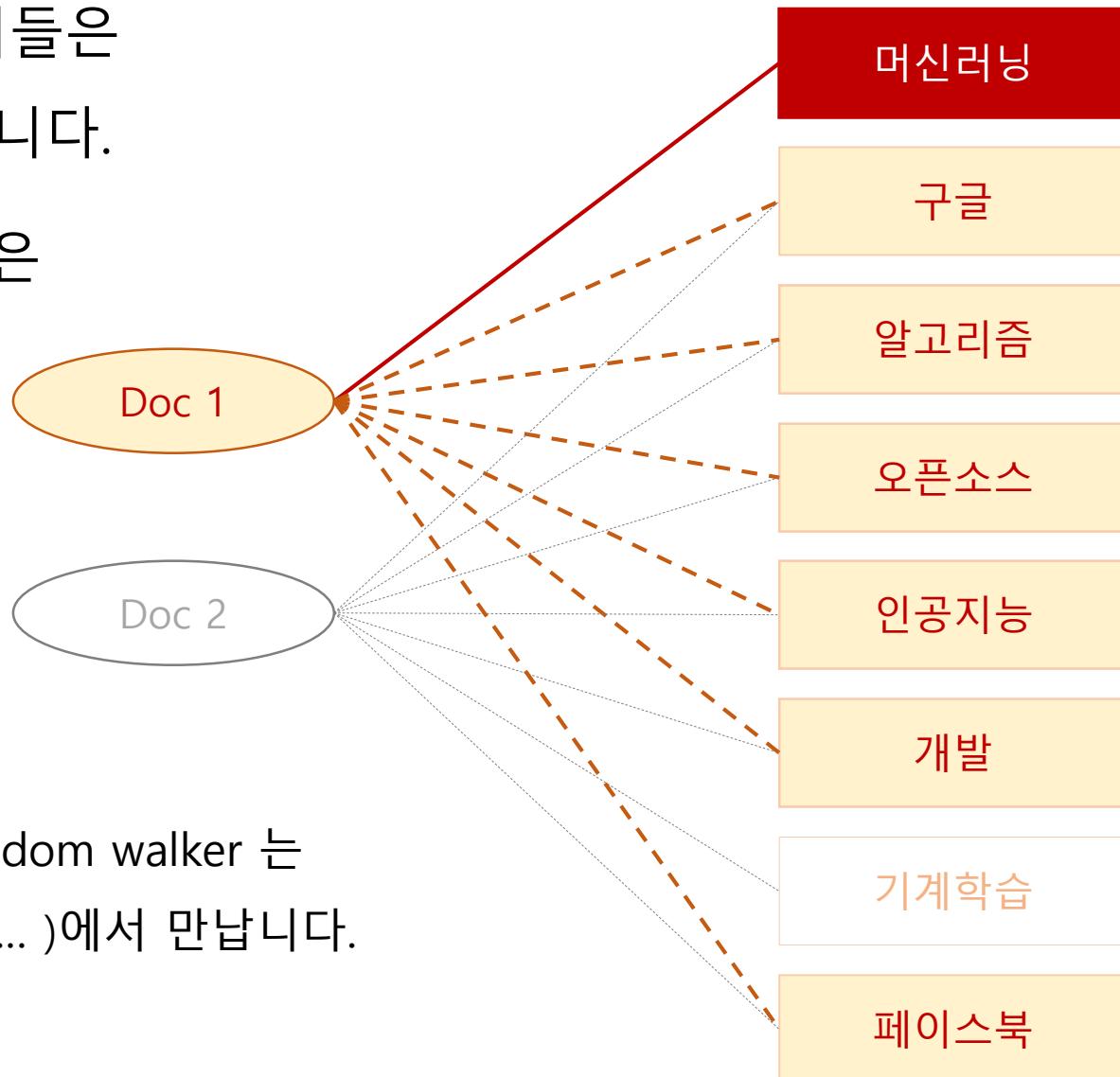
---

- Term – frequency (bipartite) graph 에서는 같은 문서에 등장하지 않은 단어더라도 유사한 토픽에서 등장하는 단어를 찾을 수 있습니다.

- (머신러닝, 기계학습) 은 같은 문서에서 등장한 적이 없습니다.

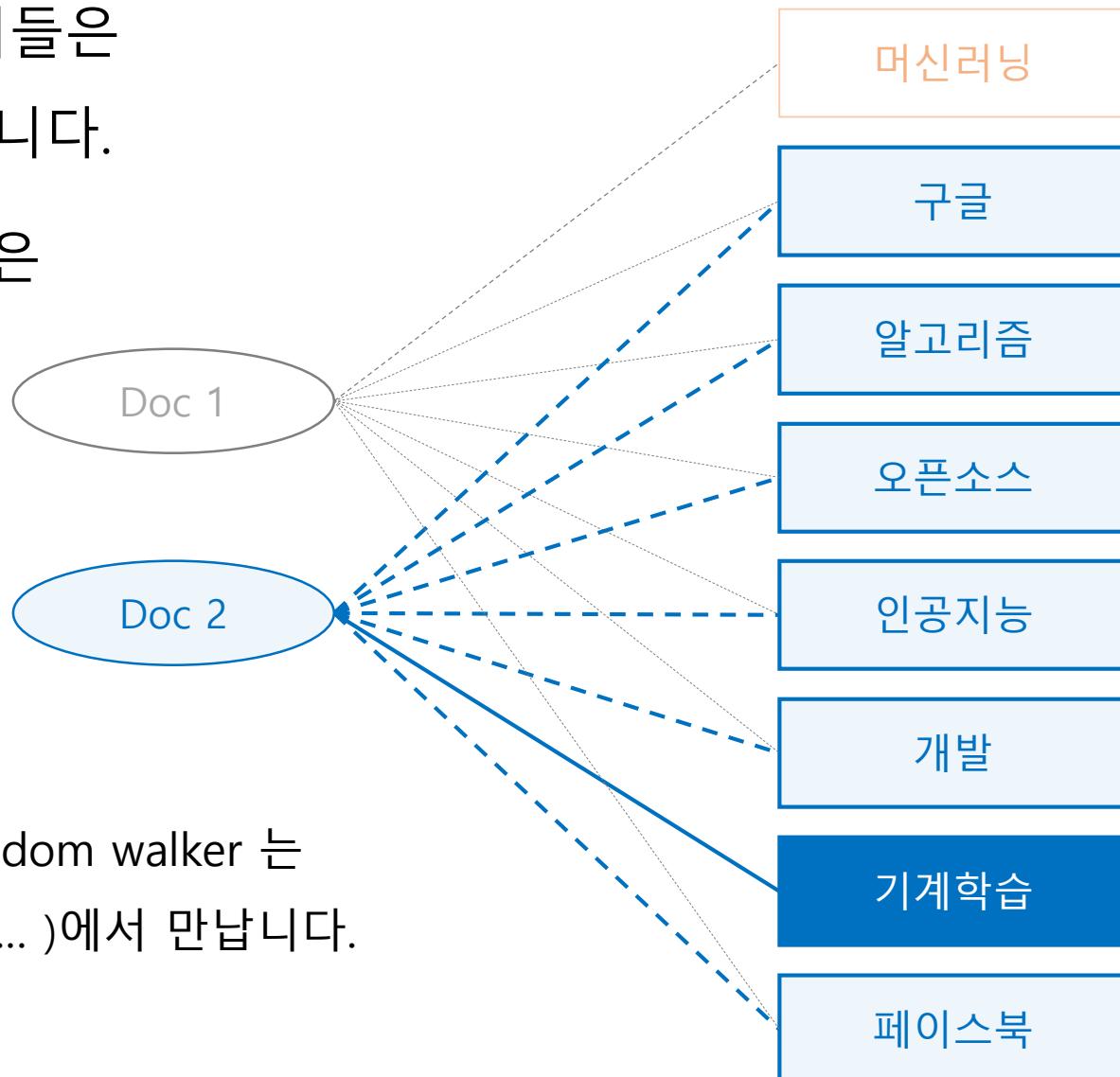


- “머신러닝”과 함께 등장한 단어들은 “기계학습”과도 함께 등장했습니다.
- “머신러닝”, “기계학습”의 이웃은 서로 비슷합니다.



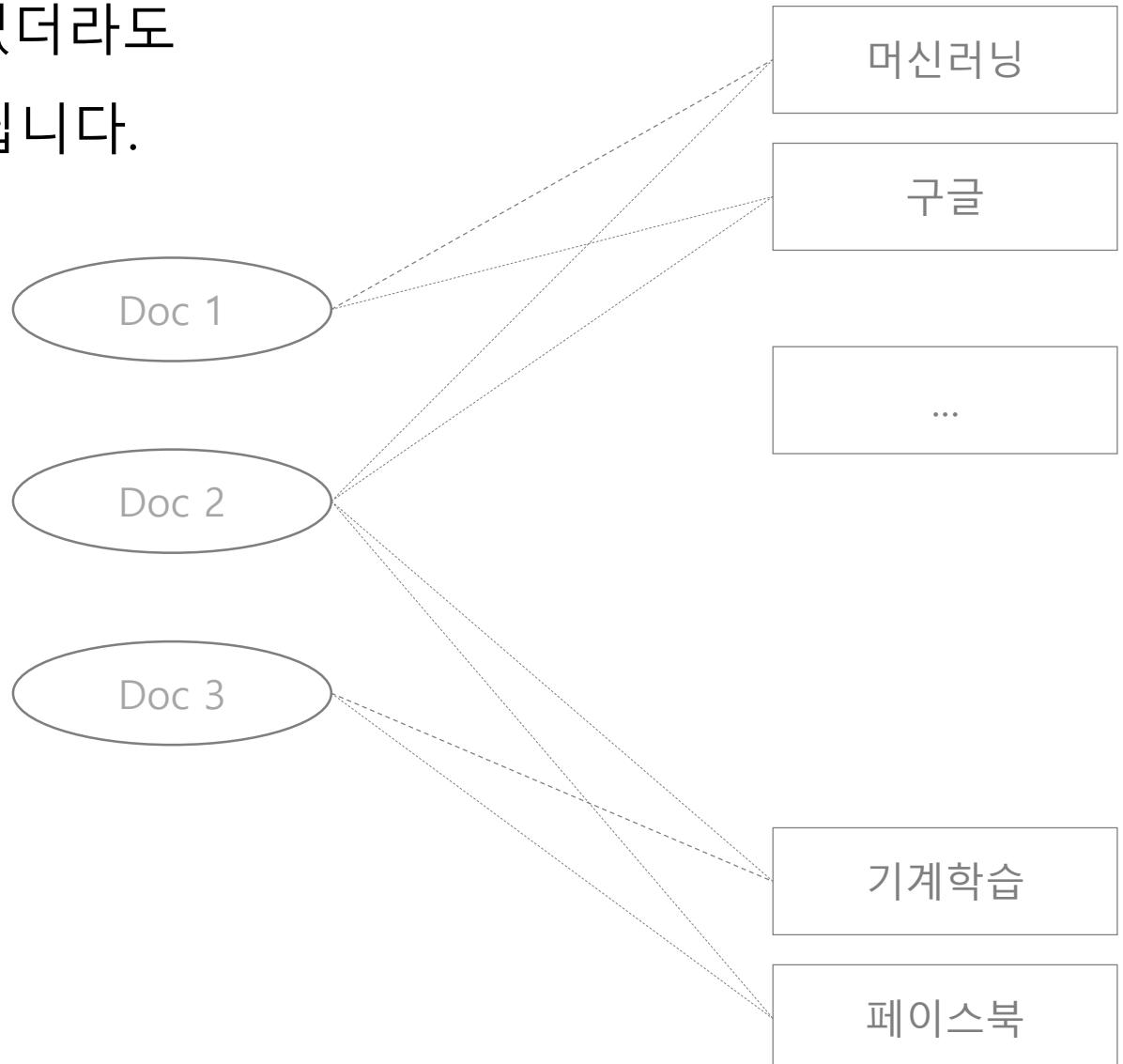
- (머신러닝, 기계학습)에서 출발한 random walker는 2 step 후 (구글, 알고리즘, 오픈소스, ...)에서 만납니다.

- “머신러닝”과 함께 등장한 단어들은 “기계학습”과도 함께 등장했습니다.
- “머신러닝”, “기계학습”의 이웃은 서로 비슷합니다.



- (머신러닝, 기계학습)에서 출발한 random walker 는 2 step 후 (구글, 알고리즘, 오픈소스, ...)에서 만납니다.

- 문서 1과 3은 공통된 단어가 없더라도 문서 2를 통하여 유사도를 지닙니다.



# SimRank

---

- Doc2Vec 은 단어 유사성을 바탕으로, 문서간 유사도를 정의합니다.
  - 공통된 단어가 없더라도 비슷한 단어들이 포함된 두 개의 문서의 document vector는 비슷합니다.
- SimRank 은 Doc2Vec 과 유사합니다.
  - 유사한 단어가 포함된 두 개의 문서는 공통된 단어가 없더라도 유사합니다.

# Small world effect

---

- Small world effect 는 Stanley Milgram 의 six degrees of separation 이론으로 설명됩니다.
  - 한 나라 안에서 모든 사람들은 여섯 단계를 걸치면 서로 아는 사이입니다.
  - 네트워크에서는 몇 단계만 거치면 similarity 가 존재합니다.

# Small world effect

- SimRank 의 similarity matrix 는 몇 번의 iteration 만으로도 sparsity 가 급격히 내려갑니다.
  - 영화 1,949개, 출연 배우 7,776명, 매우 sparse한 네트워크

# Iteration	# of visited actors	# of movie similarity > 0
1	7.158543	18.4274
2	18.4274	229.7912
3	112.9764	679.4854
<b>4</b>	<b>229.7912</b>	<b>1103.132</b>
5	1013.583	1513.811
6	679.4854	1635.561
7	2686.14	1662.276
8	1103.132	1668.141
9	4380.891	1669.592

# SimRank

---

- 모든 node pairs 간의 similarity 가 필요한 경우는 적습니다.
  - Lee et al., (2012) 에서는 한 query node 의 top k 개의 유사 마디만을 계산하는 방법을 제안하였습니다.
  - soygraph 에서 SingleVectorSimRank 로 구현되어 있습니다.

# SimRank

- SimRank 는 topically similar 한 단어들을 학습합니다.

```
from soygraph.similarity import SingleVectorSimRank
from soygraph import DictGraph

g = DictGraph(dd)
simrank = SingleVectorSimRank(g)
similar = simrank.most_similar(vocab2node['아이오아이'], max_iter=4, topk=30)
```

너무너무너무 박진영 빅브레인 완전체 신용재 오블리스

갓세븐 엠카운트다운 중독성 잠깐 세븐 다비치 상큼 소녀들

선의 산들 수록곡 프로듀스101 펜타곤 열창 타이틀곡 엠넷

본명 박소라 음원차트 깜찍 이진희 불독 키미 음악방송

# Random Walk with Restart (RWR)

---

- RWR<sup>[1,2]</sup> 은 근처에 있는 마디를 탐색합니다.
  - Random walker 의 모델을 이용하여 local nodes 를 탐색합니다.
  - $R_{k+1}(u | v) = c \cdot W \cdot R_k(u | v) + (1 - c)e_v$

