5장. 유사성과 모호성

2022-10-12

Contents

NLP Lab
Natural Language Processing

한림대학교
HALLYM UNIVERSITY

5.1 단어의 의미

- 단어의 의미 (Word Sense)
  - 겉으로 보이는 형태 내에 의미를 가짐
    - 하나의 형태에 여러 의미
      - 예) 'sow' – '씨를 뿌리다', '암퇘지'
      - 예) 'bear' - '견디다', '곰'
    - 다른 형태의 단어들이 서로 같은 의미를 공유
      - 'drink' vs 'beverage', 'buy' vs 'purchase'
  - 의미 유사성, 모호성 (Ambiguity)
    - 효율성을 추구하도록 진화해 왔기에 모호성이 발생
    - 컴퓨터가 인간의 언어를 이해하고 처리할 때 매우 큰 과제

5.1.1 단어와 의미의 관계

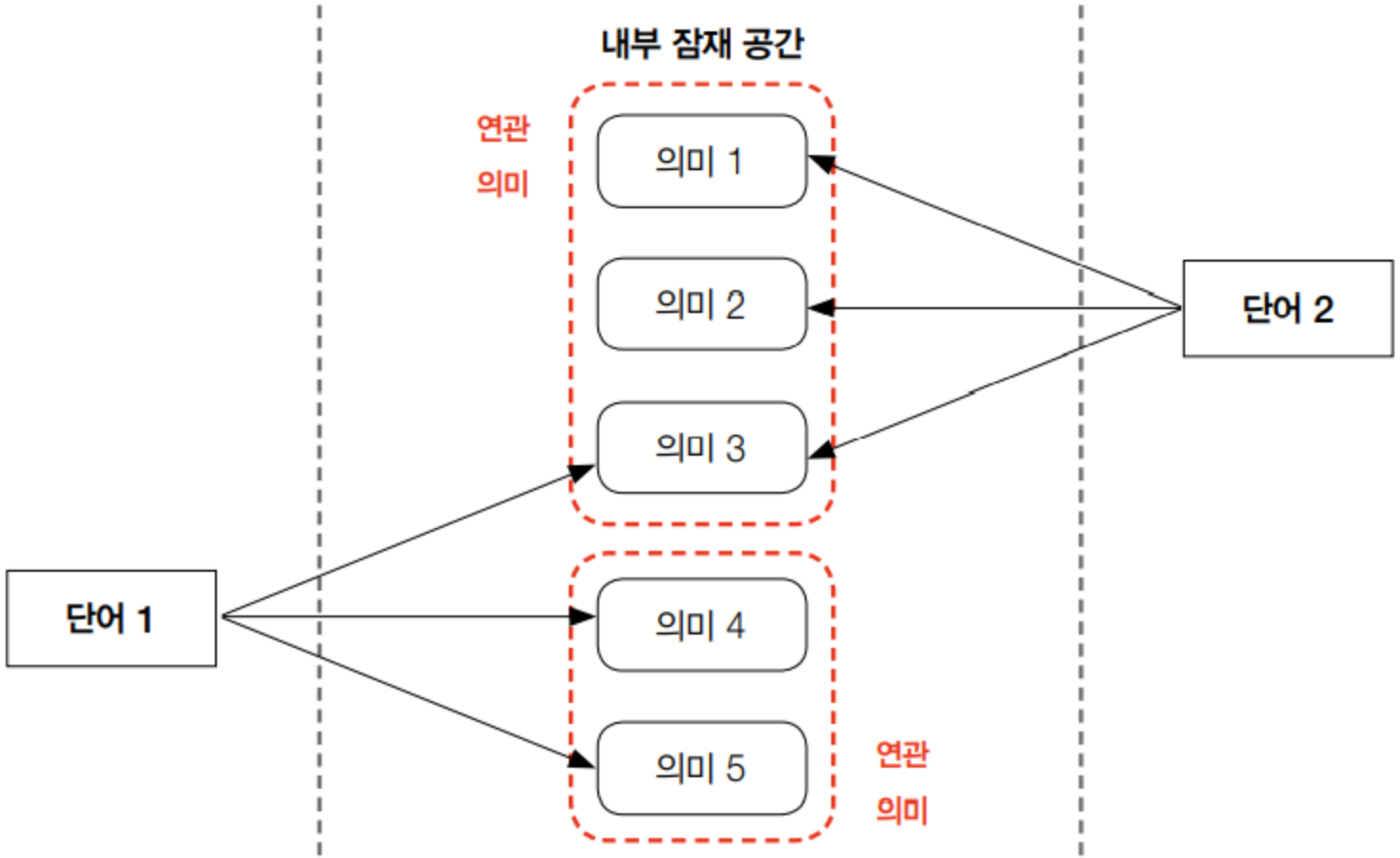
- 단어와 의미의 관계
  - 겉으로 보이는 형태 (표제어, Lemma) 내에 **여러 의미를 포함**함
  - 같은 형태의 단어도 상황에 따라 각각 다른 의미로 사용
  - 사람은 **주변 정보 (Context)**에 따라 **단어의 의미 파악**
  - 주변 정보가 부족한 경우, 모호성 증가
  - 다르게 해석한 경우, 잘못된 의미로 이해

연관 의미	세부 의미	풀이
차# 1	차1-1	좋은 향기나 맛이 있는 식물의 잎이나 뿌리, 열매 등을 달이거나 우려서 만든 마실 것.
차# 2	차2-1	바퀴가 달려 있어 사람이나 짐을 실어 나르는 기관.
	차2-2	사람이나 물건을 차에 실어 그 분량을 세는 단위.
	차2-3	장기의 말 중에서 '車' 자를 새긴 말.
차# 3	차3-1	둘 이상을 비교했을 때 서로 다르게 나타나는 수준이나 정도.
	차3-2	어떤 수나 식에서 다른 수나 식을 뺀 나머지.
차# 4	차4-1	어떤 일의 차례나 횟수를 나타내는 말.
	차4-2	어떠한 일을 하던 기회나 순간.
	차4-3	일정한 주기나 기간이 지난 해당 시기를 나타내는 말.

▶ 단어 '차'의 다양한 세부 의미와 풀이 비교

모호성 발생

5.1.1 단어와 의미의 관계



▶ 단어의 형태들과 내부 의미들이 갖는 관계

5.1.1 단어와 의미의 관계

- 형태는 공유하고 서로 다른 의미로 사용
  - 다의어 (polysemy) : 서로 비슷한 의미로 사용 (세부 의미가 다름)
  - 동형어 (homonym) : 전혀 다른 의미로 사용 (연관 의미가 다름)

- 어떤 의미는 다른 형태의 단어에서도 사용
  - 동의어 (synonym)

- 사람이 머릿속에서 생각하는 의미를 실제 단어를 사용하여 전달
  - 의미 : 심층부
  - 단어 : 표층부

- 어의 중의성 (word sense ambiguation)
  - 기계번역에서 단어의 의미에 따라서 해당 번역 단어의 형태가 결정됨
    - 연관 의미는 물론이고 세부 의미도 반영되어야 함
  - 자연어처리에서는 '단어'라는 형태로 '의미'를 파악하는 문제가 중요함
    - 어의 중의성 해소 (Word Sense Disambiguation)

5.1.2 동형어와 다의어

- 모호성을 발생시키는 어휘
  - **동형어 (Homonym)** : 형태는 같으나 뜻이 서로 다른 단어 (연관의미가 다름)
    - 어원이 다름
  - **다의어 (Polysemy)** : 하나의 형태로 여러 의미를 지니고, 그 의미들이 서로 관련성을 가지는 단어 (세부의미가 다름)

종류	단어 형태	의미1	의미2
다의어	다리	사람 다리(脚. leg)	책상 다리(跏. desk leg)
동형어	차	마시는 차(茶. tea)	달리는 차(車. car)

▶ 다의어와 동형어 비교

- 어의 중의성 해소 (Word Sense Disambiguation)
  - Classical NLP
    - 서브 모듈들의 처리 과정 중에 단어 중의성 문제를 처리
    - 주변 문맥을 통해 원래 단어의 의미를 파악
  - Deep NLP
    - 단어 중의성 해소에 대한 필요도가 낮아짐
    - End-to-end 및 RNN 구조 (자체적으로 문맥이 반영됨)

5.1.3 동의어

- **동의어 (Synonym)** : 다른 형태이지만 의미가 같은 단어
  - 동의어 집합 (Synset)이 존재

형태	의미	동의어 집합
home	home# 1	place# 7
	home# 2	dwelling# 1, domicile# 2, abode# 2, habitation# 2, ...
	home# 3	
	home# 4	home plate# 1, home base# 1, plate# 1
	home# 5	base# 14
	...	
place	place# 7	home# 1
	place# 8	position# 6, post# 3, berth# 1, office# 7, spot# 8, billet# 3, place# 8
	...	

▶ 단어의 의미별 동의어 집합 사례

5.1.4 상위어와 하위어

- **상위어와 하위어**
  - **상위어 (Hypernym)** : 상위 개념을 표현하는 단어
  - **하위어 (Hyponym)** : 하위 개념을 표현하는 단어

상위어	하위어
동물	포유류
포유류	코끼리
코끼리	아프리카코끼리
사물	전화기
전화기	핸드폰
사물	컴퓨터
컴퓨터	노트북

▶ 상위어와 하위어 사례

  - 단어 간 관계 구조를 계층화
    - 단어간의 유사도 계산, 부족한 정보를 비슷한 관계의 다른 단어에서 획득 가능

5.1.5 모호성 해소

- 모호성 제거를 위한 처리 방법
  - **단어 중의성 해소 (Word-Sense Disambiguation, WSD)**

- 모호성의 발생 이유
  - 언어는 겉은 불연속적인 심볼이지만, 내부적으로는 의미를 가지고 있음
  - 여러 의미가 하나의 형태를 공유
    - 의미들이 매우 다르기도 하고 비슷하기도 함

- 단어 중의성 해소
  - 텍스트가 내포함 진짜 의미를 파악하는 과정
  - 모호성이 높은 텍스트엣 모호성을 제거함으로써 진짜 의미를 파악

5.2 원핫 인코딩

- 단어의 의미
  - 단어는 특정 개념을 표현
  - 어휘 분류 체계가 있음
  - 의미 간의 유사도가 있음
  - 매우 비슷한 의미, 반대의 의미, 상관없는 관계 등이 존재
  - 불연속적인 심볼
    - 내부의 의미는 유사하나 겉 형태는 다를 수 있음 (동형어 제외)

- 머신 러닝 적용
  - "고양이는 좋은 반려동물입니다."
  - "강아지는 훌륭한 애완동물입니다."
    - "고양이 " vs "강아지", "좋은 " vs "훌륭한", "반려동물 " vs "애완동물"
    - 유사한 단어들로부터 부족한 정보를 보완
  - 불연속적인 단어 간 유사도를 효과적으로 구하는 과정 필요
  - 워드 임베딩

5.2. 원핫 인코딩

- 원핫 인코딩 (One-Hot Encoding)
  - 단어를 컴퓨터가 인지할 수 있는 **벡터로 변환**하는 가장 간단한 방법
  - 하나의 1과 나머지 수많은 0들로 표현
  - 원핫 인코딩 벡터의 차원 : 전체 어휘의 개수 (Number of Vocabulary)

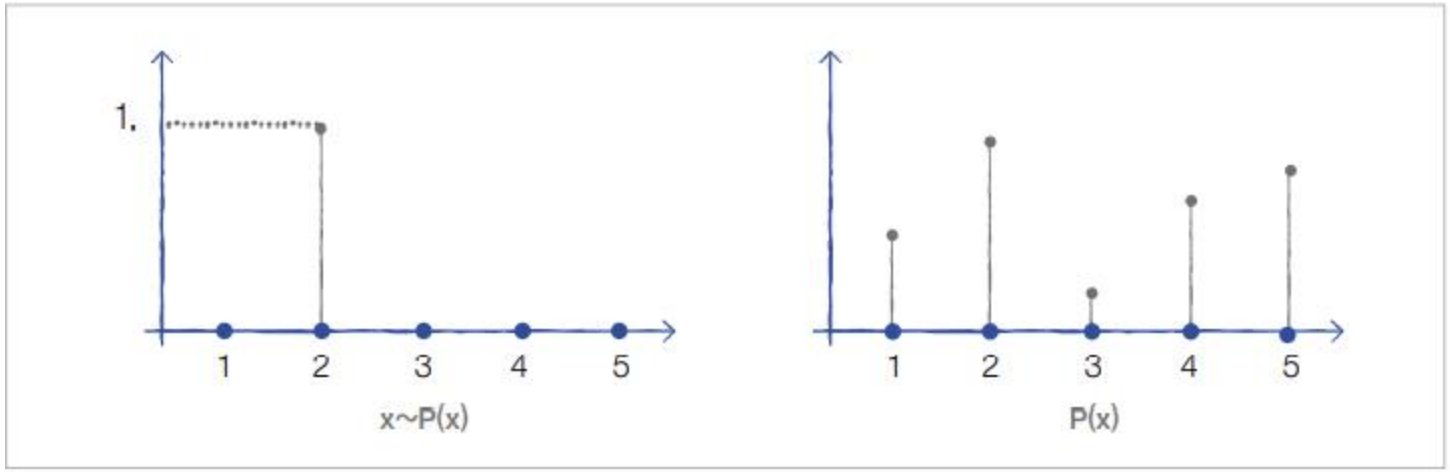
$$v \in \{0, 1\}^{|V|}, \text{ where } v \text{ is one-hot vector and } |V| \text{ is vocabulary size.}$$

단어	사전내 순서(index)	원핫 벡터
…		
강아지	8	0,0,0,0,0,0,0,1,0,0,0,…,0
개	9	0,0,0,0,0,0,0,0,1,0,0,…,0
고양이	10	0,0,0,0,0,0,0,0,0,1,0,…,0
구렁이	11	0,0,0,0,0,0,0,0,0,0,1,…,0
…		
하마	20,567	0,…,0,0,0,0,0,0,0,0,0,0,1
…		

▶ 단어별 원핫 벡터 형태

5.2 원핫 인코딩

- 원핫 벡터
  - 이산 확률 분포로부터 뽑아낸 샘플
  - 멀티눌리 확률 분포



▶ 이산 확률 분포로부터 샘플링하여 얻어지는 원핫 벡터

5.2. 원핫 인코딩

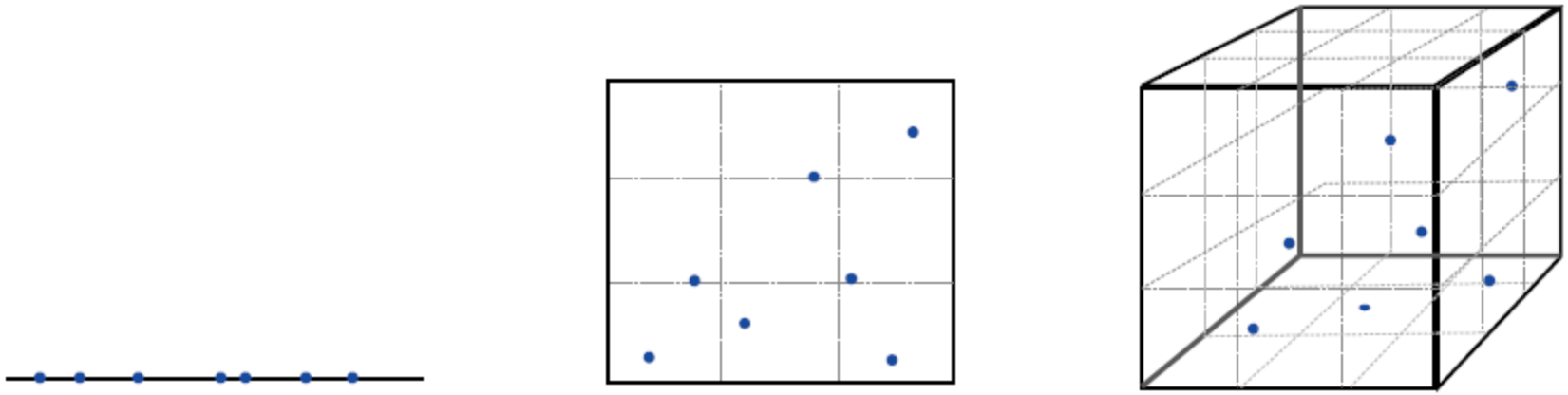
- 문제점
  - 원핫 벡터의 차원은 매우 크다.
  - 단 하나의 1과 나머지는 모두 0 (**희소 벡터, Sparse Vector**)
  - 벡터 간 연산 시 결과 값이 0이 된다. → 서로 **직교**하는 경우가 많아진다.

$$[0,0,\cdots,1,0]\times[0,1,0,\cdots,0]^T=0$$

  - "강아지" vs "개"
    - 유사한 단어이나 유사도는 0이 나옴
  - "강아지 " vs "컴퓨터"
    - 관계가 적어도 유사도는 0
  - "강아지 " 의 특징을 "개 " 로부터 받을 수 없음

5.2.1 차원의 저주

- 차원의 저주
  - 차원이 증가할 수록 벡터가 매우 낮은 밀도로 희소하게 분포하게 된다.
  - 지수 함수 (Exponential)와 같이 증가하게 된다.
  - 이를 해결하기 위해, 차원을 축소하여 단어를 표현할 **필요성 증가 → 현재 워드 임베딩**



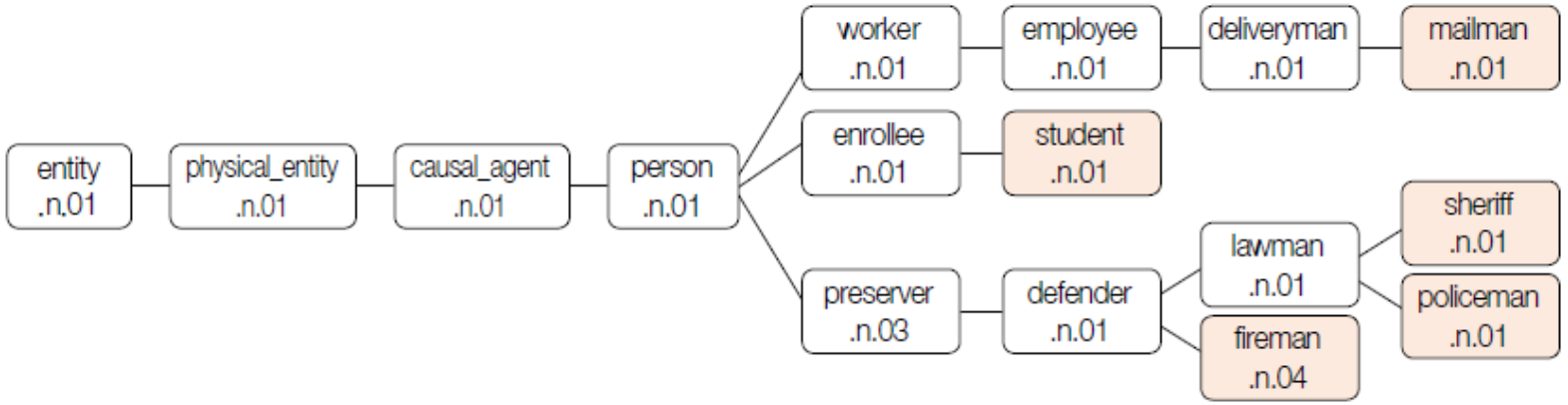
▶ 차원의 저주: 차원이 높을수록 같은 정보를 표현하는 데 불리합니다.

5.3. 시소러스를 활용한 단어 의미 파악

- 시소러스 (Thesaurus)
  - 단어의 의미는 개념과 같아서 계층적 구조를 가짐
  - 원핫 벡터로는 이러한 계층 구조를 표현할 수 없음
  - 계층 구조를 잘 분석하고 분류하여 데이터베이스로 구축 => 시소러스
  - 어휘 분류 사전
  - 대표적인 예시 : 워드넷 (WordNet)

5.3.1 워드넷

- 워드넷 (WordNet)
  - 1985년 프린스턴 대학교의 조지 아미티지 밀러 교수가 개발
  - 초기 목적 : 기계번역 문제 해결
  - 동의어 집합, 상위어, 하위어 등의 정보를 가진 데이터베이스
  - 유향 비순환 그래프 (Directed Acyclic Graph)로 구성됨
  - 트리구조가 아님
    - 하나의 노드가 여러 상위 노드를 가질 수 있음



▶ 워드넷 내의 단어별 top-1 의미의 top-1 상위어만 선택하여 트리 구조로 나타낸 경우

5.3. 시소러스를 활용한 단어 의미 파악

WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: bank Search WordNet

Display Options: (Select option to change) Change
Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"
Display options for word: word#sense number

Noun

- S: (n) **bank#1** (sloping land (especially the slope beside a body of water)) *"they pulled the canoe up on the bank"; "he sat on the bank of the river and watched the currents"*
- S: (n) depository financial institution#1, **bank#2**, banking concern#1, banking company#1 (a financial institution that accepts deposits and channels the money into lending activities) *"he cashed a check at the bank"; "that bank holds the mortgage on my home"*
- S: (n) **bank#3** (a long ridge or pile) *"a huge bank of earth"*
- S: (n) **bank#4** (an arrangement of similar objects in a row or in tiers) *"he operated a bank of switches"*
- S: (n) **bank#5** (a supply or stock held in reserve for future use (especially in emergencies))
- S: (n) **bank#6** (the funds held by a gambling house or the dealer in some gambling games) *"he tried to break the bank at Monte Carlo"*
- S: (n) **bank#7**, cant#2, camber#2 (a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force)
- S: (n) savings bank#2, coin bank#1, money box#1, **bank#8** (a container (usually with a slot in the top) for keeping money at home) *"the coin bank was empty"*
- S: (n) **bank#9**, bank building#1 (a building in which the business of banking transacted) *"the bank is on the corner of Nassau and Witherspoon"*
- S: (n) **bank#10** (a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)) *"the plane went into a steep bank"*

Verb

- S: (v) **bank#1** (tip laterally) *"the pilot had to bank the aircraft"*
- S: (v) **bank#2** (enclose with a bank) *"bank roads"*
- S: (v) **bank#3** (do business with a bank or keep an account at a bank) *"Where do you bank in this town?"*
- S: (v) **bank#4** (act as the banker in a game or in gambling)
- S: (v) **bank#5** (be in the banking business)
- S: (v) deposit#2, **bank#6** (put into a bank account) *"She deposits her paycheck every month"*
- S: (v) **bank#7** (cover with ashes so to control the rate of burning) *"bank a fire"*
- S: (v) count#8, bet#3, depend#2, swear#5, rely#1, **bank#8**, look#10, calculate#5, reckon#5 (have faith or confidence in) *"you can count on me to help you any time"; "Look to your friends for support"; "You can bet on that!"; "Depend on your family in times of crisis"*

이름	기관	웹사이트
KorLex	부산대학교	http://korlex.pusan.ac.kr/
Korean WordNet(KWN)	KAIST	http://wordnet.kaist.ac.kr/

▶ 한국어 워드넷 소개

동의어 집합 ('depository financial institution#1
banking concern#1
banking company#1)

5.3. 시소러스를 활용한 단어 의미 파악

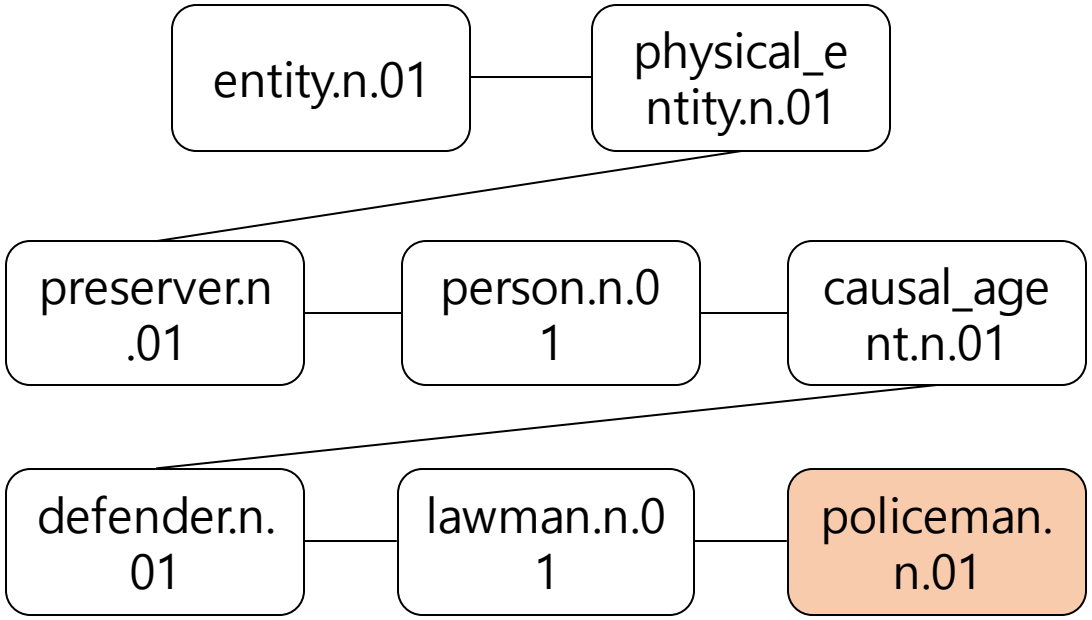

```
## 5.3.3. 워드넷을 활용한 단어간 유사도 비교.
from nltk.corpus import wordnet as wn

def hypernyms(word):
    current_node = wn.synsets(word)[0]
    print(current_node)

    while True:
        try:
            current_node = current_node.hypernyms()[0]
            print(current_node)
        except IndexError:
            break

hypernyms('policeman')
```

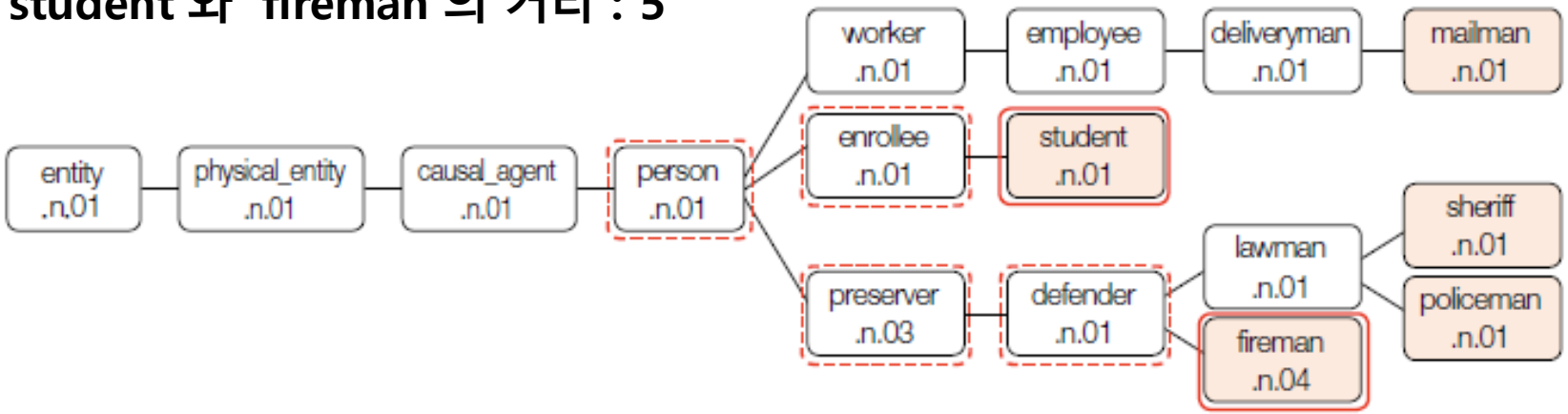
```
Synset('policeman.n.01')
Synset('lawman.n.01')
Synset('defender.n.01')
Synset('preserver.n.03')
Synset('person.n.01')
Synset('causal_agent.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```



쿼리 결과의 구조도

5.3. 시소러스를 활용한 단어 의미 파악

'student'와 'fireman'의 거리 : 5



▶ 'student'와 'fireman' 사이에 위치한 노드들

$$\text{similarity}(w, w') = -\log \text{distance}(w, w')$$

5.4. 특징

- 효과적인 표현 방법은?
  - 어떤 대상의 특징 (Feature)를 **잘 표현**해야 한다.
    - 특징은 수치로 표현
    - 최대한 많은 샘플을 설명
    - 각 샘플의 수치가 **서로 다르다**.
    - 최대한 **다양하게 표현**되어야 한다.
- MNIST 예제
  - 굵기, 기울기 등이 특징
- 특징 벡터 (feature vector)
  - 특징별 수치들을 모아 벡터로 표현할 것

▶ 각 특징에 따른 MNIST 예제

5.4.1 단어의 특징 벡터 구성을 위한 가정

- 단어의 특징 벡터 구성을 위한 가정
  1) **의미가 비슷한 단어라면 쓰임새가 비슷할 것**
  2) **쓰임새가 비슷하므로, 비슷한 문장 안에서 비슷한 역할로 사용될 것**
  3) **따라서 함께 나타나는 단어들이 유사할 것**

5.5. 특징 추출하기 : TF-IDF

- TF-IDF (Term Frequency-Inverse Document Frequency)

$$\mathrm{TF-IDF}(w,d) = \frac{\mathrm{TF}(w,d)}{\mathrm{DF}(w)}$$

  - 단어 w가 문서 d 에서 얼마나 중요한지를 나타내는 수치
    - 이 수치가 높을수록 w는 d를 대표하는 성질을 띠게 됨
  - TF (Term Frequency) : 단어의 문서 내에 출현한 횟수
    - 숫자가 클수록 문서에서 중요한 단어
  - IDF (Inverse Document Frequency) : 그 단어가 출현한 문서의 숫자의 역수
    - 어디서나 흔하게 나타나는 단어들에게 패널티
  - 다른 문서들에서는 잘 나타나지 않지만 특정 문서에서만 잘 나타난 경우를 반영
    - 특정 문서에서 얼마나 중요한 역할을 차지하는지 나타냄

5.4. 특징 – TF-IDF 구현

[14]:
```
## TF-IDF 구현하기.
import pandas as pd
from operator import itemgetter
import numpy as np

def get_term_frequency(document, word_dict=None):
    if word_dict is None:
        word_dict = dict()
    words = document.split()

    for w in words:
        word_dict[w] = 1 + (0 if word_dict.get(w) is None else word_dict[w])
    return pd.Series(word_dict).sort_values(ascending=False)

def get_document_frequency(documents):
    dicts = list()
    vocab = set(list())
    df = dict()

    for d in documents:
        tf = get_term_frequency(d)
        dicts += [tf]
        vocab = vocab | set(tf.keys())

    for v in list(vocab):
        df[v] = 0
        for dict_d in dicts:
            if dict_d.get(v) is not None:
                df[v] += 1
    return pd.Series(df).sort_values(ascending=False)

docs = ['i go to school', 'do you want a go school', 'i will be go home']

def get_tfidf(docs):
    vocab = dict()
    tfs = list()

    for d in docs:
        vocab = get_term_frequency(d, vocab)
        tfs += [get_term_frequency(d)]
    df = get_document_frequency(docs)

    stats = list()
    for word, freq in vocab.items():
        tfidfs = list()
        for idx in range(len(docs)):
            if tfs[idx].get(word) is not None:
                tfidfs += [tfs[idx][word] * np.log(len(docs) / df[word])]
            else:
                tfidfs += [0]
        stats.append((word, freq, *tfidfs, max(tfidfs)))
    return pd.DataFrame(stats, columns=('word', 'freq', 'doc1', 'doc2', 'doc3', 'max')).sort_values('max', ascending=False)

get_tfidf(docs)
```

[14]:

	word	freq	doc1	doc2	doc3	max
3	home	1	0.000000	0.000000	1.098612	1.098612
4	be	1	0.000000	0.000000	1.098612	1.098612
5	will	1	0.000000	0.000000	1.098612	1.098612
6	to	1	1.098612	0.000000	0.000000	1.098612
7	do	1	0.000000	1.098612	0.000000	1.098612
8	you	1	0.000000	1.098612	0.000000	1.098612
9	want	1	0.000000	1.098612	0.000000	1.098612
10	a	1	0.000000	1.098612	0.000000	1.098612
1	i	2	0.405465	0.000000	0.405465	0.405465
2	school	2	0.405465	0.405465	0.000000	0.405465
0	go	3	0.000000	0.000000	0.000000	0.000000

5.6. 특징 벡터 만들기

- TF 행렬 만들기
  - **TF** 또한 **훌륭한 특징**이 될 수 있다.

[16]:
```
## TF 행렬 구현하기.
def get_tf(docs):
    vocab = dict()
    tfs = list()

    for d in docs:
        vocab = get_term_frequency(d, vocab)
        tfs += [get_term_frequency(d)]

    stats = list()
    for word, freq in vocab.items():
        tf_v = list()
        for idx in range(len(docs)):
            if tfs[idx].get(word) is not None:
                tf_v += [tfs[idx][word]]
            else:
                tf_v += [0]
        stats.append((word, freq, *tf_v))
    return pd.DataFrame(stats, columns=('word', 'freq', 'doc1', 'doc2', 'doc3')).sort_values('freq', ascending=False)

get_tf(docs)
```

[16]:

	word	freq	doc1	doc2	doc3
0	go	3	1	1	1
1	i	2	1	0	1
2	school	2	1	1	0
3	home	1	0	0	1
4	be	1	0	0	1
5	will	1	0	0	1
6	to	1	1	0	0
7	do	1	0	1	0
8	you	1	0	1	0
9	want	1	0	1	0
10	a	1	0	1	0

※ 문서가 지나치게 많으면, 벡터의 차원도 지나치게 커질 수 있다.

5.6. 특징 벡터 만들기

- **컨텍스트 윈도우**로 함께 출현한 단어들의 정보 활용하기
  - **동시발생 (Co-occurrence) 단어들**을 활용하는 방법
  - 윈도잉 (Windowing) 수행
    - 윈도우를 움직이며 **그 안에 있는 유닛들의 정보**를 **취합**하는 방법
  - **문서 내 단어 출현 횟수 (TF)** 보다 더 정확함

[30]:

	home	go	will	want	to	you	a	be	do	school	i
home	0	1	0	0	0	0	0	1	0	0	0
go	1	0	1	1	0	0	1	1	0	1	0
will	0	0	0	0	0	0	0	0	0	0	0
want	0	0	0	0	0	1	1	0	1	0	0
to	0	1	0	0	0	0	0	0	0	1	1
you	0	0	0	0	0	0	0	0	0	0	0
a	0	1	0	1	0	1	0	0	0	0	0
be	0	1	1	0	0	0	0	0	0	0	1
do	0	0	0	0	0	0	0	0	0	0	0
school	0	2	0	0	1	0	1	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0

[30]:

```
## Context Windowing.
from collections import defaultdict
import pandas as pd

def get_context_counts(lines, w_size=2):
    co_dict = defaultdict(int)

    for line in lines:
        words = line.split()
        for i, w in enumerate(words):
            for c in words[i-w_size:i+w_size]:
                if w != c:
                    co_dict[(w, c)] += 1
    return pd.Series(co_dict)

def co_occurrence(co_dict, vocab):
    data = list()

    for word_1 in vocab:
        row = list()
        for word_2 in vocab:
            try:
                count = co_dict[(word_1, word_2)]
            except KeyError:
                count = 0
            row.append(count)
        data.append(row)
    return pd.DataFrame(data, index=vocab, columns=vocab)

get_context_counts(docs)
vocab = list(set([i for d in docs for i in d.split()]))
co_occurrence(co_dict, vocab)
```

5.7. 벡터 유사도 구하기

- L1 거리 (파랑색 선)
  - L1 Norm을 사용한 것으로, "맨하튼 거리 (Manhattan Distance)"라고도 부른다.
  - 두 벡터의 각 차원별 값의 차이에 대한 절대값을 모두 합한 값

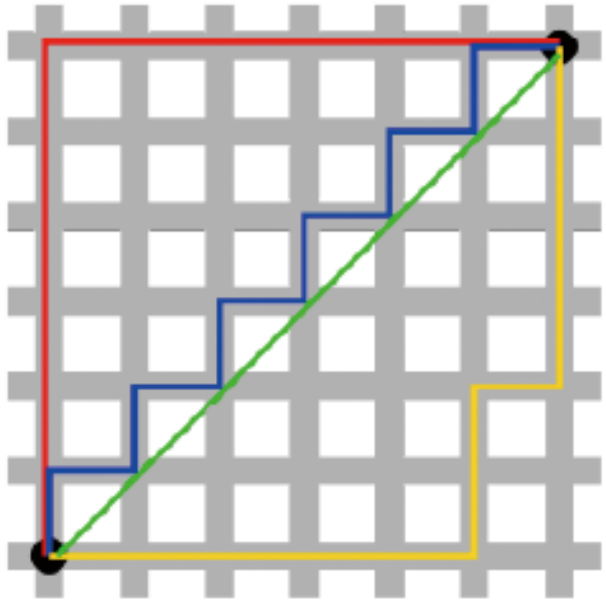
$$d_{L1}(w,v) = \sum_{i=1}^{d} |w_i - v_i|, \text{ where } w,v \in \mathbb{R}^d.$$

```
# L1 거리 구하기.
def get_l1_distance(x1, x2):
    return ((x1 - x2).abs()).sum()
```

- L2 거리 (초록색 선)
  - 유클리디안 거리 (Euclidean Distance)

$$d_{L2}(w,v) = \sqrt{\sum_{i=1}^{d} (w_i - v_i)^2}, \text{ where } w,v \in \mathbb{R}^d.$$

```
def get_l2_distance(x1, x2):
    return ((x1 - x2)**2).sum()**.5
```



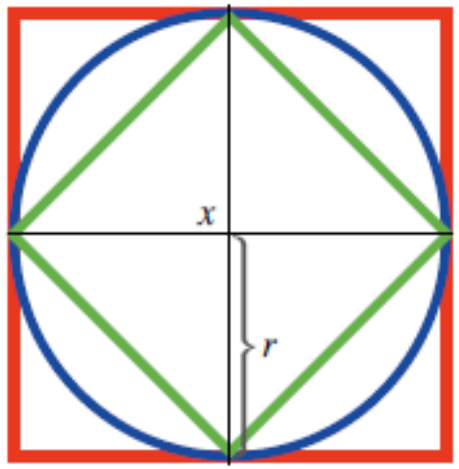
▶ L1 vs L2(초록색)_wikipedia

5.7. 벡터 유사도 구하기

- Infinity Norm (L_∞)
  - 차원 별 값의 차이 중 가장 큰 값을 표현

$$d_\infty(w,v) = \max\left(|w_1 - v_1|, |w_2 - v_2|, \cdots, |w_d - v_d|\right), \text{ where } w,v \in \mathbb{R}^d$$

```python
# Infinity Norm 구하기.
def get_infinity_distance(x1, x2):
    return ((x1 - x2).abs()).max()
```



GREEN L1 Norm
BLUE L2 Norm
RED Infinity Norm

▶ 같은 값 r 크기를 갖는 L_1, L_2, L_∞ 거리를 그림으로 나타낸 모습

- L1 : 벡터의 각 값들의 거리를 동시에 표현
- L2, L_∞ : 벡터 내 큰 값에 대해 집중하여 거리 표현

- 최적화 수행 시
  - L2 : 각 거리를 최소화하도록 최적화 사용
  - L_∞ : 전체 벡터 중 큰 값이 작아지도록 최적화 수행

5.7. 벡터 유사도 구하기

- 코사인 유사도 (Cosine Similarity)

$$\text{sim}_{\cos}(w,v) = \frac{\overbrace{w \cdot v}^{\text{dot product}}}{|w||v|} = \overbrace{\frac{w}{|w|}}^{\text{unit vector}} \cdot \frac{v}{|v|}$$

$$= \frac{\sum_{i=1}^{d} w_i v_i}{\sqrt{\sum_{i=1}^{d} w_i^2}\sqrt{\sum_{i=1}^{d} v_i^2}}$$

where $w, v \in \mathbb{R}^d$

```python
# Cosine Similarity 구하기.
def get_cosine_similarity(x1, x2):
    return (x1 * x2).sum() / ((x1**2).sum()**.5 * (x2**2).sum()**.5)
```

  - **벡터의 내적**과 동일

5.7. 벡터 유사도 구하기

- 자카드 유사도 (Jaccard Similarity)
  - **두 집합 간의 유사도**를 구하는 방법
  - 수치 자체에 대한 자카드 유사도 계산은 min, max 연산을 통해 수행

$$\text{sim}_{\text{jaccard}}(w,v) = \frac{|w \cap v|}{|w \cup v|}$$

$$= \frac{|w \cap v|}{|w| + |v| - |w \cap v|}$$

$$\approx \frac{\sum_{i=1}^{d} \min(w_i, v_i)}{\sum_{i=1}^{d} \max(w_i, v_i)}$$

where $w, v \in \mathbb{R}^d$

```python
# Jaccard Similarity 구하기.
def get_jaccard_similarity(x1, x2):
    return torch.stack([x1, x2]).min(dim=0)[0].sum() / torch.stack([x1, x2]).max(dim=0)[0].sim()
```

5.7. 벡터 유사도 구하기

- 문서 간 유사도 구하기
  - TF 또는 TF-IDF 벡터를 통해 유사도를 구할 수 있다.

5.8. 단어 중의성 해소

- 시소러스 기반 중의성 해소 : 레스크 (Lesk) 알고리즘
  - 가장 간단한 **사전 기반**의 중의성 해소 방법
  - 주어진 문장에서 **특정 단어의 의미를 명확히 할 때** 사용 가능
    - **중의성을 해소하고자 하는 단어에 대한 사전의 의미 별 설명** 과 **주어진 문장 내에 등장한 단어의 사전에서 의미별 설명** 사이의 유사도 계산
    - 가장 간단한 방법 → 겹치는 단어의 개수 구하기

[34]:
```
### 중의성 해소
## Lesk 알고리즘 구현.
def lesk(sentence, word):
    from nltk.wsd import lesk
    best_synset = lesk(sentence.split(), word)
    print(best_synset, best_synset.definition())

sentence = 'i went fishing last weekend and i got a bass and cooked it'
word = 'bass'
lesk(sentence, word)
```

물고기

```
Synset('sea_bass.n.01') the lean flesh of a saltwater fish of the family Serranidae
```

5.8. 단어 중의성 해소

- 시소러스 기반 중의성 해소 : 레스크 (Lesk) 알고리즘

```
sentence = 'i love the music from the speaker which has strong beat and bass'
word = 'bass'
lesk(sentence, word)
```

```
Synset('bass.n.02') the lowest part in polyphonic music
```

음악에서의 역할

5.9. 선택 선호도

- 문장
  - 여러 단어의 시퀀스로 구성

- 단어
  - **문장 내 주변의 단어**들에 따라 **의미가 결정**됨

- 선택 선호도
  - 이를 수치화하여 표현하는 방법
  - 예시
    - '마시다'라는 동사 → '음료' 집단에 속하는 단어가 올 확률이 매우 높음
    - '차'라는 단어가 '음료' 집단에 속하는지 '교통수단' 집단에 속하는지 파악 가능

5.9. 선택 선호도

- 선택 선호도 강도

동사가 주어졌을 때 목적어 관계에 있는 표제어 단어들 (명사)의 분포

↕

평소 문서 내 해당 명사가 나올 분포

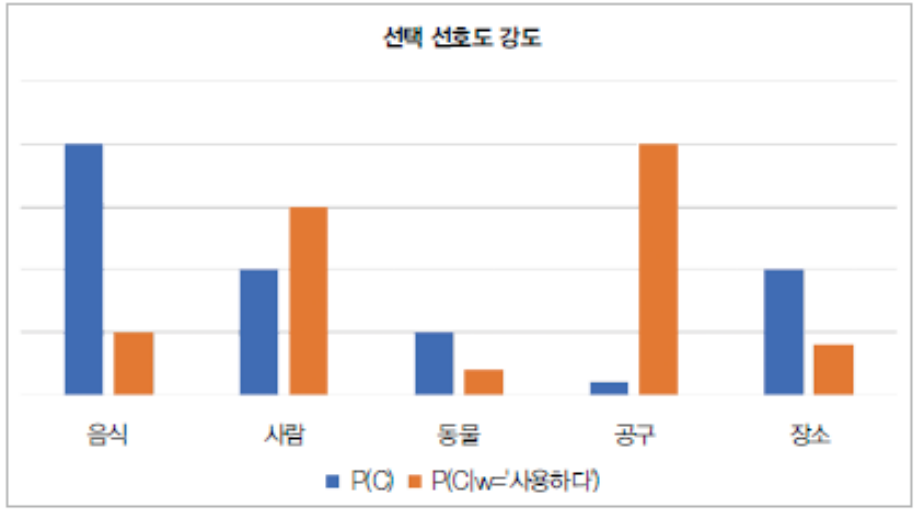
  - 이 차이가 클수록 강한 선택 선호도를 갖게 됨
  - **쿨백-라이블러 발산 (KLD)**

$$
\begin{aligned}
S_R(w) &= \mathrm{KL}(P(C \mid w) \,\|\, P(C)) \\
&= -\sum_{c \in C} P(c \mid w) \log \frac{P(c)}{P(c \mid w)} \\
&= -\mathbb{E}_{C \sim P(C \mid w)} [\log \frac{P(C)}{P(C \mid W = w)}]
\end{aligned}
$$

5.9. 선택 선호도

- 쿨백-라이블러 발산 (KLD)
  - 선택 선호도 강도 ($S_R(w)$)
    - w가 주어졌을 때 오브젝트 클래스의 분포 ($P(C|w)$)
    - 일반적인 해당 클래스들의 사전 분포 ($P(C)$)와의 KLD로 정의됨

$$
\begin{aligned}
S_R(w) &= \mathrm{KL}(P(C \mid w) \| P(C)) \\
&= -\sum_{c \in \mathcal{C}} P(c \mid w) \log \frac{P(c)}{P(c \mid w)} \\
&= -\mathbb{E}_{C \sim P(C|w)} [\log \frac{P(C)}{P(C \mid W = w)}]
\end{aligned}
$$



▶ 클래스의 사전 확률 분포와 술어가 주어졌을 때의 확률 분포 변화

5.9. 선택 선호도

- 선택 관련도

$$A_R(w,c) = -\frac{P(c|w)\log\frac{P(c)}{P(c|w)}}{S_R(w)}$$

  - 선택 선호도 강도가 낮은 술어 → 윗변이 클 경우에는 술어와 클래스 간의 더 큰 선택 관련도를 갖게 됨

  - 해당 술어가 선택 선호도 강도가 낮음 → 특정 클래스만 해당 술어에 영향을 받아 윗변의 값이 크다면, 해당 클래스에 대한 선택 관련도가 높다는 것을 추정할 수 있음

※ '마시다'라는 술어를 통해 '차'가 음료 클래스라는 것을 확인할 수 있다.
※ 하지만, '차'만 보았을 때 '교통수단'인지 '음료'인지 구분하는 것이 중요한 문제이다.

5.9. 선택 선호도

- 워드넷 기반의 선택 선호도
  - 상위어와 하위어 사전을 통해 성능이 뛰어남

$$\text{Count}_R(w,c) \approx \sum_{h \in c} \frac{\text{Count}_R(w,h)}{|\text{Classes}(h)|} \quad \Rightarrow \quad \hat{c} = \underset{c \in \mathcal{C}}{\text{argmax}} A_R(w,c), \text{ where } \mathcal{C} = \text{hypernym}(h)$$

  - c : 클래스
  - h : 실제 코퍼스에 나타난 단어
  - w : 술어
  - $|Classes(h)|$: 클래스 c 집합의 크기

5.9. 선택 선호도

- 유사 어휘를 통한 선택 선호도 평가
  - 유사 어휘 : 두 단어를 인위적으로 합성한 단어
  - 예시 : 'banana'와 'door' → 'banana-door'
  - 'eat'과 'open'을 통해 사용한 선택 선호도 알고리즘의 성능 확인 가능

5.9. 선택 선호도


```
### 유사도 기반의 선택 선호도 구현.
from konlpy.tag import Kkma

def count_seen_headwords(lines, predicate='VV', headword='NNG'):
    tagger = Kkma()
    seen_dict = dict()

    for line in lines:
        pos_result = tagger.pos(line)

        word_h = None
        word_p = None
        for word, pos in pos_result:
            if (pos == predicate) or (pos[:3] == predicate + '+'):
                word_p = word
                break
            if pos == headword:
                word_h = word

            if (word_h is not None) and (word_p is not None):
                seen_dict[word_p] = [word_h] + ([] if seen_dict.get(word_p) is None else seen_dict[word_p])
    return seen_dict

def get_selectional_association(predicate, headword, lines, dataframe, metric):
    v1 = torch.FloatTensor(dataframe.loc[headword].values)
    seens = seen_headwords[predicate]

    total = 0
    for seen in seens:
        try:
            v2 = torch.FloatTensor(dataframe.loc[seen].values())
            total += metric(v1,v2)
        except:
            pass
    return total

def wsd(predicate, headwords):
    selectional_associations = list()
    for h in headwords:
        selectional_associations += [get_selectional_association(predicate, h, lines, co, get_cosin_similarity)]
    print(selectional_associations)
```

```
wsd('피우', ['담배', '맥주', '사과'])
```