

2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

한국어와 영어 등 우리가 평소에 쓰는 말을 자연어라고 한다.

자연어 처리(NLP)를 풀어서 말하면

'우리의 말을 컴퓨터에게 이해시키기 위한 기술(분야)'이다.

자연어 처리가 추구하는 목표는 사람의 말을 부드럽게
컴퓨터가 이해하도록 만들어서,
컴퓨터가 우리에게 도움이 되는 일을 수행하게 하는 것이다.

우리의 말은 '문자로'로 구성되며, 말의 의미는 '단어'로 구성된다.

단어는 의미의 최소 단위이기 때문에 자연어를 컴퓨터에게 이해시키는 데는 '단어의 의미'를 이해시키는 것이 중요하다.

세가지 기법

- 시소러스를 활용한 기법
- 통계 기반 기법
- 추론 기반 기법(word2vec)

2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

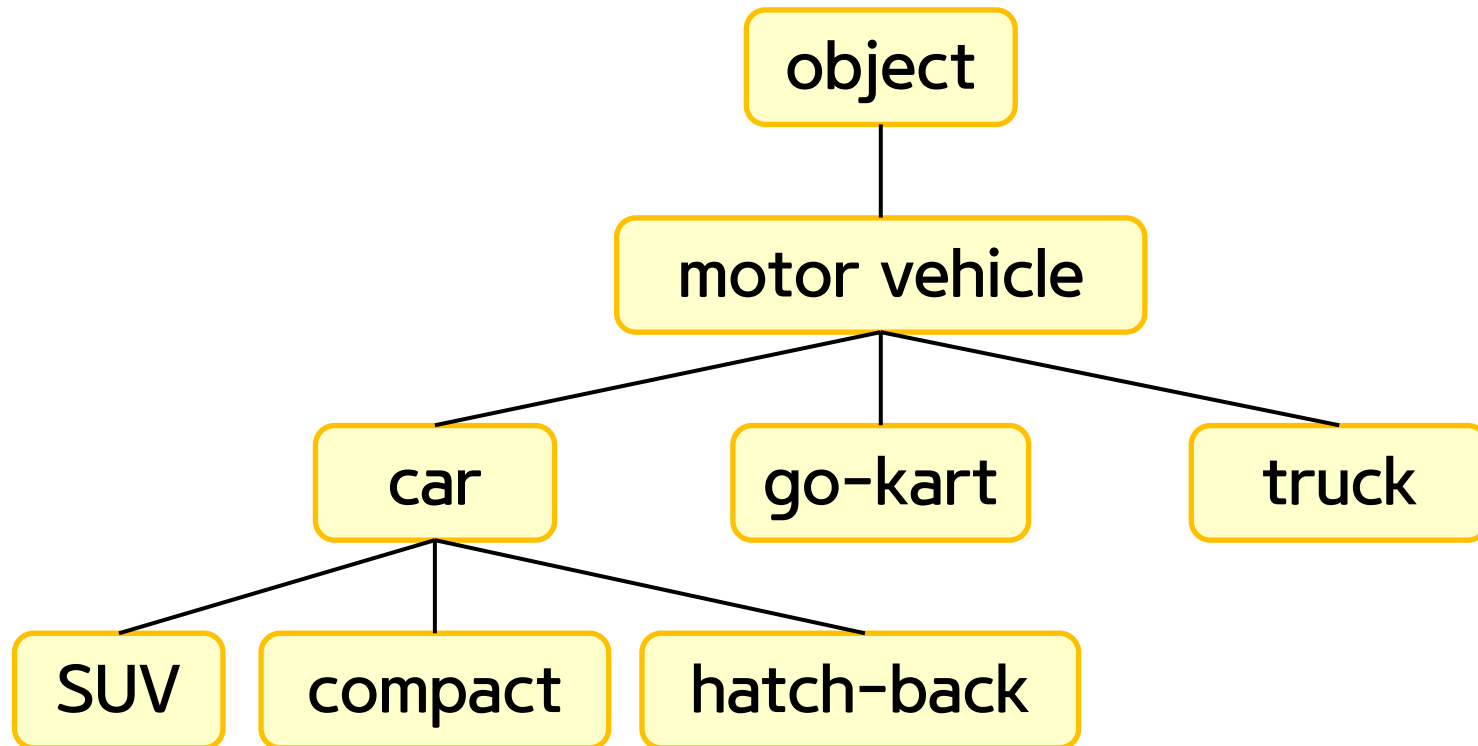
2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

동의어의 예: "car", "auto", "automobile" 등은 "자동차"를 뜻하는 동의어다.



단어들의 의미의 상/하위 관계에 기초해 그래프로 표현한다.



WordNet과 같은 시소러스에는 수많은 단어에 대한 동의어와 계층 구조 등의 관계가 정의되어 있다.

그리고 이 지식을 이용하면 '단어의 의미'를 컴퓨터에 전달할 수 있다.

하지만 사람이 수작업으로 레이블링하는 방식에는 문제들이 존재한다.

- **시대 변화에 대응하기 어렵다.**

신조어 혹은 의미 변화된 단어들을 바로 적용 시키기 어렵다.

- **사람을 쓰는 비용이 든다.**

현존하는 영어 단어의 수는 1,000만 개가 넘으며 WordNet에 등록된 단어는 20만 개 이상이다.

- **단어의 미묘한 차이를 표현할 수 없다.**

가령, 빈티지와 레트로의 의미는 같으나 용법의 차이가 존재한다.

위 문제점들을 피하기 위해 '통계 기반 기법'과 신경망을 사용한 '추론 기반 기법'을 알아볼 것이다.

2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

자연어 처리에는 다양한 말뭉치가 사용되는데

예로는 구글 뉴스와 위키백과 등의 텍스트 데이터를 들 수 있다.

※코드로 확인

['you', 'say', 'goodbye', 'and', 'i', 'say', 'hello', '.']



```
word_to_id = {}
id_to_word = {}
```

$\text{hash}(\text{'say'}) \% 8$

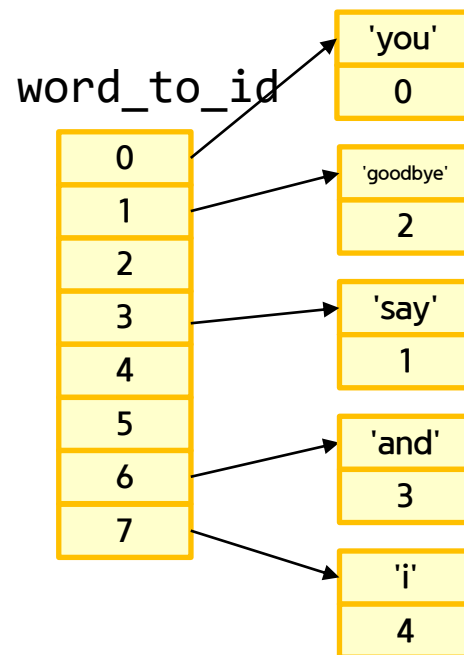
```
for word in words:
```

```
    if word not in word_to_id:
```

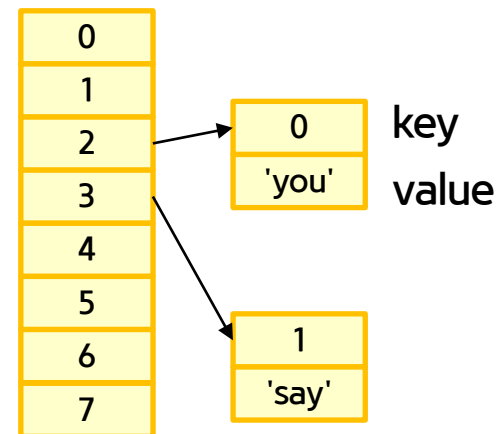
```
        new_id = len(word_to_id)
```

```
        word_to_id[word] = new_id
```

```
        id_to_word[new_id] = word
```



id_to_word



자연어 처리에는 다양한 말뭉치가 사용되는데

예로는 구글 뉴스와 위키백과 등의 텍스트 데이터를 들 수 있다.

※코드로 확인

색에는 고유한 이름이 붙여진 다채로운 색들도 있고, RGB(Red/Green/Blue)라는 세가지 성분이 어떤 비율로 섞여 있느냐로 표현하는 방법이 있다. 전자는 색의 가짓수만큼 의 이름을부여하는 반면에 후자는 색을 3차원의 벡터로 표현한다.

여기서 주목할 점은 RGB같은 벡터 표현이 단 3개의 성분으로 간결하게 표현할 수 있고, 색을 더 정확하게 명시할 수 있다는 점이다.

'색'을 벡터로 표현하듯 '단어'도 벡터로 표현할 수 있다. 이를 단어의 '분산 표현'이라고 한다.

'Red' = (255,0,0)

'Blue' = (0,0,255)

'Green' = (0,255,0)

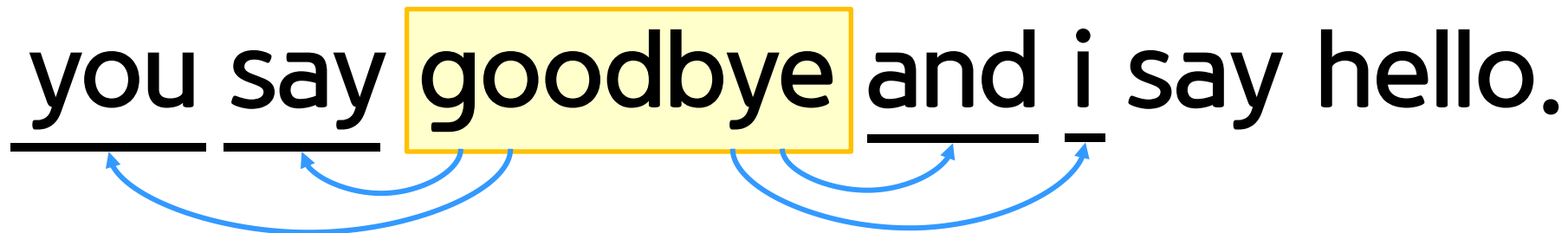
'Yellow' = (255,255,0)

분포 가설이란 단어의 의미는 주변 단어에 의해 형성된다는 것이다.
분포 가설이 말하고자 하는 것은 단어 자체에는 의미가 없고,
그 단어가 사용된 '맥락'이 의미를 형성한다는 것이다.

예를 들어, I drink beer를 I guzzle beer라고 해도 guzzle을 drink로 이해할 수 있다는 것이다.

윈도우 크기가 2인 '맥락'의 예. 단어 "goodbye"에 주목한다면,
그 좌우의 두 단어(총 네 단어)를 맥락으로 이용한다.

you say **goodbye** and i say hello.



위 그림에서 goodbye를 기준으로 좌우의 두 단어씩이 '맥락'에 해당한다.
맥락의 크기를 '윈도우 크기'라고 한다. 여기서는 '윈도우 크기'가 2이기 때문에
좌우로 두 단어씩이 맥락에 포함된다.

분포 가설에 기초해 단어를 벡터로 나타내는 방법을 생각해보면
주변 단어를 세어보는 방법이 떠오를 것이며 이를 '통계 기반' 기법이라고 한다.

단어 "you"의 맥락을 세어본다.

you say goodbye and i say hello.



단어가 총 7개이며 윈도우 크기는 1로 하고 단어 ID가 0인 'you'부터
단어의 맥락에 해당하는 단어의 빈도를 세어보겠다.
'you'의 맥락은 'say'라는 단어 하나뿐이다.

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0

단어 "say"의 맥락을 세어본다.

you **say** goodbye and i **say** hello.

'say'라는 단어는 벡터 $[1, 0, 1, 0, 1, 1, 0]$ 으로 표현할 수 있다.

	you	say	goodbye	and	i	hello	.
say	1	0	1	0	1	1	0

모든 단어 각각의 맥락에 해당하는 단어의 빈도를 세어 표로 정리 한다.

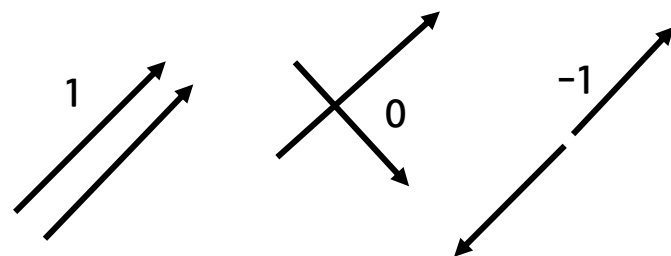
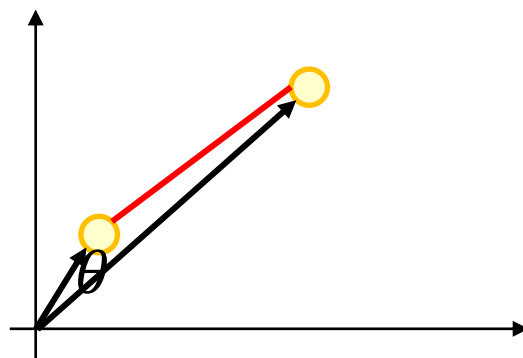
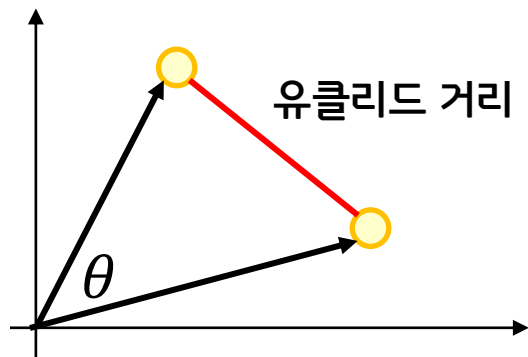
	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

위의 표는 모든 단어에 대해 동시발생하는 단어를 표에 정리한 것이다.
 위 표의 각 행은 벡터이며 행렬의 형태를 띄어 동시발생 행렬이라 한다.

단어 벡터의 유사도를 나타낼 때는 코사인 유사도를 자주 이용한다.

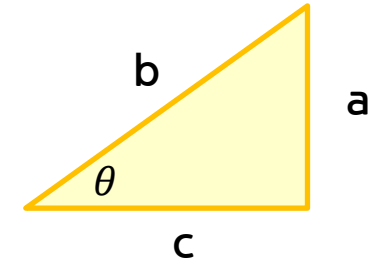
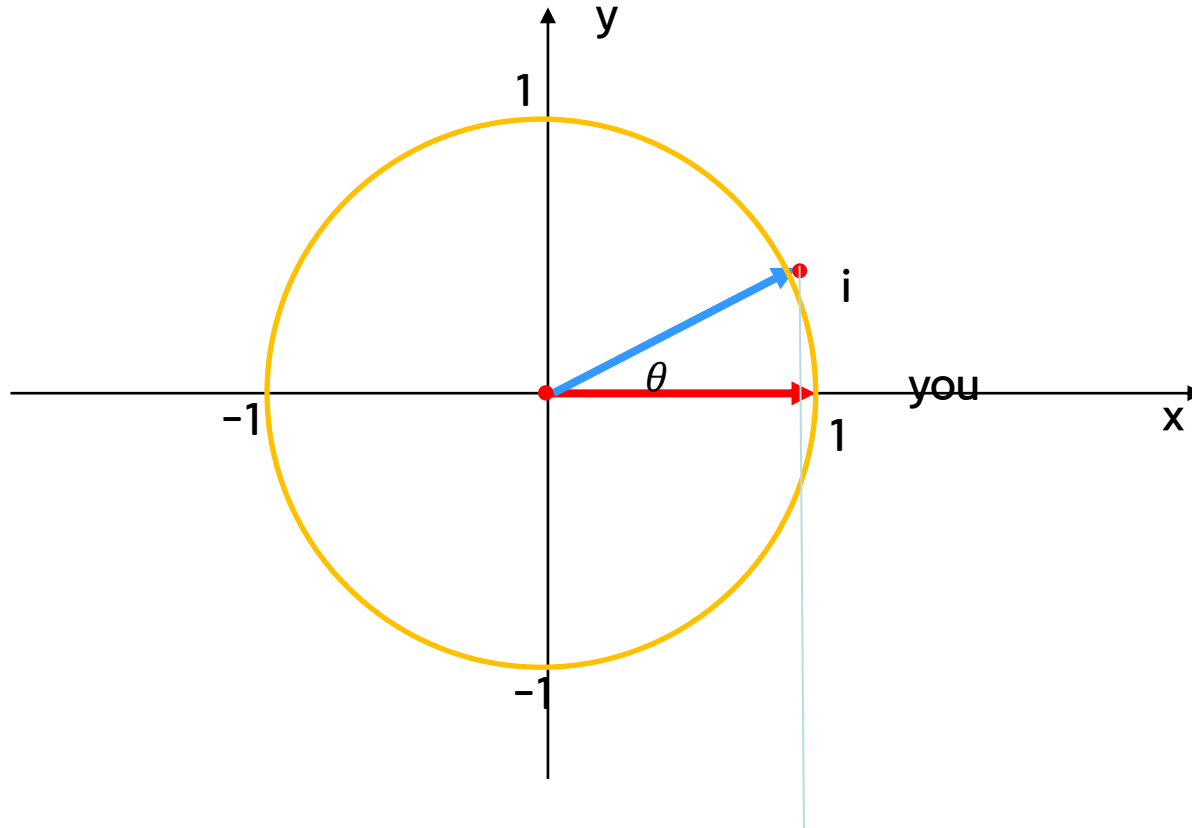
$$\text{similarity}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{x_1 y_1 + \dots + x_n y_n}{\sqrt{x_1^2 + \dots + x_n^2} \sqrt{y_1^2 + \dots + y_n^2}}$$

위의 식처럼 정의되며 분자에는 벡터의 내적이 분모에는 벡터의 노름(크기)이 등장한다.
위 식의 핵심은 벡터를 정규화하고 내적을 구하는 것이다.



Win : 7 * 3

코사인 유사도



$$\cos(\theta) = b/c$$

$$\text{similarity}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{x_1 y_1 + \dots + x_n y_n}{\sqrt{x_1^2 + \dots + x_n^2} \sqrt{y_1^2 + \dots + y_n^2}}$$

코사인 유사도를 이용하여 어떤 단어가 주어지면,
그 검색어와 비슷한 단어를 유사도 순으로 출력하는 함수를 만들어 본다.

이를 구현하기 위한 코드에는 밑에 같은 함수의 인수들이 쓰인다.

인수명	설명
query	검색어(단어)
word_to_id	단어에서 단어 ID로으 딕셔너리
id_to_word	단어 ID에서 단어로의 딕셔너리
word_matrix	단어 벡터들을 한데 모은 행렬, 각 행에는 대응하는 단어의 벡터가 저장되어 있다고 가정한다.
top	상위 몇 개까지 출력할지 설정

※ 코드 참조

```
def create_co_matrix(corpus, vocab_size, window_size=1):
    corpus_size = len(corpus)
    co_matrix = np.zeros((vocab_size, vocab_size), dtype=np.int32)
```

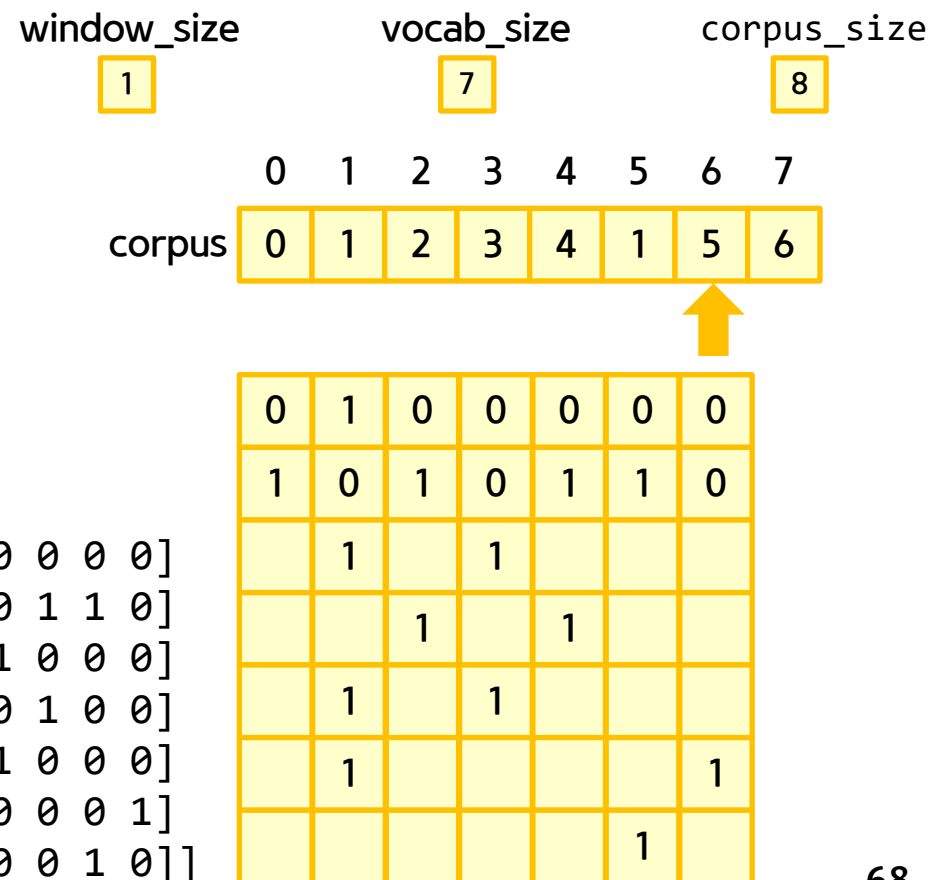
```
    for idx, word_id in enumerate(corpus):
        for i in range(1, window_size + 1):
            left_idx = idx - i
            right_idx = idx + i

            if left_idx >= 0:
                left_word_id = corpus[left_idx]
                co_matrix[word_id, left_word_id] += 1

            if right_idx < corpus_size:
                right_word_id = corpus[right_idx]
                co_matrix[word_id, right_word_id] += 1
```

```
    return co_matrix
```

```
[[0 1 0 0 0 0 0]
 [1 0 1 0 1 1 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 0 0]
 [0 1 0 1 0 0 0]
 [0 1 0 0 0 0 1]
 [0 0 0 0 0 1 0]]
```



2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

동시발생 행렬의 원소는 두 단어가 동시에 발생한 횟수를 나타내지만 '발생' 횟수라는 것은 사실 좋은 특징이 아니다.
예를 들어 'the' 와 'car'의 동시발생을 생각해보자.
'...the car...'라는 문구가 자주 보일 것이며 'car'와 'drive'는 관련이 깊다.
하지만 'the'가 고빈도 단어이기 때문에 'car'와 더 관련이 있어 보이게 결과가 나올 수 있다.

이를 해결하기 위해 점별 상호정보량이라는 척도를 사용한다.
PMI(Pointwise Mutual Information)는 확률 변수 x 와 y 에 대해 다음 식으로 정의 된다.

$$PMI(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

$P(x)$ 는 x 가 일어날 확률, $P(y)$ 는 y 가 일어날 확률, $P(x, y)$ 는 x, y 가 동시에 일어날 확률이다.
PMI값이 높을수록 관련성이 높다는 의미이다.

$$PMI(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} = \log_2 \frac{\frac{C(x, y)}{N}}{\frac{C(x)}{N} \frac{C(y)}{N}} = \log_2 \frac{C(x, y) \cdot N}{C(x)C(y)}$$

여기서 C는 동시발생 행렬, C(x,y)는 단어 x와 y가 동시발생하는 횟수, C(x)와 C(y)는 각각 단어 x와 y의 등장 횟수이며 N은 말뭉치에 포함된 단어 수이다.

이 식을 토대로 1,000번 등장한 'the', 20번 등장한 'car'와 10번 등장한 'drive'를 계산해보자.

$$PMI("the", "car") = \log_2 \frac{10 \cdot 10000}{1000 \cdot 20} \approx 2.32$$

$$PMI("car", "drive") = \log_2 \frac{5 \cdot 10000}{20 \cdot 10} \approx 7.97$$

두 PMI의 결과를 살펴보면 'car'와 'drive'의 관계성이 강하다는 것을 볼 수 있다. 이러한 결과가 나온 이유는 단어가 단독으로 출현하는 횟수가 고려되었기 때문이다. 이 예에서는 'the'가 자주 출현하였기 때문에 PMI값이 낮아진 것이다.

하지만 PMI에도 문제가 하나 있다.

이는 두 단어의 동시발생 횟수가 0이면 $\log(0,2) = -\infty$ 가 된다.

이 문제를 피하기 위해 실제 구현할 때는 양의 상호정보량(PPMI)를 사용한다.

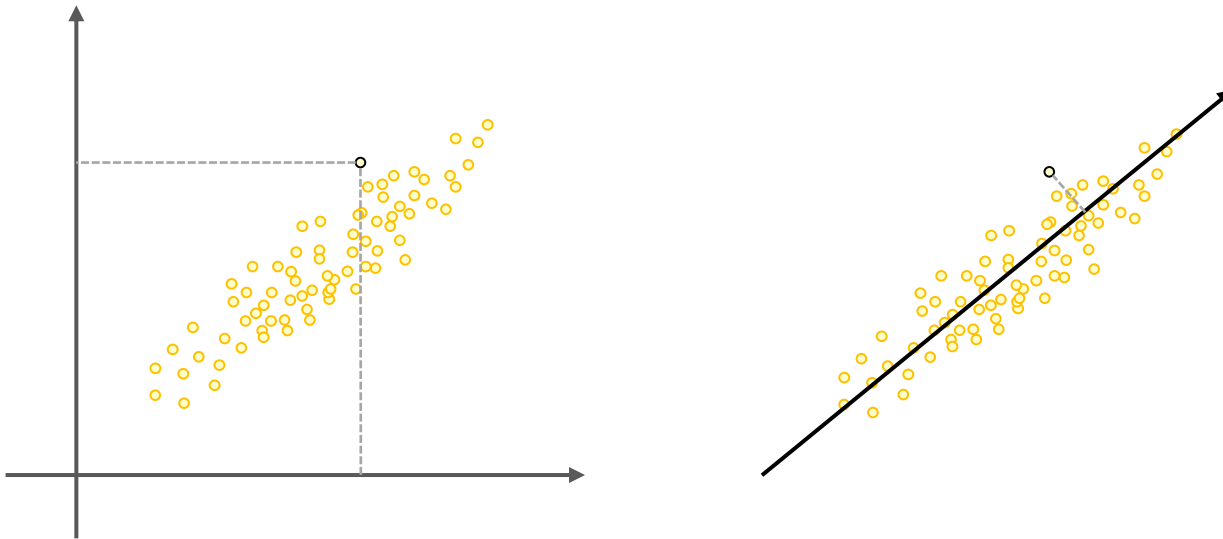
$$PPMI(x, y) = \max(0, PMI(x, y))$$

이 식에 따라 PMI가 음수인 때는 0으로 취급하며 단어 사이의 관련성을 0 이상의 실수로 나타낼 수 있다.

하지만 PPMI 행렬에도 문제가 있는데 말뭉치의 어휘 수가 증가함에 따라 각 단어 벡터의 차원 수도 증가한다는 문제이다.

이 문제를 대처하고자 자주 수행하는 기법이 '벡터의 차원 감소'이다.

그림으로 이해하는 차원 감소: 2차원 데이터를 1차원으로 표현하기 위해 중요한 축을 찾는다.



위의 그림 예시처럼 데이터의 분포를 고려해 중요한 '축'을 찾는 일을 수행한다.
왼쪽 그림은 데이터점들을 2차원 좌표에 표시한 모습이고
오른쪽 그림은 새로운 축을 도입하여 똑같은 데이터를 좌표축 하나만으로 표시했다.

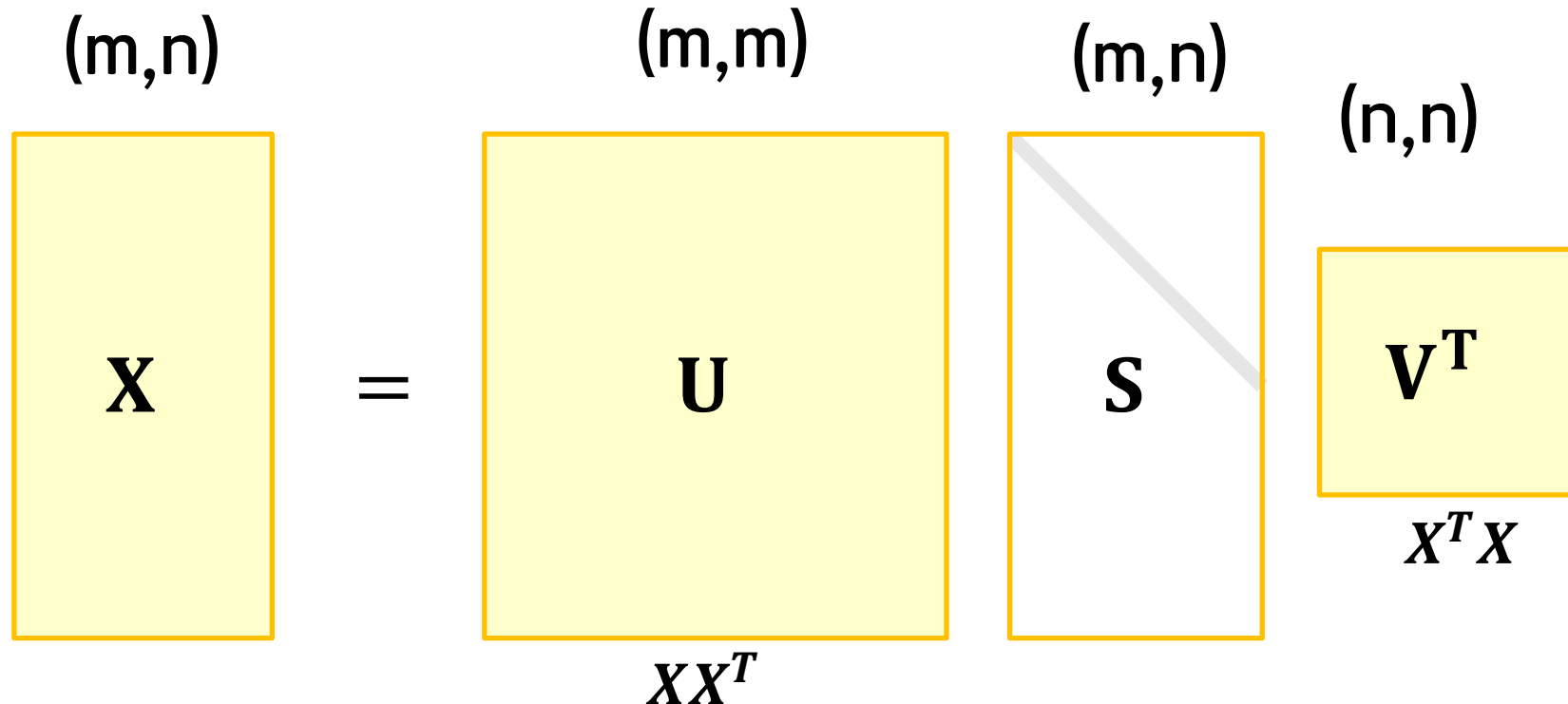
여기서 중요한 것은 가장 적합한 축을 찾아내는 일로,
1차원 값만으로 데이터의 본질적인 차이를 구별할 수 있어야 한다.
그리고 다차원 데이터에 대해서도 수행 가능하다.

차원을 감소시키는 방법 중 하나인 특잇값분해(SVD)는 임의의 행렬을 세 행렬의 곱으로 분해하며, 수식으로는 다음과 같다.

$(n,m)(m,n)$

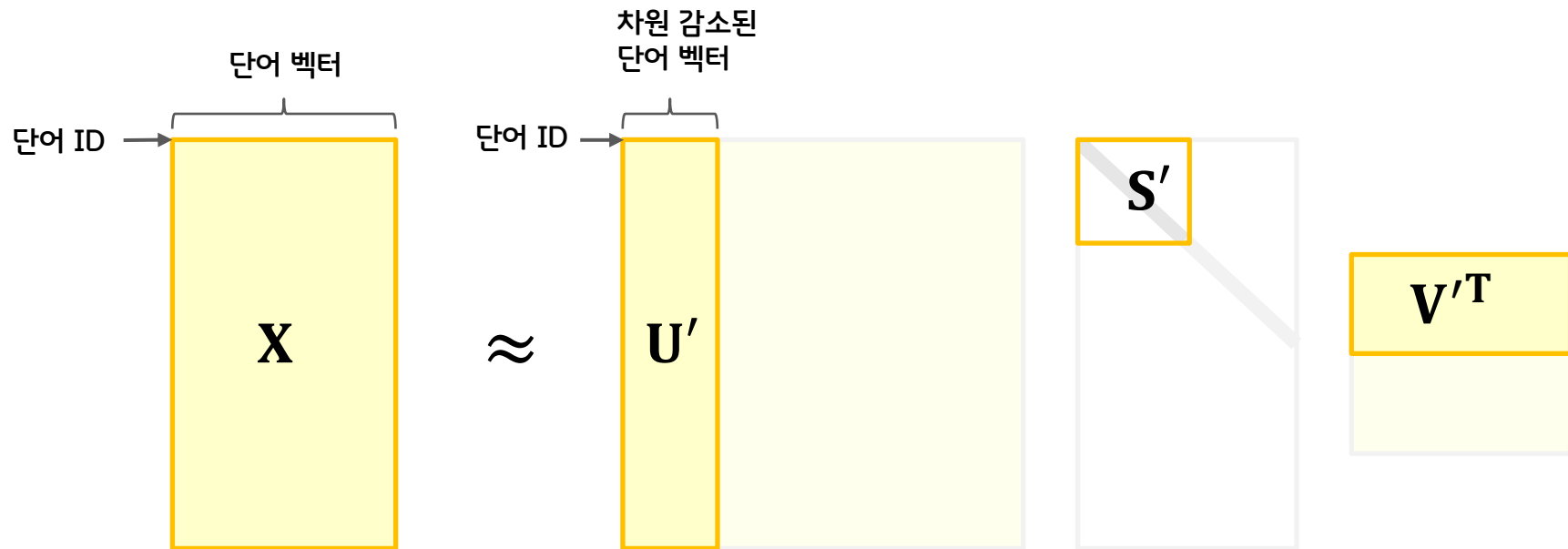
$$X = USV^T$$

SVD에 대한 행렬의 변환(행렬의 '흰 부분'은 원소가 0임을 뜻함)



행렬 S 에서 특 값이 작다면 중요도가 낮다는 뜻이므로
행렬 U 에서 여분의 열벡터를 깎아내려 원래의 행렬을 근사할 수 있다.

SVD에 의한 차원의 감소



이를 '단어의 PPMI 행렬'에 적용하면 행렬 X 의 각 행에는 해당 단어 ID의 단어 벡터가 저장되어 있으며, 그 단어 벡터가 행렬 U' 라는 차원 감소된 벡터로 표현된다.

우리가 사용할 PTB(펜 트리뱅크) 말뭉치는 word2vec의 발명자인 토마스 미콜로프의 웹 페이지에서 받을 수 있다.

※ 코드 참조

결과적으로 말뭉치를 사용해 맥락에 속한 단어의 등장 횟수를 센 후 PPMI 행렬로 변환하고 다시 SVD를 이용해 차원을 감소시킴으로서 더 좋은 단어 벡터를 얻었다. 이것이 단어의 분산 표현이고, 각 단어는 고정 길이의 밀집벡터로 표현되었다.