차원의 저주, 차원 축소

- 차원의 저주
- 주성분 분석 PCA
- [추가]t-SNE

변수가 많을수록 모델 성능 향상?

✓차원(dimension)

- 차원은 여러가지 의미로 사용되지만,
- 여기서는 **차원의 수 = 변수의 수**

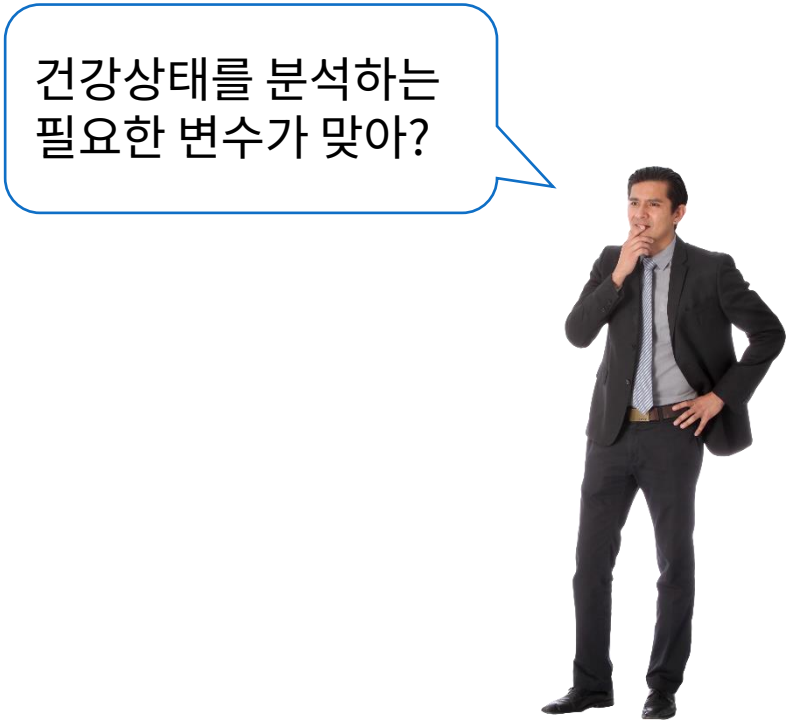
✓다양한 변수를 고려 ➔ 모델 성능 향상

- 예, **고객의 건강상태 분석**
  - **키**, **몸무게** (2차원) ➔ 기본적인 분석
  - **+ 혈압**, **체성분 지수**, **나이 (5차원) :** ➔ 더 구체적인 건강상태 분석 가능

변수가 많을수록 모델 성능 향상?

✓이어서 더 많은 변수 수집

- 성별
- 출신지역
- 소득수준
- 자녀 수
- 주거형태
- …



✓변수를 계속 추가하다 보니…

- 꼭 필요한 데이터가 아닌데 포함
  - 오히려 방해가 될 수도
  - 불필요하게 복잡한 모델

- 더 큰 문제는 데이터가 굉장히 **희박**해 집니다!

희박한(sparse) 데이터

키
작은
작은
작은
중간
중간
중간
중간
큰
큰
큰

- 1차원 3구간
- 첫 구간(작은) : 3건

키	몸무게
작은	가벼운
작은	중간
작은	무거운
중간	중간
중간	중간
중간	무거운
중간	가벼운
큰	중간
큰	무거운
큰	무거운

- 2차원 각각 3구간
- 작은 키, 가벼운 몸무게 구간 : 1건

키	몸무게	혈압
작은	가벼운	정상
작은	중간	정상
작은	무거운	고혈압
중간	중간	저혈압
중간	중간	정상
중간	무거운	고혈압
중간	가벼운	정상
큰	중간	저혈압
큰	무거운	정상
큰	무거운	고혈압

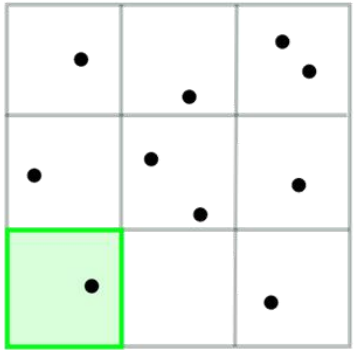
- 3차원 각각 3구간
- 작은 키, 가벼운 몸무게, 저혈압 구간 : 0건

차원이 늘어날 수록 데이터가 희박해 집니다.

희박한(sparse) 데이터

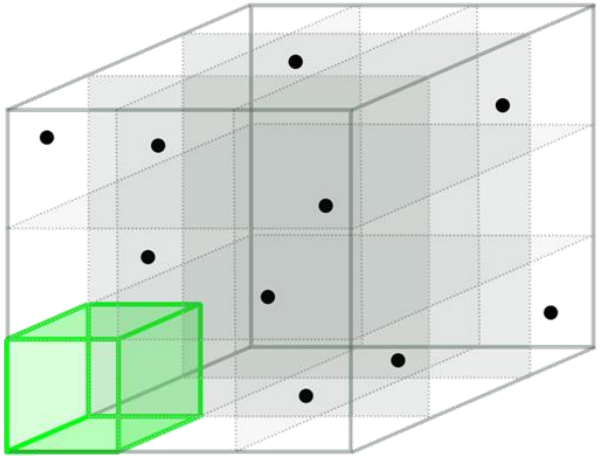
3/10

- 1차원 3구간
- 첫 구간(작은) : 3건



1/10

- 2차원 각각 3구간
- 작은 키, 가벼운 몸무게 구간 : 1건



0/10

- 3차원 각각 3구간
- 작은 키, 가벼운 몸무게, 저혈압 구간 : 0건

차원이 늘어날 수록 데이터가 희박해 집니다.

차원의 저주(Curse of Dimension)

✓변수가 많아지면

- 조건(작은 키, 가벼운 몸무게, 저혈압)에 맞는 데이터 **희박**
  ➔ 학습이 적절하게 되지 않을 가능성 높아짐.

✓이를 차원의 저주이라고 부릅니다.

- ~~많다고 꼭 좋은 건 아니...~~

차원의 저주 : 고차원 문제?

✓**고차원 문제의 본질 : [희박한 데이터 문제]**

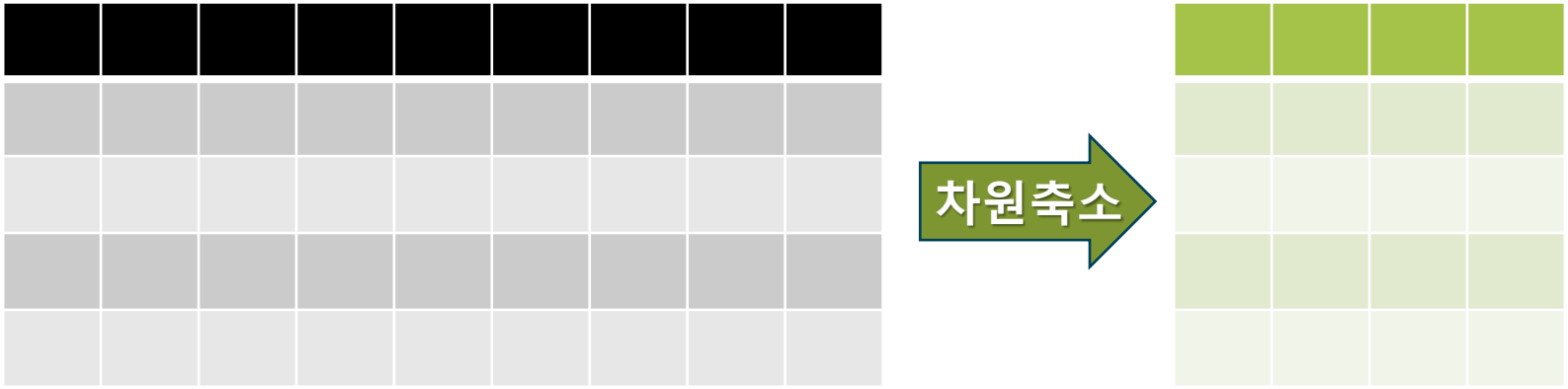
✓**[희박한 데이터 문제] 해결방안**

- ① **행**을 늘리기. ➔ 데이터 늘리기
- ② **열**을 줄이기. ➔ **차원 축소**

해결방안 : 차원 축소

✓차원 축소

- 다수의 feature (고차원) ➔ 새로운 소수의 feature (저차원)로 축소
- **기존 특성을 최대한 유지**



✓대표적인 방법 : 주성분 분석(PCA), t-SNE

차원을 줄이는 방법

✓주성분 분석(PCA Principal Component Analysis)

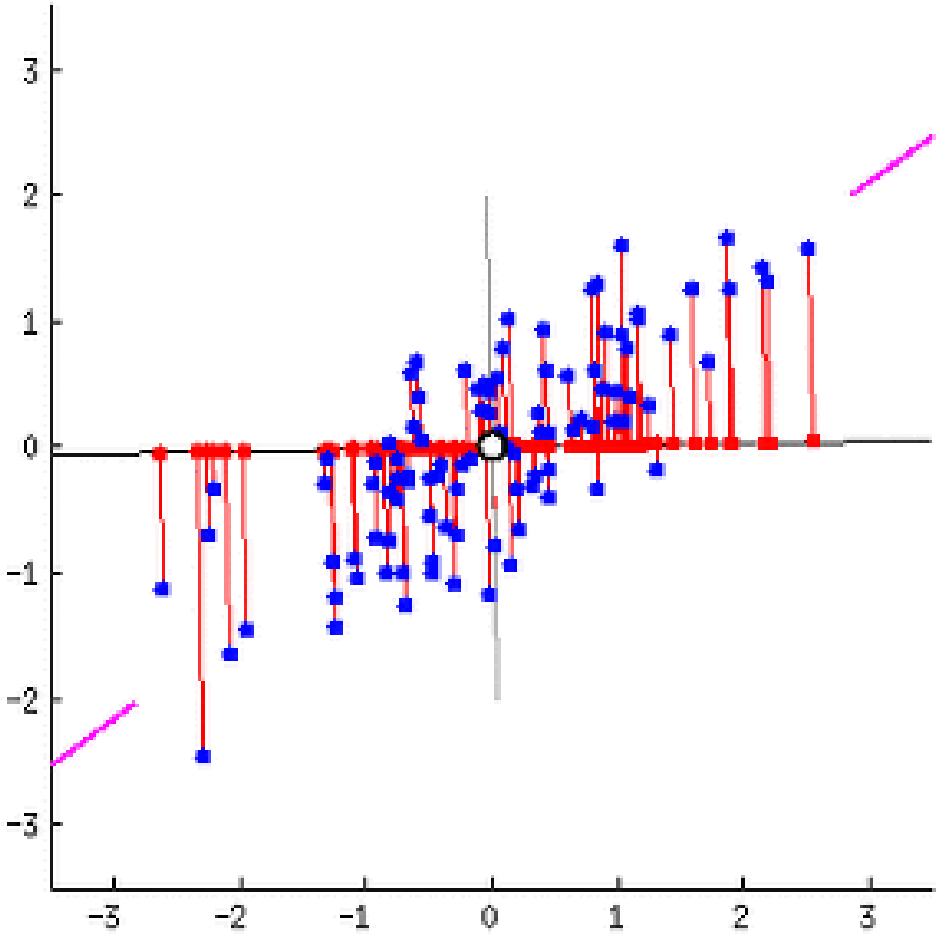
- 변수(차원)의 수보다 적은 저차원의 평면으로 **투영** Projection
- 평면1, 평면2 중 럭비 공의 특징을 **잘 반영한 평면**은 무엇일까요?
  (어떤 그림자가 럭비공과 비슷하나요?)

PCA Principal Component Analysis 사용해 보기

일단
주성분 분석을
사용해 봅시다!

주성분 분석 PCA

✓2차원 데이터를 1차원으로 투영해 봅시다.



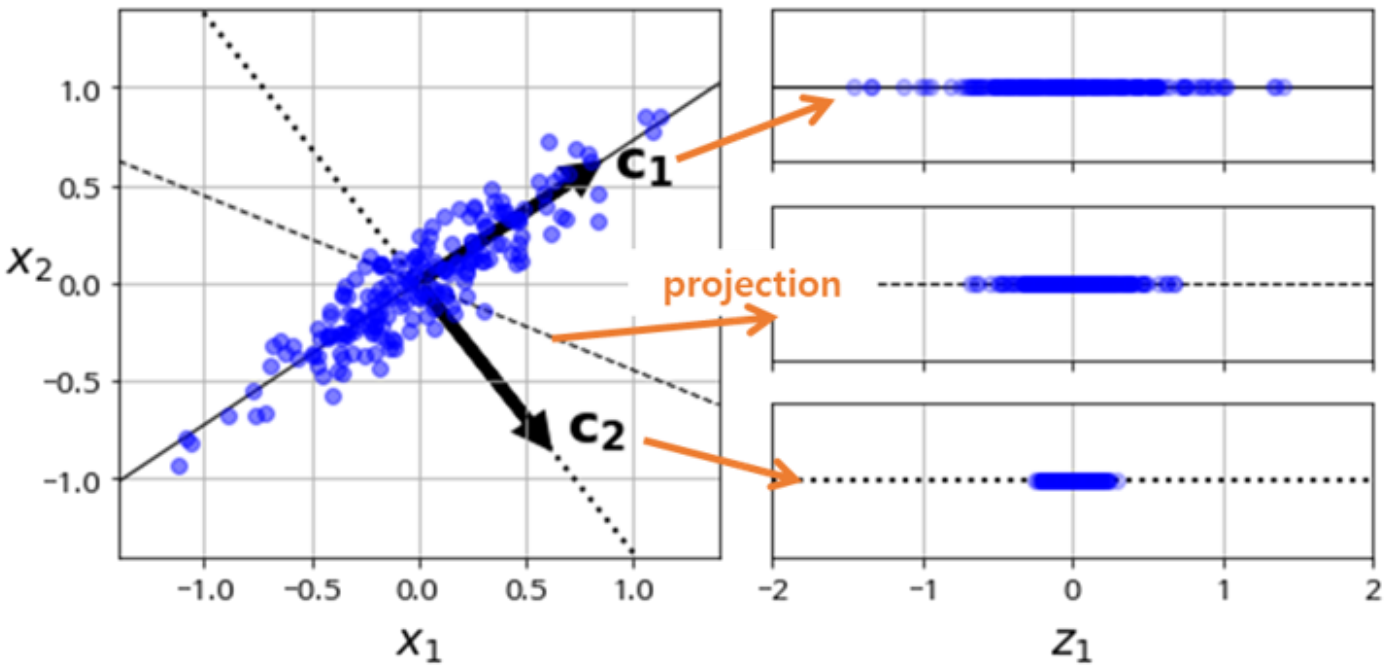
출처 : https://imgur.com/Uv2dlsH

주성분 분석 PCA

✓**어떤 평면으로 투영시킬 것인가?**

- 정보의 특성을 최대한 유지하면서
- 차원을 축소

} **분산**을 최대한 유지



분산이 가장 큼

분산이 가장 작음

주성분 분석 PCA

✓PCA 절차

① 학습 데이터셋에서 **분산이 최대**인 **첫번째 축(axis)**을 찾음.
② 첫번째 축과 직교(orthogonal)하면서 **분산이 최대**인 **두 번째 축**을 찾음.
③ 첫 번째 축과 두 번째 축에 직교하고 **분산이 최대**인 **세 번째 축**을 찾음.
④ ①~③과 같은 방법으로 데이터셋의 차원 만큼의 축을 찾음.

✓주성분 분석을 수행하면…

- 각 축의 단위벡터를 **주성분**이라고 부르고,
- 각 축 별 투영된 값이 저장됨

PCA 사용하기

✓전처리 : 스케일링필요

- 주성분 결정시, 분산 비교(크기 비교)
- 스케일링 없이 PCA를 수행 ➔ 스케일이 가장 큰 변수에 영향을 가장 많이 받게 됨

✓PCA 문법

- 선언 : 주성분의 개수(n) 지정
  - 1 ≤ n ≤ feature의 수
  - 생성 후 조정할 수 있음
- 적용
  - x_train으로 fit & transform
  - 다른 데이터는 transform
  - 결과는 numpy array가 됨.

Python Code

```
# 주성분 분석 선언
pca = PCA(n_components=n)

# 만들고, 적용하기
x_train_pc = pca.fit_transform(x_train)
x_val_pc = pca.transform(x_val)
```

주성분 분석 PCA Principal Component Analysis

✓주성분의 개수 정하기

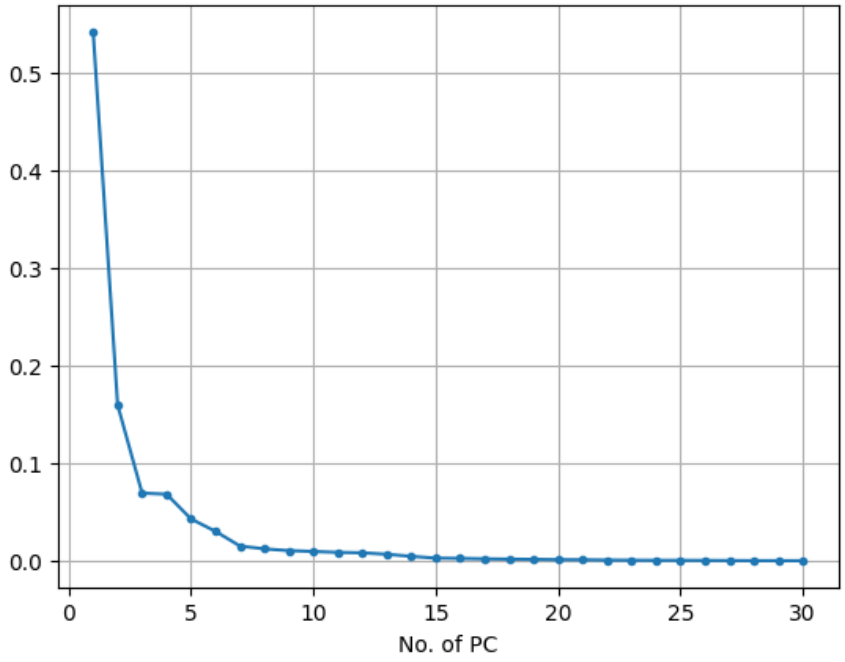
- 주성분의 개수를 늘려가면서
- 원본 데이터 분산과 비교
- Elbow Method

Python Code

```
plt.plot(range(1,n+1), pca.explained_variance_ratio_,
         marker = '.')
plt.xlabel('No. of PC')
plt.grid()
plt.show()
```

- **.explained_variance_ratio_**

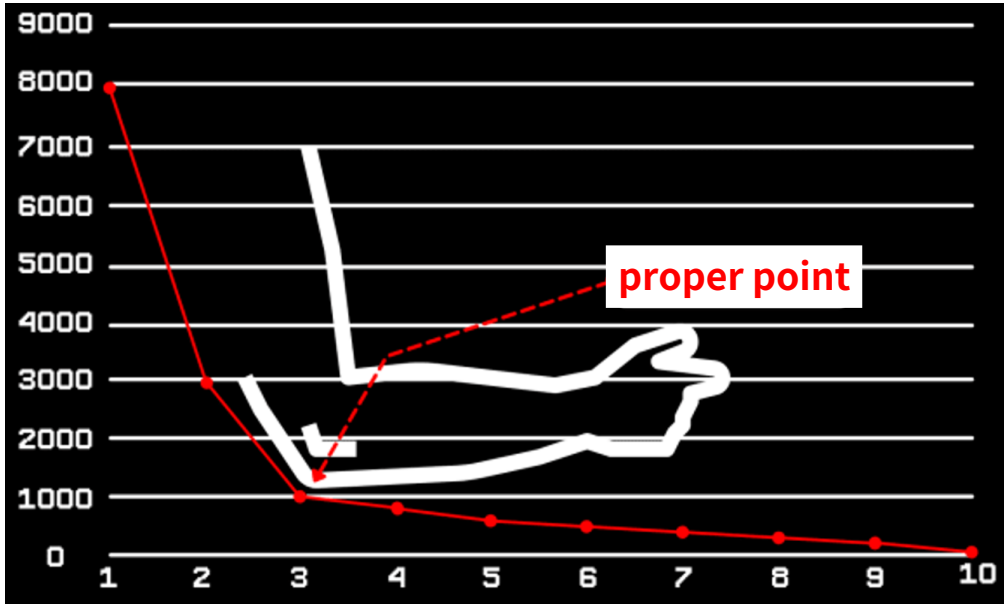
원본데이터의 전체 분산 대비
누적 주성분의 차이 비율



Elbow Method

✓팔꿈치 지점 근방에서 적절한 값을 찾아라.

- X축의 값이 계속 증가해도, y축의 값의 개선 폭이 줄어 드는 지점
- Trade-Off 관계일 때, 적절한 지점을 찾기 위한 휴리스틱 방법.



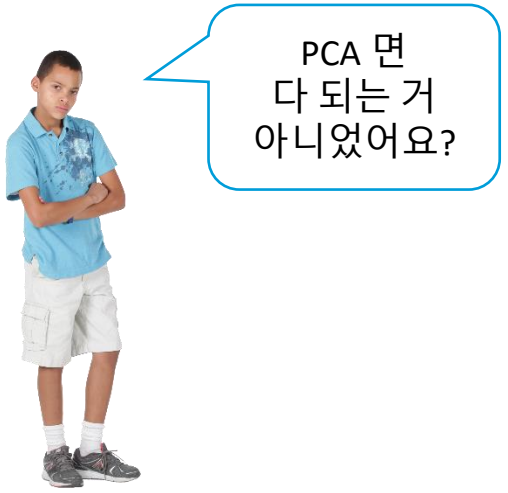
휴리스틱 법(heuristics) : 불충분한 시간이나 정보로 인하여 합리적인 판단을 할 수 없거나, 체계적이면서 합리적인 판단이 굳이 필요하지 않은 상황에서 사람들이 빠르게 사용할 수 있게 보다 용이하게 구성된 간편추론의 방법

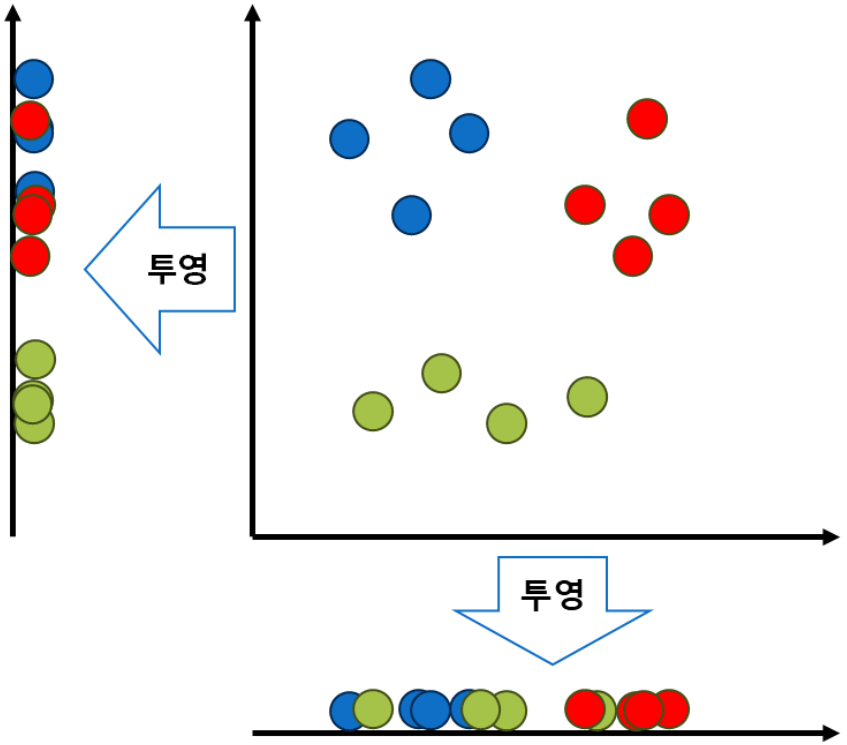
PCA의 단점

✓아래 2차원 데이터를 1차원으로 투영 한다면…

✓PCA → 선형 축소 방식

- 분산의 크기만 고려한 **선형**축소방식
- 저차원에서 특징을 잘 담아내지 못하는 경우 발생



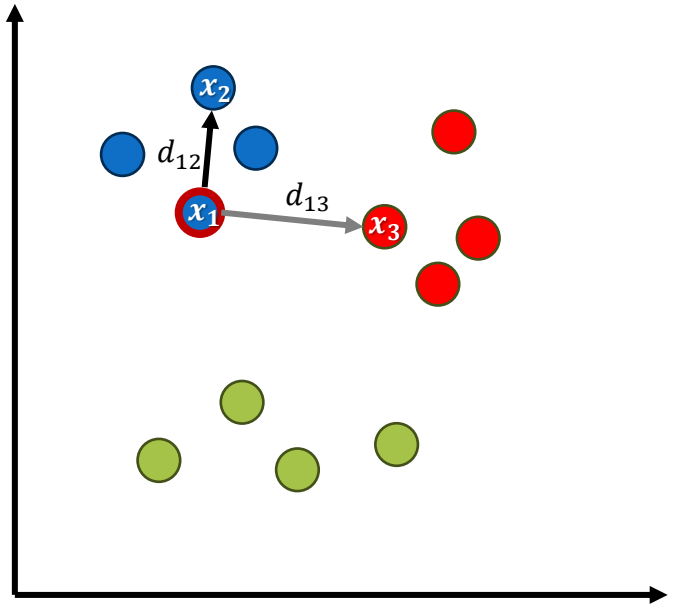


t-SNE t-distributed Stochastic Neighbor Embedding

✓원본의 특성을 최대한 살리면서 축소하는 방법이 필요!

- ① 원본에서 가까운 거리의 점들은 ➔ 원본의 유사도 맵 생성
- ② 축소한 후에도 가깝게 만들자. ➔ 축소된 데이터의 유사도 맵

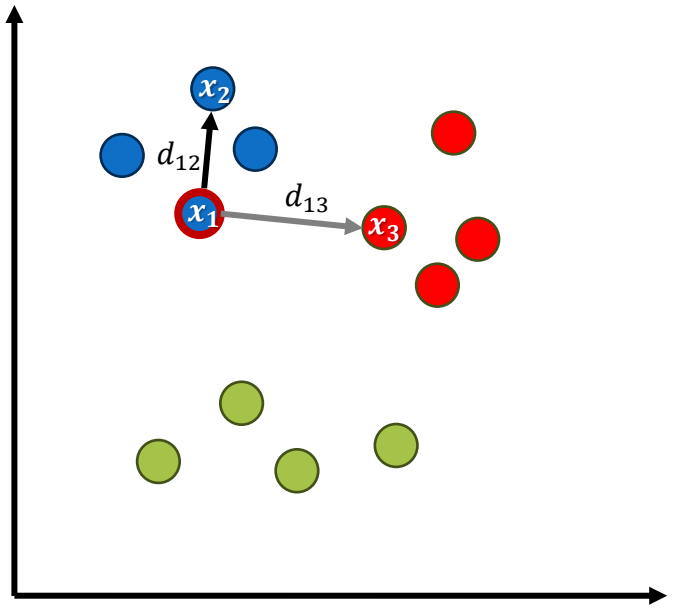
두 맵의 오차를 줄이는 방향으로 축소된 데이터 조정



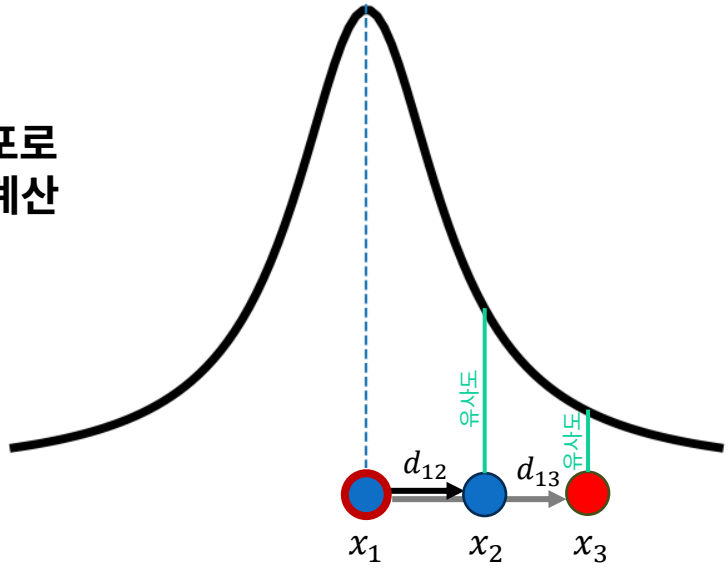
t-SNE 원리

✓원본 데이터에서 유사도 맵 만들기

① (유클리드) 거리 계산



② t-분포로 유사도 계산



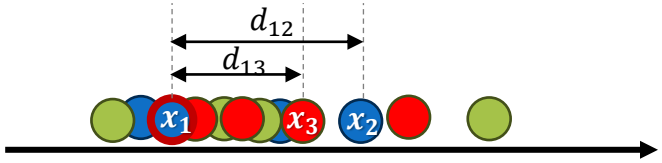
③ 유사도 맵

	x_1	x_2	x_3	...
x_1		0.12	0.05	
x_2				
x_3				
...				

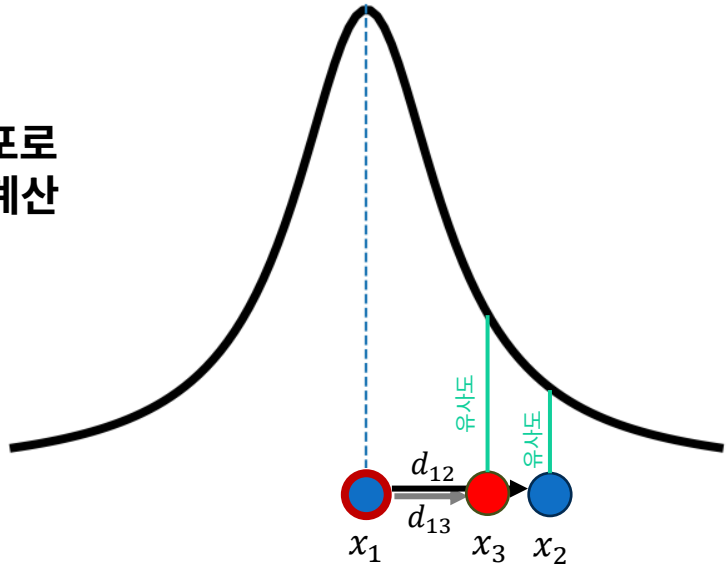
t-SNE 원리

✓저 차원 데이터에서 유사도 맵 만들기

① 저차원에서 유클리드 거리 계산



② t-분포로
유사도 계산



③ 유사도 맵

	x_1	x_2	x_3	...
x_1		0.08	0.11	
x_2				
x_3				
...				

④ 두 유사도 맵 간의 오차 계산
(KL Divergence)

⑤ 저차원에서 데이터 위치 조정

반복

t-SNE 사용하기

✓전처리 : 스케일링

- 꼭 필요한 것은 아니지만, 해주는 것을 권장

✓학습

- 고차원을 2 ~ 3 차원으로 축소하여
- 주로 **데이터 시각화**를 위해 사용됨
- 학습하는데 오래 걸림

Python Code

```
from sklearn.manifold import TSNE

# 2차원으로 축소하기
tsne = TSNE(n_components = 2)
x_tsne = tsne.fit_transform(x)
```