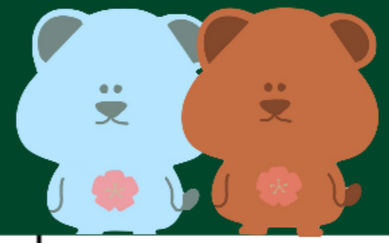
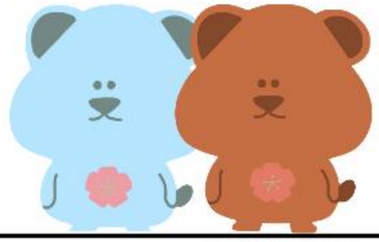


7주차 발표

DA팀 손소현 오수진 이서영

EWHA EURON

목차



EWHA EURON

1. 차원의 저주

EWHA EURON

1. 차원의 저주

야외 활동에 적합한 날씨인지 분류하는 머신러닝 모델을 만들어 보자 (good=1, bad=0)

가령 101개의 야외활동과 관련된 항목들과 그 수치가 나열된 데이터 테이블이 있다고 한다면

	온도	**습도**	**강수량**	**미세먼지**	**풍속**	**태풍여부**	**...**	교통량	유동인구
0801	32	**15**	**66**	3	111	0		15	40326127
0802									
...									
0831									

온도, 강수량, 풍속 등 많은 요인이 존재

주어진 데이터를 학습 모델에 그대로 입력하면 총 101차원 벡터가 됨

1. 차원의 저주

101개의 특성을 학습하면 되면, 여러 문제들이 발생할 수 있다!

1) 훈련속도가 느려진다.

2) 시각화가 어렵다. 우리가 인식할 수 있는 범위는 3차원까지 뿐, 101차원은 데이터 패턴을 우리가 쉽게 인지하기 어렵다.

3) 쓸모없는 특성을 학습하면서 노이즈가 섞일 수도 있다.

4) 큰 차원을 커버 할 만한 매우 많은 데이터가 수집되어야 한다.

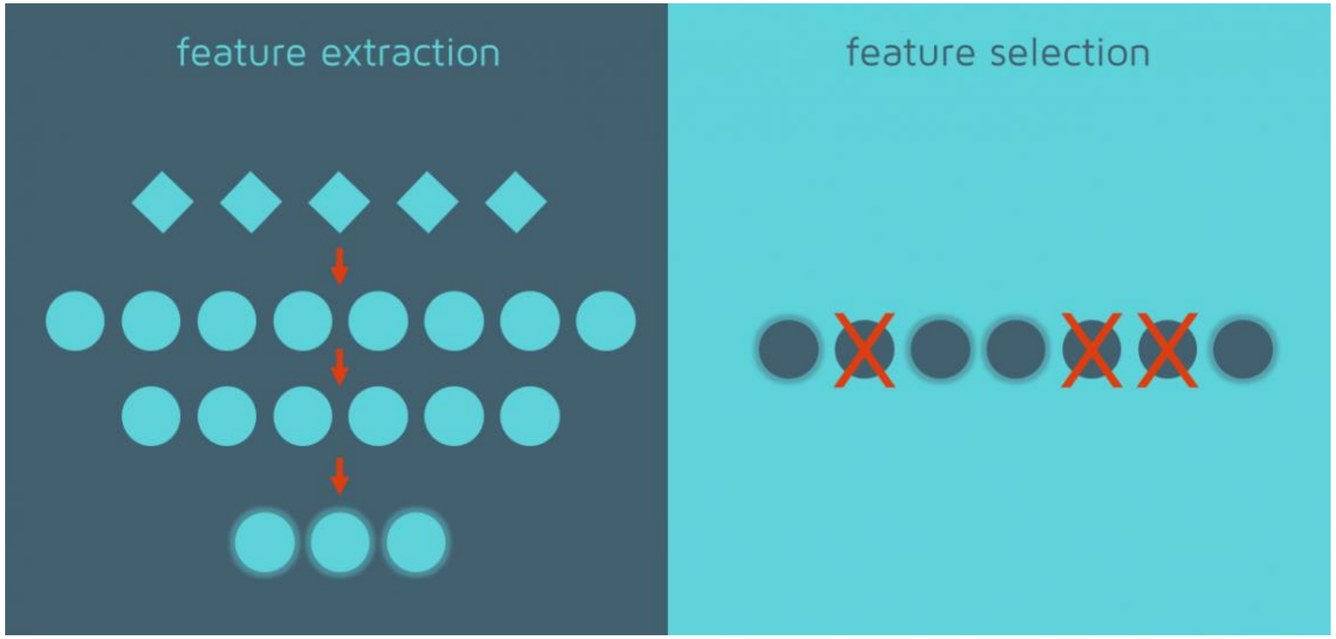
5) 몇몇 특성들끼리 강한 상관관계를 보이는 경우로 다중공선성 문제가 발생할 수 있다. 의존성이 높은 속성을 함께 학습하면 모델의 과적합이 발생해 학습성능이 저하된다.

=> 매우 많은 양의 데이터를 모으는 것은 어려우니까 어떤 속성이 모델의 성능 향상에 도움이 될지 파악하고 속성을 선택(가공)하는 차원축소 과정을 거치자!

1. 차원의 저주

1. 피처 선택 : 데이터의 특징을 잘 나타내는 주요 속성만 선택 상관계수 값으로 판단

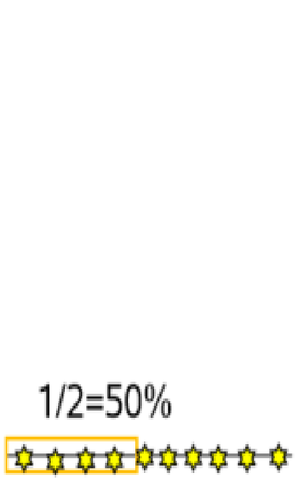
2. 피처 추출 : 기존 속성을 저차원의 중요 속성으로 압축해 추출
(기존 속성과는 완전히 다른 값이 된다. PCA, LDA,SVD,NMF와 같은 차원축소 방법을 통해 구현)

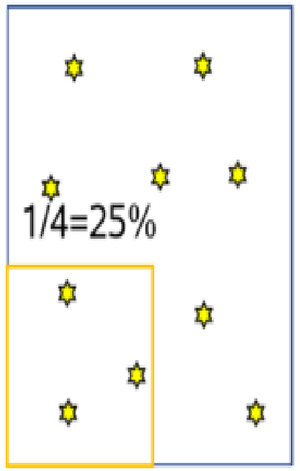


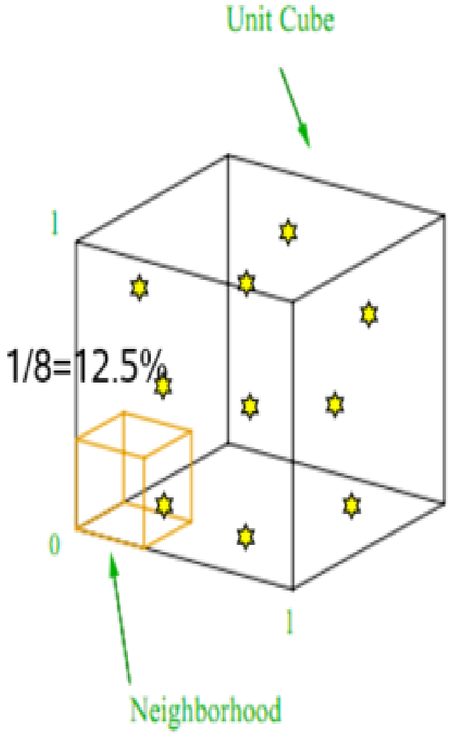
피처추출의 예시
: 학생 평가 요소로 모의고사 성적, 내신 성적, 수능성적, 봉사, 대외활동을 학업 성취도와 같은 함축적 요약 특성으로 추출 가능

1. 차원의 저주

차원이 늘어남에 따라 (=변수 x의 개수가 증가할수록) 같은 영역의 자료를 가지고 있더라도, 전체 영역대비 설명할 수 있는 데이터의 패턴은 줄어들게 된다.
즉 대부분의 훈련 데이터가 서로 멀리 떨어져 있고 따라서 예측이 불안정해진다.







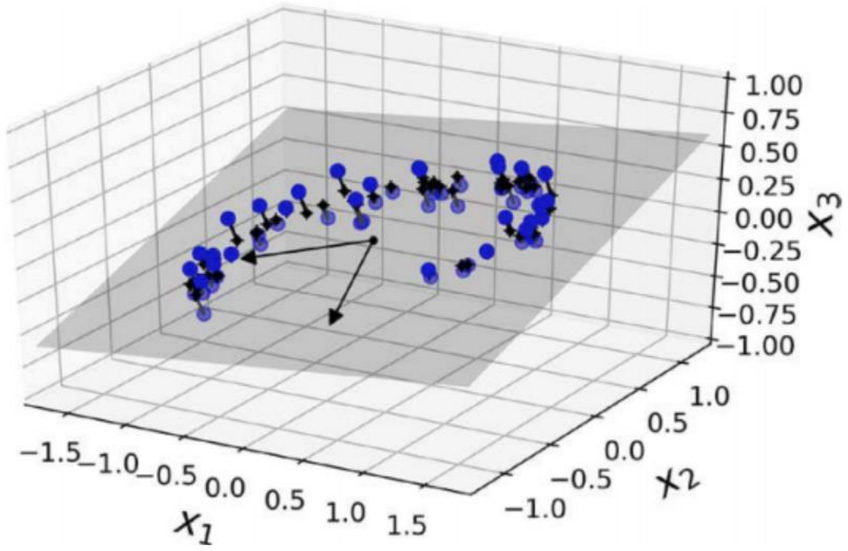
데이터에서 모델을 학습할 때 독립적인 샘플이 많을수록 학습이 잘 되는 반면,
차원이 커질수록 학습이 어려워지고 더 많은 데이터를 필요로 하는 현상을 말한다.
즉, 관측치 수 < 변수의 수 일 때 발생하는 현상이다.

EWHA EURON

2. 차원축소를 위한 접근법

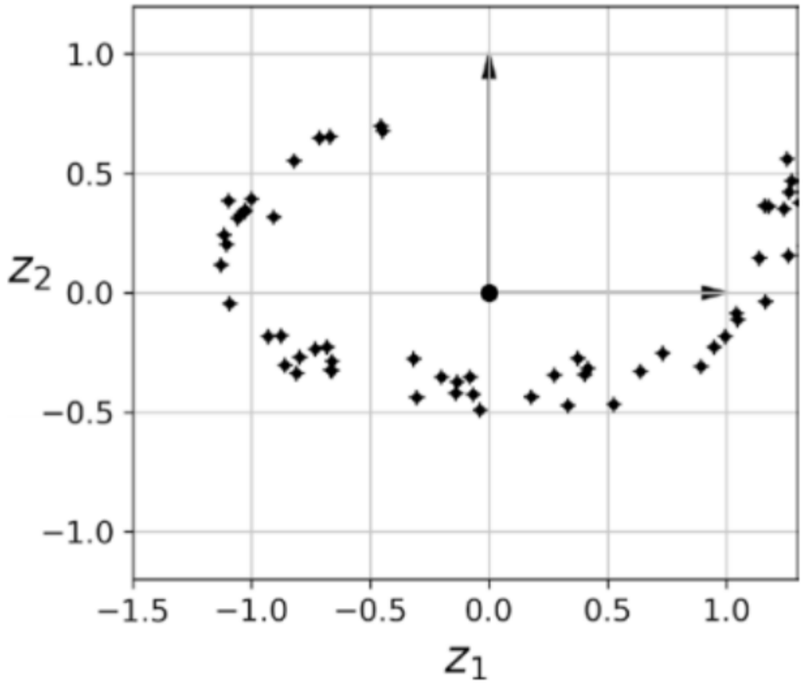
2. 차원축소 접근법

투영



=> 3차원 공간이지만, 대부분의 훈련 샘플은 회색 평면(2차원) 위에 위치한다.

따라서 3차원을 2차원으로 수직으로 정사영(투영) 시키면 오른쪽과 같은 2차원 데이터 셋을 얻을 수 있다.

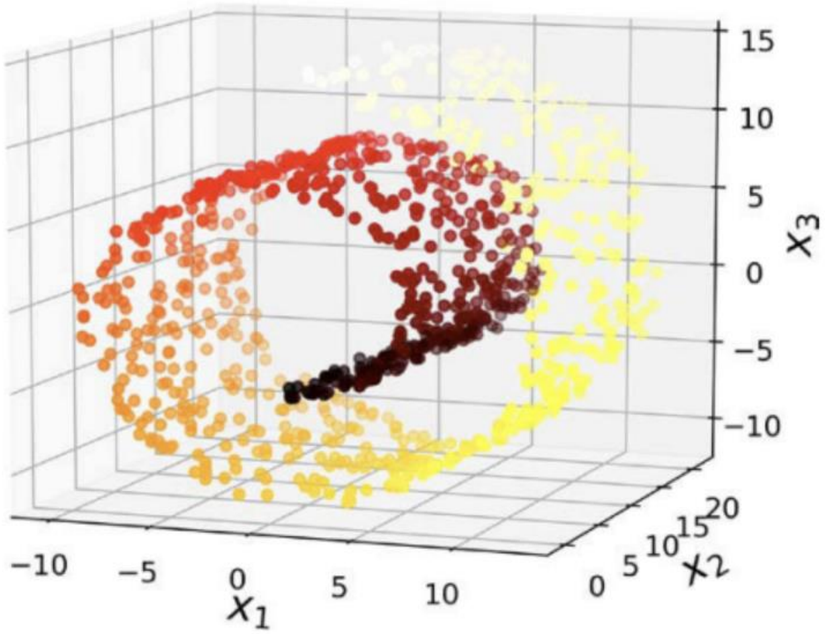


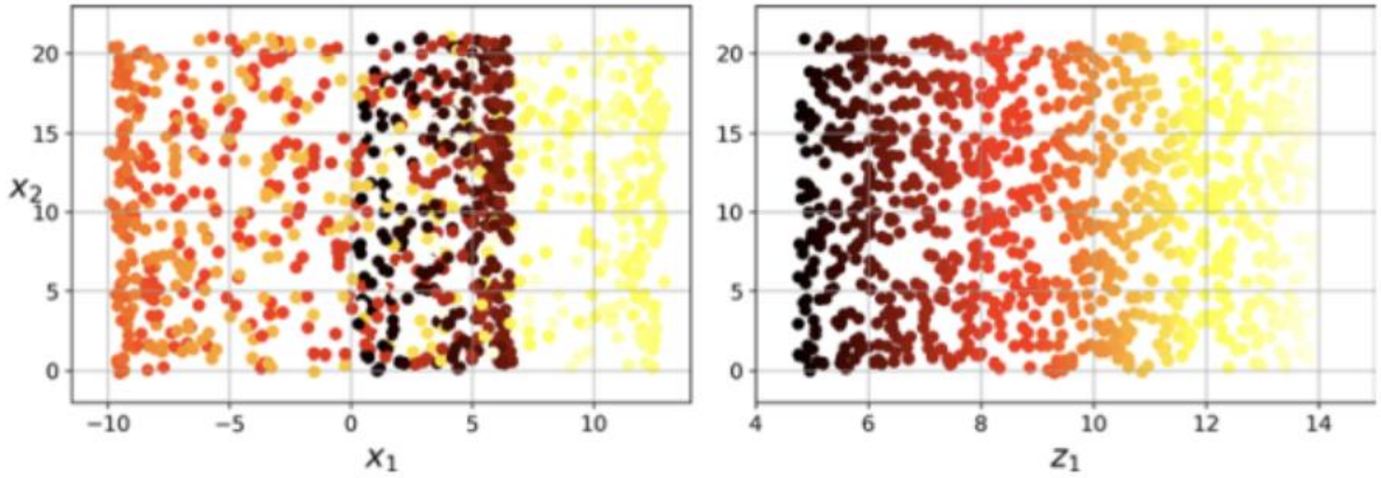
** 정사영 : 어떤 선이나 물체 위에서 빛이 비춰졌을 때 아래 평면에 그림자가 생기는 것

EWHA EURON

2. 차원축소 접근법

"스위스 롤" 형태의 데이터 셋은 부분 공간이 뒤틀려있기 때문에, 투영을 통해 차원을 축소시키는 방법은 좋지 못한 결과를 초래한다.



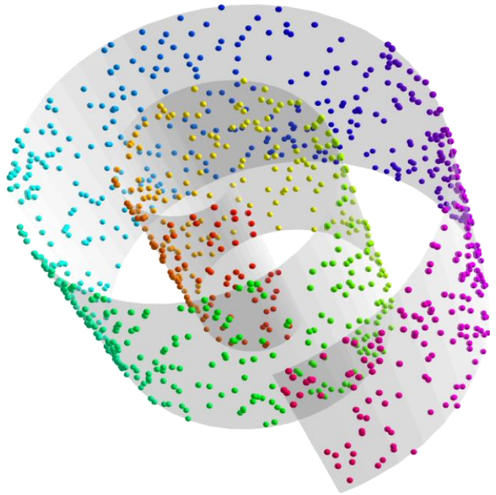


오른쪽처럼 스위스 롤을 펼친 형태의 2차원 데이터를 원했으나, 왼쪽처럼 뭉개진 데이터가 투영 결과로 나옴

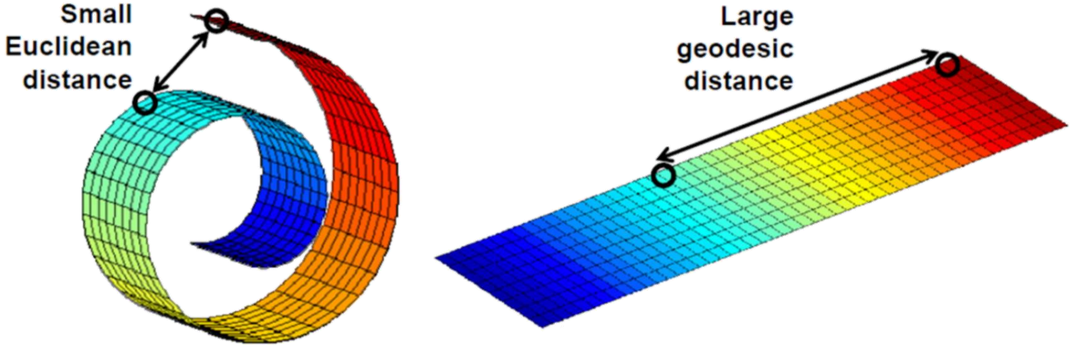
=> 투영이 항상 최선의 방법은 아니다!

2. 차원축소 접근법

매니폴드 학습



고차원 데이터를 데이터 공간에 뿌리면 샘플들을 잘 아우르는 subspace가 있을 것이라는 가정에서 시작
: 스위스 롤의 경우, 3차원 공간에 분포한 데이터를 아우르는 소용돌이 모양의 구부러진 평면인 2차원 매니폴드 중 하나.
이런 매니폴드를 찾아 2차원 평면에 데이터 포인트를 매핑하면 된다.
일반적으로 d차원 매니폴드는 전체 중 일부분이 d차원 초평면으로 보일 수 있는 n차원 공간의 일부이다.
많은 차원축소 알고리즘이 훈련샘플이 놓여있는 매니폴드를 모델링 하는 식으로 작동한다.

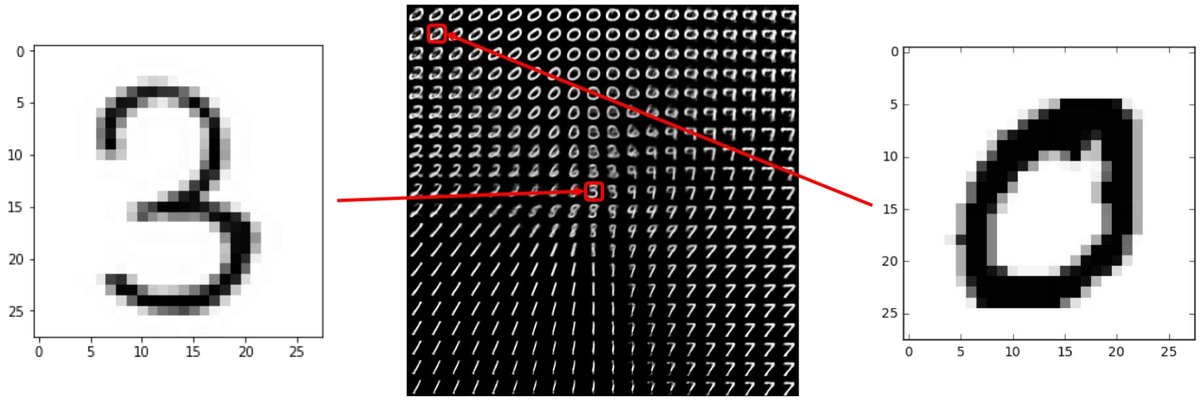


고차원상에서 가까운 거리에 있던 데이터 포인트들일지라도, 매니폴드를 보다 저차원 공간으로 맵핑하면 오히려 거리가 멀어질 수 있다. 그리고 저차원의 공간상에서 가까운 점끼리는 실제로도 비슷한 특징feature을 갖는다. 즉, 저차원의 각 공간의 차원 축은 고차원에서 비선형적으로 표현될 것이며, 데이터의 특징을 각각 표현하게 된다.

2. 차원축소 접근법

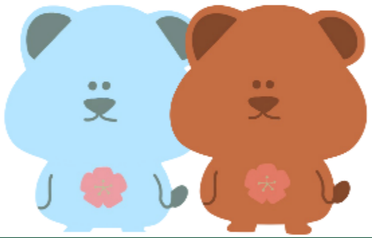
매니폴드 학습은 딥러닝에서 훌륭한 성능을 내는데, 매니폴드는 비선형적인 방식으로 차원축소를 수행하기 때문에, 복잡한 데이터셋을 다루는 딥러닝은 매니폴드를 자연스럽게 찾아낼 수 있다.

예를 들어, 다음 그림과 같이 MNIST 데이터를 2차원의 숨겨진 저차원에 표현한다고 가정하자. 빨간색으로 표시된 각 샘플은 2차원 공간에서는 사람이 인지하는 특징과 비슷한 특징을 갖는 위치와 관계에 있겠지만, 원래의 데이터 차원인 784차원의 고차원 공간에서는 전혀 다른 거리와 관계를 지닐 것이다.



https://kh-kim.gitbook.io/natural-language-processing-with-pytorch/00-cover-5/02-dimension-reduction

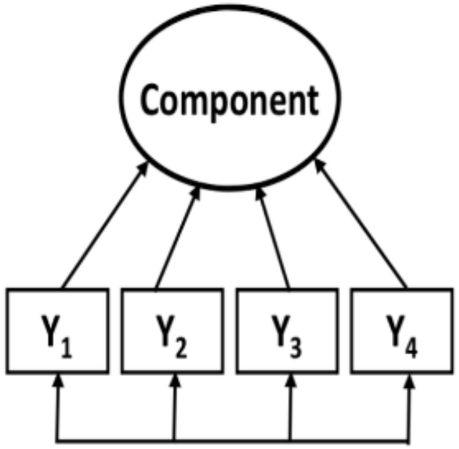
3. PCA

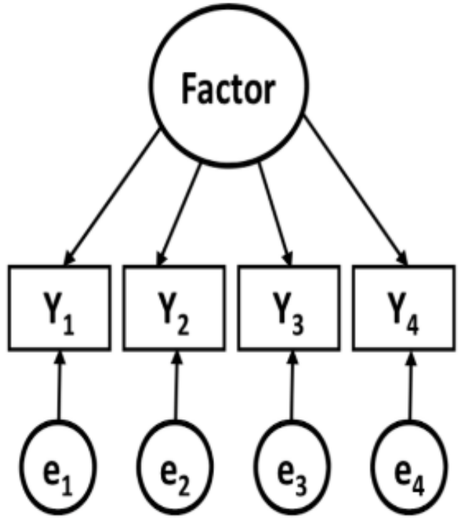


EWHA EURON

3 PCA

- 차원축소에는 **주성분 분석(PCA**, Pricipal Component Analysis)과 **요인분석**(Factor Analysis)이 있음
- **요인분석**의 목적은 데이터의 여러 변수들이 가지고 있는 공분산 구조를 밝히는 것. 데이터에 대한 가정을 내리고 그를 검증하는 통계 모델에 사용 (**주로 사회과학 논문**)
- **주성분 분석**은 구조를 모델링한다기보다는 자료를 파악하거나 이후 진행될 작업의 성능과 효율을 높이기 위해, 주어진 데이터를 최대한 보존하는 저차원의 데이터를 얻는 기법 (**데이터 분석**)





(좌) PCA 도식 (우) 요인분석 도식 - *대충 비슷하게 생겼는데?*

3 PCA

PCA : 데이터에 가장 가까운 **초평면**을 정의한 다음 데이터를 평면에 투영함

투영시킬 **올바른 초평면** 선택하기

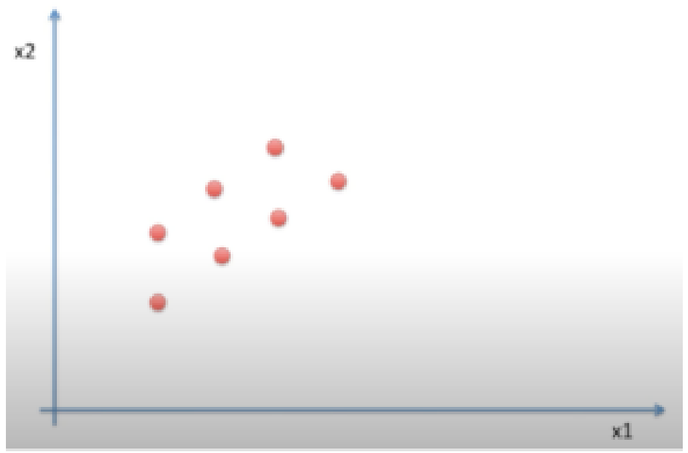
정보가 적게 손실되는(=분산이 최대로 보존되는) 축을 선택하기

- 초평면 : 평면에서 더 확장된 개념으로, 직선은 2차원의 초평면, 평면은 3차원의 초평면 이라고 한다. 즉 n차원에 그려진 초평면은 n-1 차원 공간과 같다. 즉 그려진 공간의 차원보다 한 차원 낮은 공간을 의미한다. 초평면은 주로 공간을 분할하는 역할에 쓰이며, (직선이 평면을 분할하는 것처럼) 이런 원리는 분류 머신러닝 모델을 만들 때 사용된다.
-

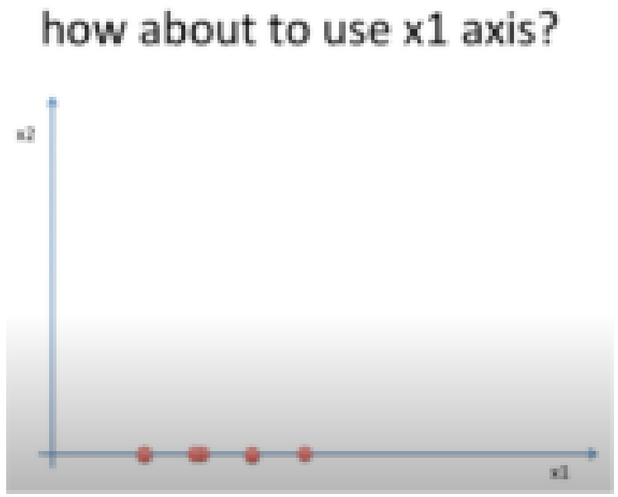
3 PCA

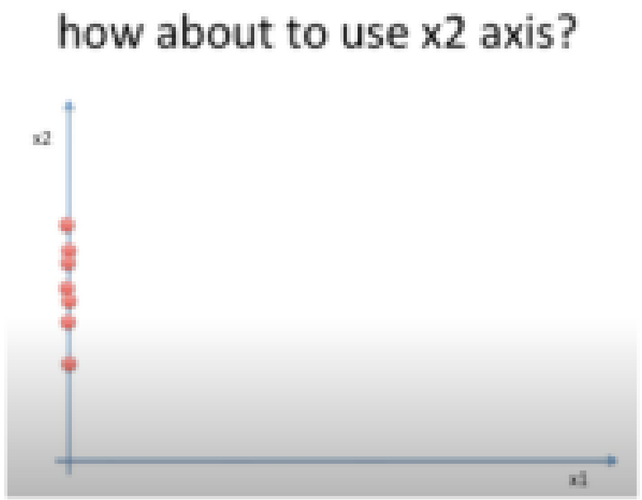
쉽게 말하자면…

목표 0. 2차원 공간의 데이터를 1차원 공간의 데이터로 만들어주기

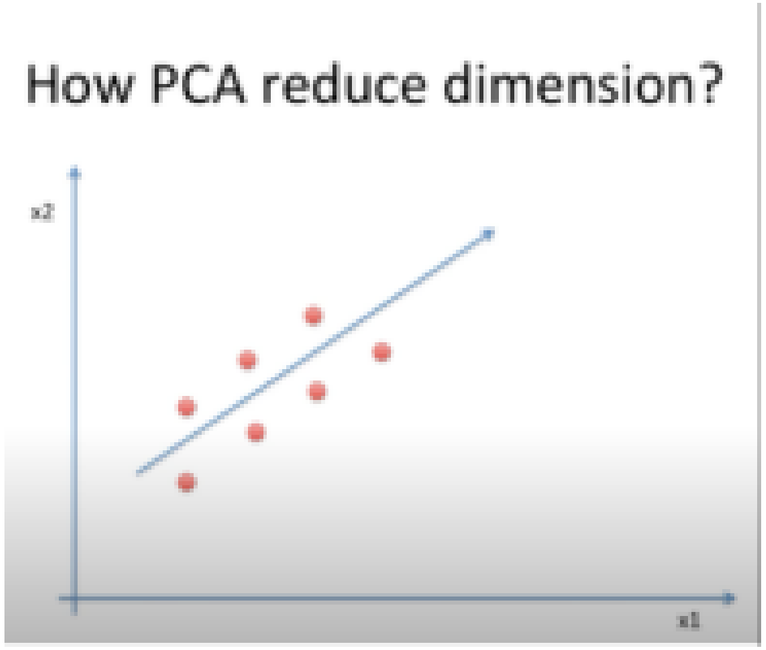


방법 1. 그냥 한 축으로 내려버리기





방법 2. 새로운 차원찾기

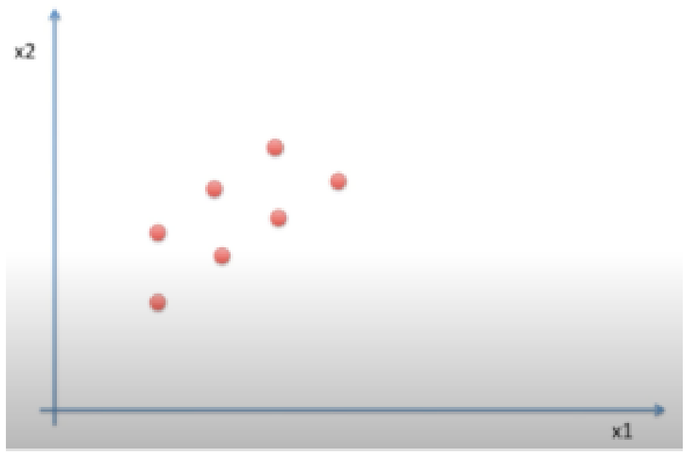


-> 새로운 차원인 화살표에 데이터를 내려주면, 데이터들이 겹치지 않고 새로운 축에 잘 반영됨

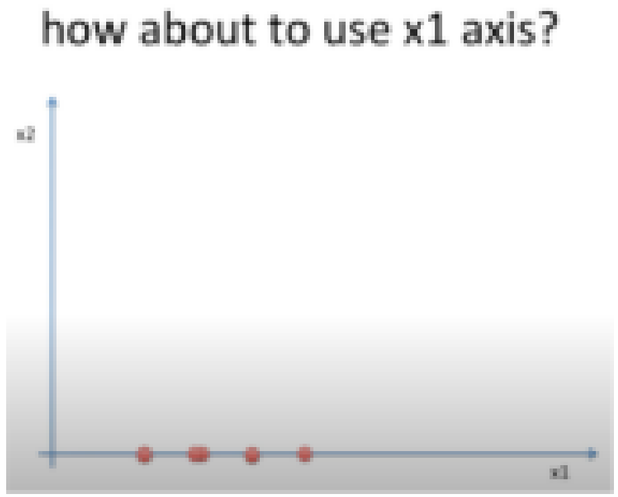
3 PCA

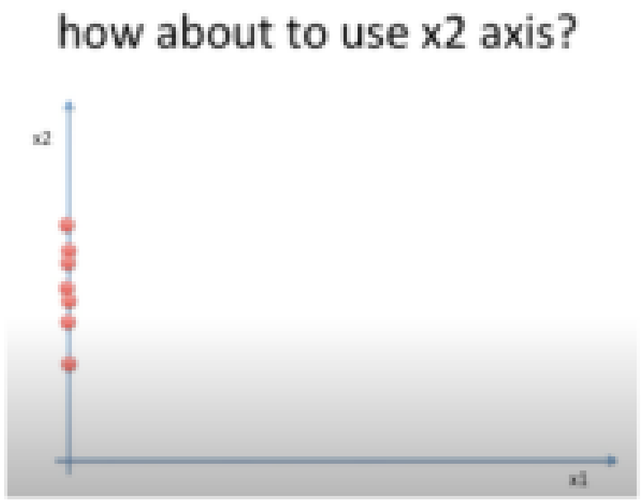
쉽게 말하자면…

목표 0. 2차원 공간의 데이터를 1차원 공간의 데이터로 만들어주기

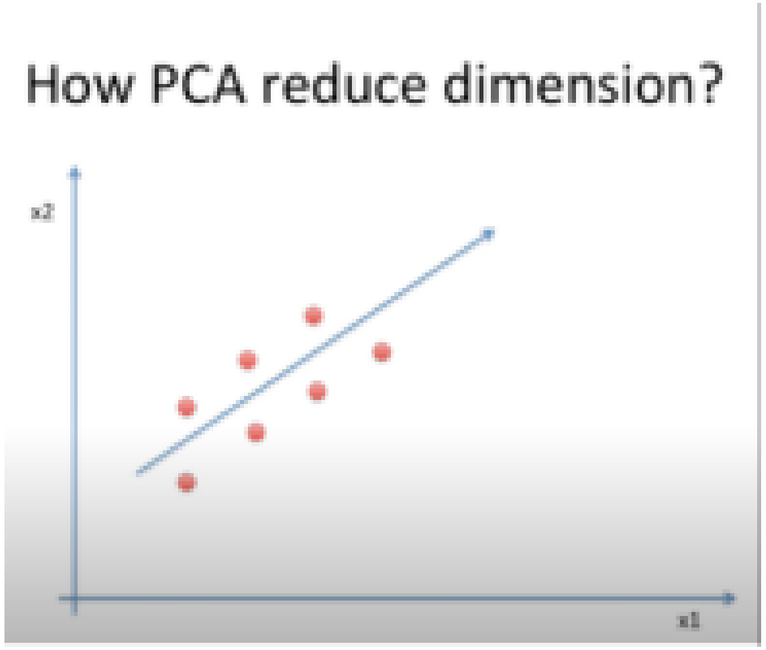


방법 1. 그냥 한 축으로 내려버리기



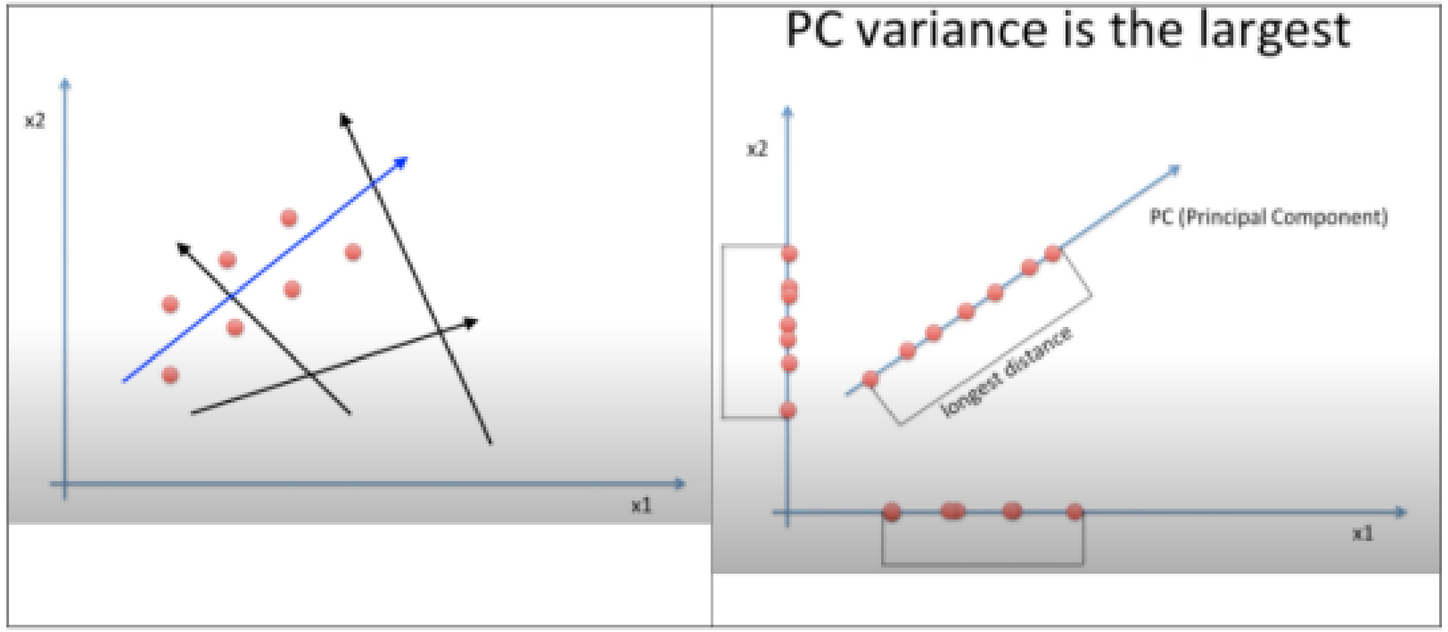


방법 2. 새로운 차원찾기



-> 새로운 차원인 화살표에 데이터를 내려주면, 데이터들이 겹치지 않고 새로운 축에 잘 반영됨

3 PCA



→ 무수히 다양한 화살표를 그릴 수 있는데, 그중 파란색 화살표처럼 데이터를 해당 화살표에 1차원으로 내렸을 때, 데이터들이 서로 겹치지 않게 하는 화살표를 찾아야 한다. 데이터가 최대한 안 겹치려면 데이터들끼리 멀리 퍼지게 해야 하고, 즉 차원을 내렸을 때, 데이터의 분산이 커야 하고 이는 화살표의 길이가 길어야 함을 의미한다.

→ 지금 예시는 2차원을 1차원으로 내리는 거지만, 3차원 이상의 경우를 저차원으로 내리는 경우엔, 만든 화살표에 직각이 되는 또 다른 축(화살표)를 찾아 나갈 수 있다.

→ IN 선형대수학

공분산 행렬에서 고유벡터, 고유값을 구하고 가장 분산이 큰 방향을 가진 고유벡터 e1 에 입력데이터를 선형변환하고, 그 다음으로 e1과 직교하며 e1 다음으로 분산이 큰 e2 고유벡터에 또 선형변환하고……

EWHA EURON

3 PCA

공분산 행렬

두 속성 간의 변동. 속성 쌍들의 변동(데이터의 분포형태) 이 얼마나 닮았는가(변수 간 상관정도)

대칭행렬

고유벡터를 직교행렬로
고유값을 정방행렬로
대각화 (행렬을 쪼갬)

$$A = V \Lambda V^T$$

* 행렬 x 벡터 -> 벡터 : 행렬은 하나의 벡터공간을 선형적으로 다른 벡터공간으로 매핑하는 기능을 가진다.

고유벡터 (화살표=새로운 축)

: 행렬X에 다른 행렬A을 곱했을 때, 행렬 X의 벡터들 중 벡터의 크기는 변하지만, 방향은 변하지 않는 벡터. 공분산 행렬의 고유벡터는 데이터가 어떤 방향으로 분산되었는지를 나타내 준다.

고유값 (화살표 길이)

: 해당 고유 벡터의 크기. 고유벡터 방향으로 얼마만큼의 크기로 벡터 공간이 늘려지는지 이야기 한다. 고유값이 큰 순서대로 (가장 길이가 긴 화살표) 고유벡터를 정렬하면 중요한 순서대로 주성분(새로운 축)을 구하는 것이 된다.

3 PCA

[정리]

0. 평균이 0이 되도록 데이터들을 정규화 해준다. 현재 평균을 구해 각 데이터에 평균만큼 빼면 전체 평균이 0이 되도록 만들 수 있다.
1. 입력 데이터의 공분산 행렬을 구한다.
2. 공분산 행렬을 고유 값 분해 해서 고유 벡터와 고유 값을 구한다 (앞서 말한 최고의 화살표찾기)
3. 고유 값이 가장 큰 (화살표 길이가 가장 긴) k개의 고유 벡터를 추출
4. 고유 값이 가장 큰 순으로 추출된 고유벡터를 이용해 입력 데이터들을 선형 변환 (저차원으로 매핑)

PCA는 N개의 차원에서 N개의 주요 성분(새로운 축=PC)이 나오게 된다. 이 주요성분 중 데이터들을 잘 표현하고 있는 주요 성분만 선택하여 사용한다. 여기서 PC(새로운 축)는 기존의 속성을 가지고 새롭게 정의된 성분임을 잊지 말아야 한다.

★ 고유 값의 크기가 PC의 중요도! 고유값의 크기가 가장 큰 것이 고유벡터를 선택하는 기준이 됨

3 PCA

공분산 행렬 : 두 속성 간의 변동. 속성 쌍들의 변동이 얼마나 닮았는가(변수 간 상관정도)를 행렬형태로 보여줌

대칭행렬

고유벡터를 직교행렬로
고유값을 정방행렬로 대각화 할 수 있음
(-> SVD 특이값 분해=행렬을 대각화 하는 방법)

$$A = V \Lambda V^T$$

공분산행렬은 대칭행렬이기 때문에 고유값 분해를 했을 때, 고유백터의 직교 행렬(V) * 고유값의 정방행렬 ∧ * 고유 벡터의 직교 행렬의 전치행렬(VT) 로 표현된다. 이때 v1 은 가장 분산이 큰 방향을 가진 고유벡터이며, v2는 v1 에 직교하면서, 다음으로 가장 분산이 큰 방향을 가진 고유벡터가 된다. 즉 고유벡터들이 분산이 큰 화살표들인 것임!

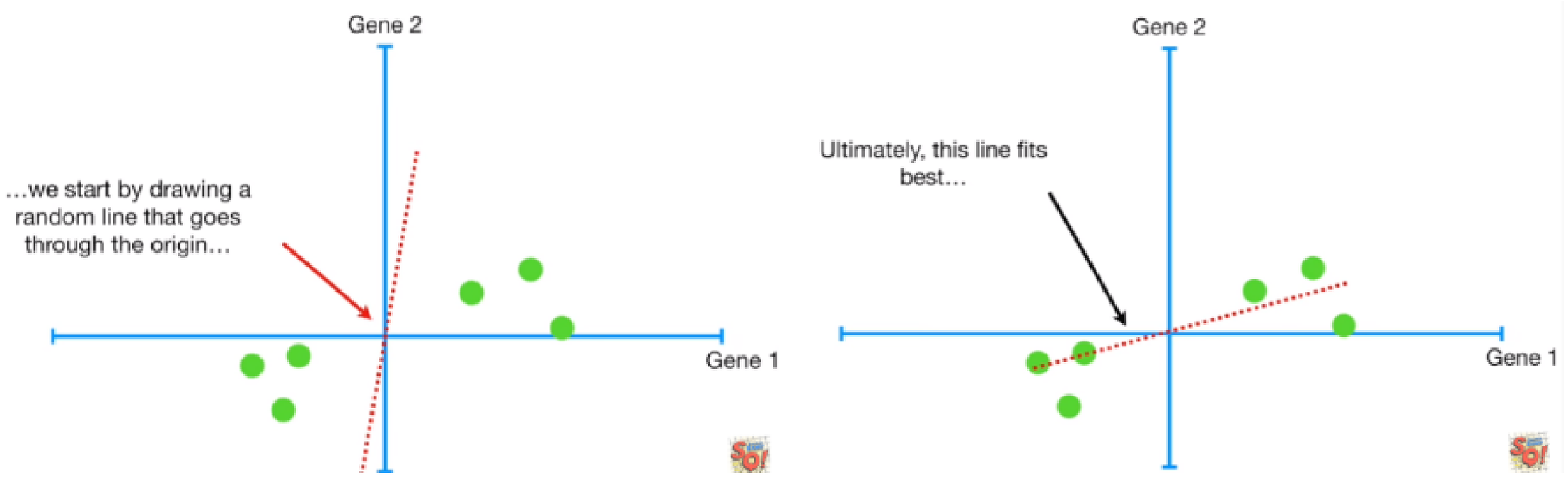
$$A = V \Lambda V^T$$

$$= \begin{bmatrix} v_1 & v_2 & \cdots & v_N \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_N^T \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 v_1 & \lambda_2 v_2 & \cdots & \lambda_N v_N \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_N^T \end{bmatrix}$$

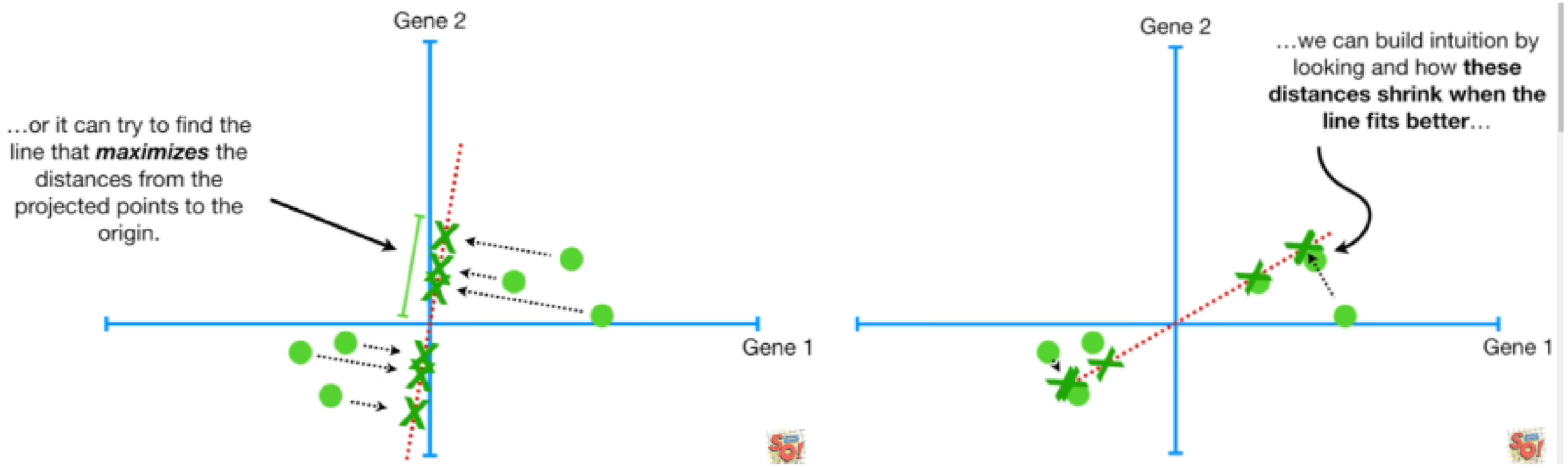
3 PCA

왜 분산이 큰 화살표를 사용해야하는지 그림으로 이해해보자!



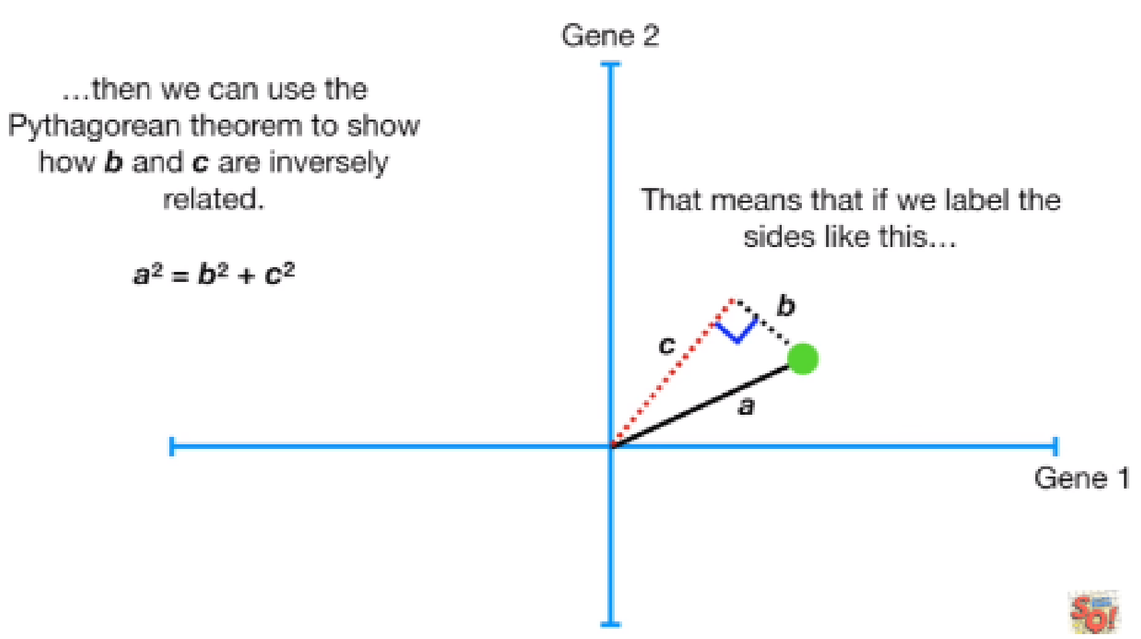
3 PCA

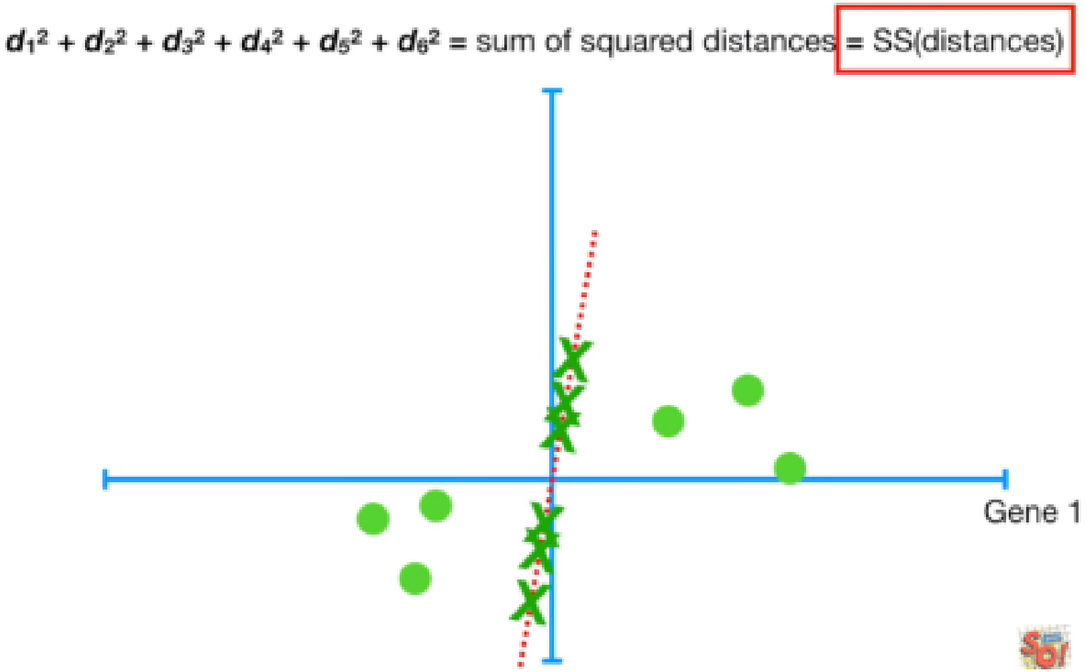
왜 분산이 큰 화살표를 사용해야하는지 그림으로 이해해보자!



3 PCA

왜 분산이 큰 화살표를 사용해야하는지 그림으로 이해해보자!





3 PCA

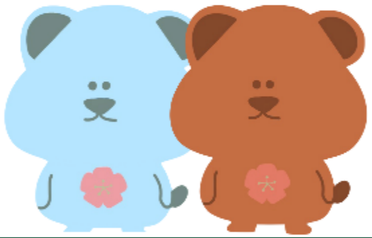
코드

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

#fit( )과 transform( ) 을 호출하여 PCA 변환 데이터 반환
pca.fit(iris_scaled)
iris_pca = pca.transform(iris_scaled)
print(iris_pca.shape)
```

```
(150, 2)
```

n_componen는 PCA로 변환할 차원의 수. 여기서는 2로 설정

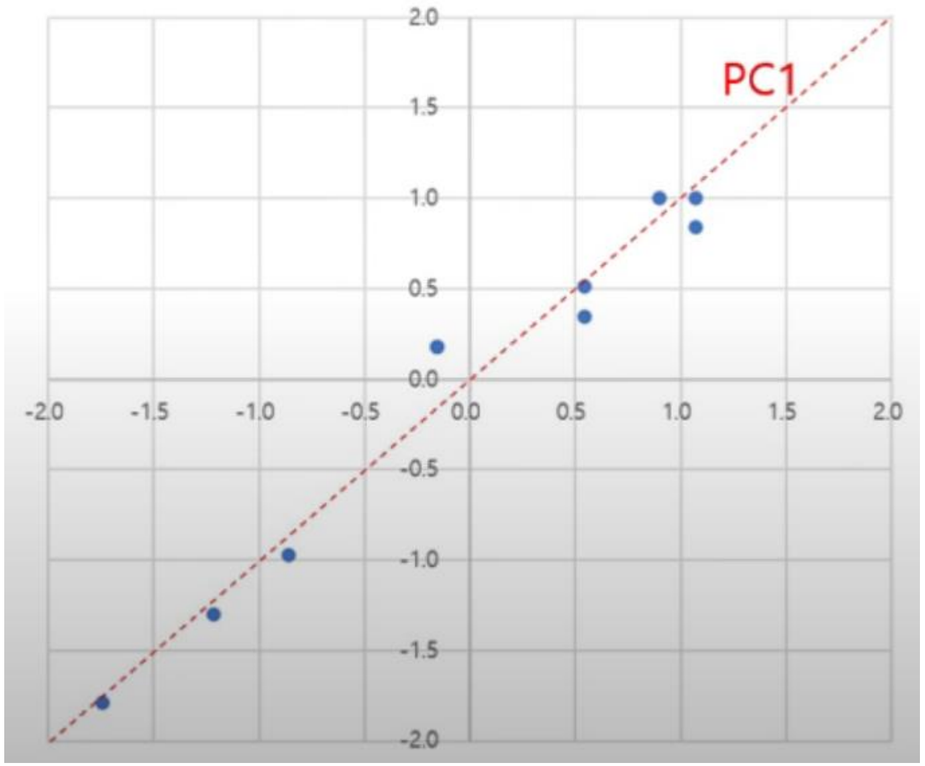
4. Explained Variance Ratio



4.1 Explained Variance Ratio

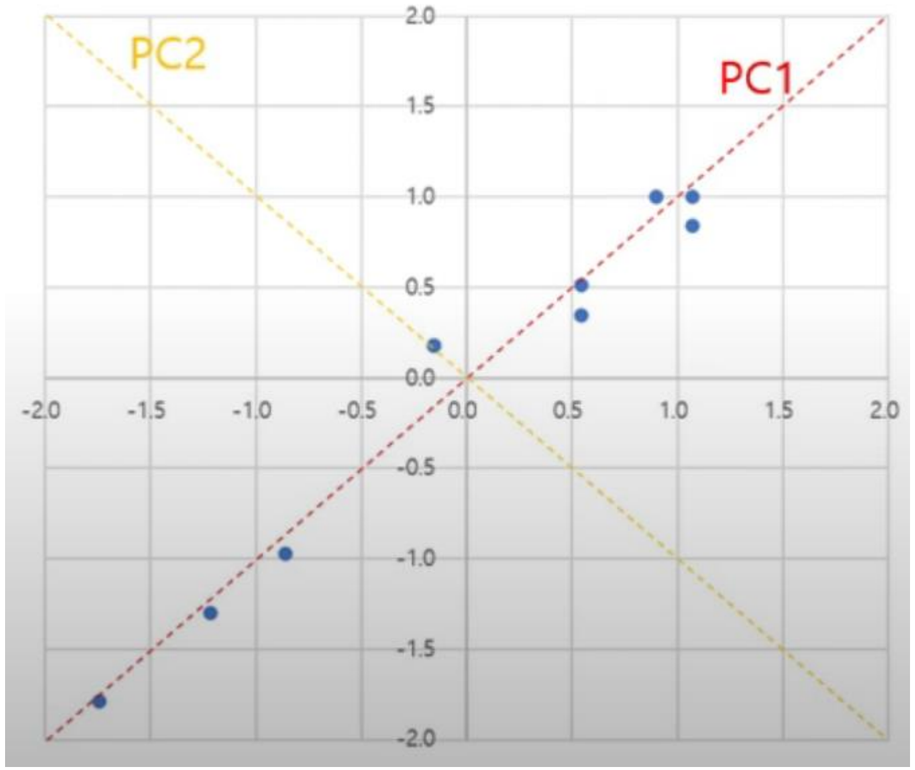
고유값이 1.98과 0.02가 나왔으면 …

1. 고유값이 1.98일 때



1.98이 설명하는 분산의 양 = 1.98/2(분산합) = 99%

2. 고유값이 0.02일 때



0.02이 설명하는 분산의 양 = 0.02/2(분산합) = 1%

EWHA EURON

4.1 Explained Variance Ratio

expained_variance_ratio_ 변수에는 원본 데이터셋에 대해 **PC가 보존하는 분산의 비율**이 들어있음.
다음은 가장 높게 보존하는 순으로 두가지 PC의 expained_variance_ratio_ 를 살펴보는 코드
(아래는 iris 데이터 PCA의 예시)

```
pca.explained_variance_ratio_
```

```
array([0.72962445, 0.22850762])
```

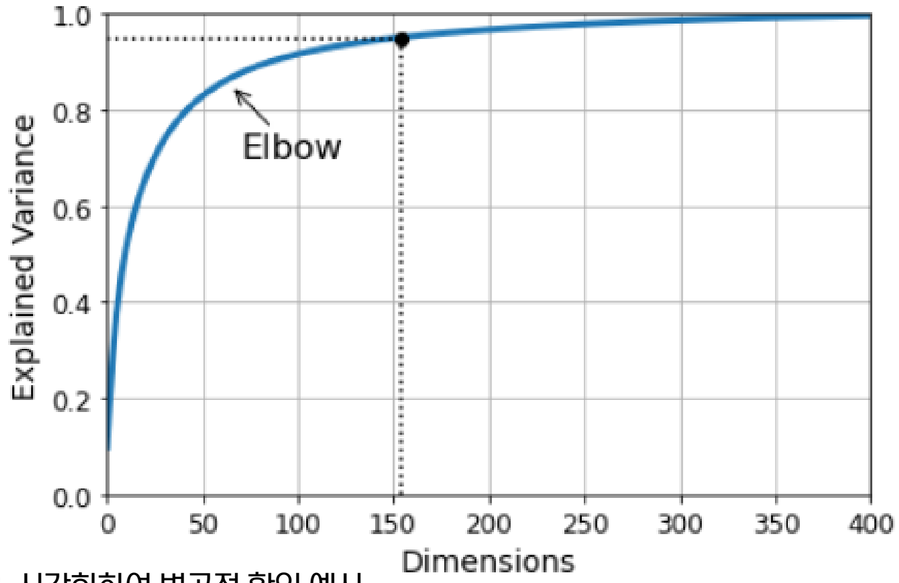
2개의 주성분으로 전체분산의 약 94% 설명가능!

4.2 적절한 차원 수 선택

축소할 차원 수를 정하는 방법에는 **각 PC별로 표현하는 데이터 분산의 합이 충분할 때까지 해야 하는데** 2가지 방법이 있음.

1. 데이터셋의 분산을 95%로 유지하는데 필요한 최소한의 PC 개수 즉 차원 수를 계산
2. 시각화하여 변곡점 확인

* 데이터 시각화를 위해 차원을 축소하는 경우는 보통 2, 3차원을 사용



2. 시각화하여 변곡점 확인 예시

```
import numpy as np

cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

```
pca = PCA(n_components=0.95)
iris_scaled_reduced = pca.fit_transform(iris_scaled)
```

```
iris_scaled_reduced = pd.DataFrame(iris_scaled_reduced)
iris_scaled_reduced
```

	0	1
0	-2.264703	0.480027
1	-2.080961	-0.674134
2	-2.364229	-0.341908
3	-2.299384	-0.597395
4	-2.389842	0.646835
...	...	...

1. 데이터셋의 분산을 95%로 유지하는데 필요한 최소한의 PC 개수 즉 차원 수를 계산한 예시

5. 일반 vs 점진적 vs 커널 PCA

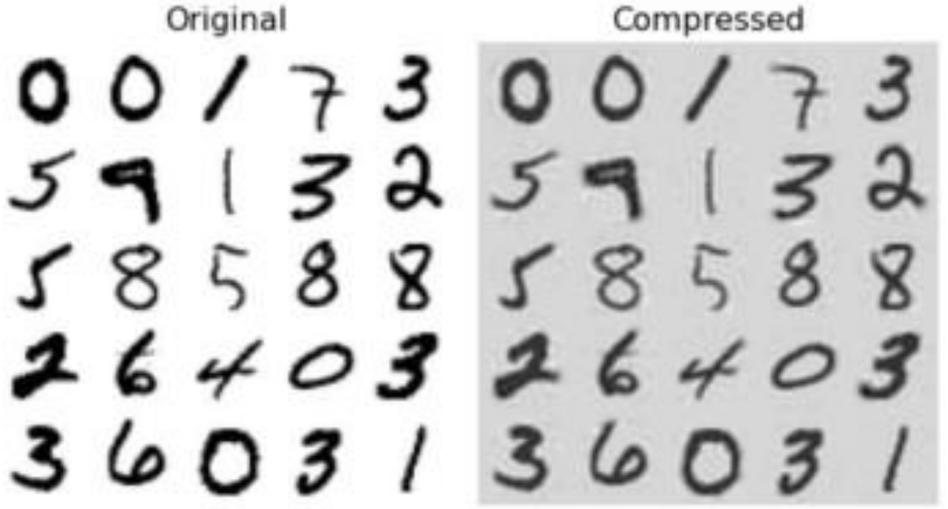
5.1 압축을 위한 PCA

- **차원 축소**를 하면 훈련 세트의 크기가 줄어들어, **분류 알고리즘의 속도를 크게 높일 수 있다.**
- 또한 반대로 **압축된 데이터 셋에 PCA 투영 반환을 적용시켜, 원본 데이터와 유사하게 만들 수 있다.**
  - inverse_transform( ) **메소드**를 사용하면 된다.
  - 단, **원본 데이터와 똑같이 복구시킬 수는 없다!!**

```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}} \mathbf{W}_d^T$$

다음은 MNIST 데이터셋에 대하여 원본 데이터셋과 압축 후 복원된 결과를 비교한 그림입니다. 이미지의 품질이 손상되긴 했지만 숫자의 모양은 온전한 것을 확인할 수 있습니다.



5.2 랜덤 PCA

svd_solver = "randomized"로 지정하면 sklearn은 **랜덤 PCA**라는 확률적 알고리즘을 이용하여 **축소할 d차원에 대한 d개의 PC를 '근삿값'으로 빠르게 찾습니다.**

```
rnd_pca = PCA(n_components=154, svd_solver="randomized",
random_state=42)
X_reduced = rnd_pca.fit_transform(X_train)
```

svd_solver의 기본값은 **"auto"**인데, 원본 데이터의 크기나 차원 수가 500보다 크고, 축소할 차원이 이것들의 80%보다 작으면 sklearn은 자동으로 랜덤 PCA 알고리즘을 사용합니다. 만약 이것을 방지하고 싶다면 **"full"**을 사용하면 됩니다.

5.3 점진적 PCA

PCA 구현의 문제는 SVD 알고리즘 실행을 위해 전체 데이터셋을 메모리에 올려야 한다는 점입니다. **점진적 PCA(IPCA : incremental PCA)는 dataset을 mini-batch로 나눈 뒤 하나 씩 주입하여 적용하여 이를 보완합니다.**

다음 코드는 MNIST 데이터셋을 100개의 mini-batch로 나눠 차원을 축소하는 코드입니다.

```
from sklearn.decomposition import IncrementalPCA

n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
    print(".", end="") # 책에는 없음
    inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

5.4 커널 PCA

커널 트릭

샘플을 매우 높은 고차원 공간(특성 공간)으로 암묵적으로 매핑하여 서포트 벡터 머신의 비선형 분류와 회귀를 가능하게 하는 수학적 기법

-> 고차원 특성 공간에서의 선형 결정 경계는 원본 공간에서는 복잡한 비선형 결정 경계에 해당함

커널 PCA

같은 기법을 PCA에 적용해 차원 축소를 위한 복잡한 비선형 투형 수행 가능

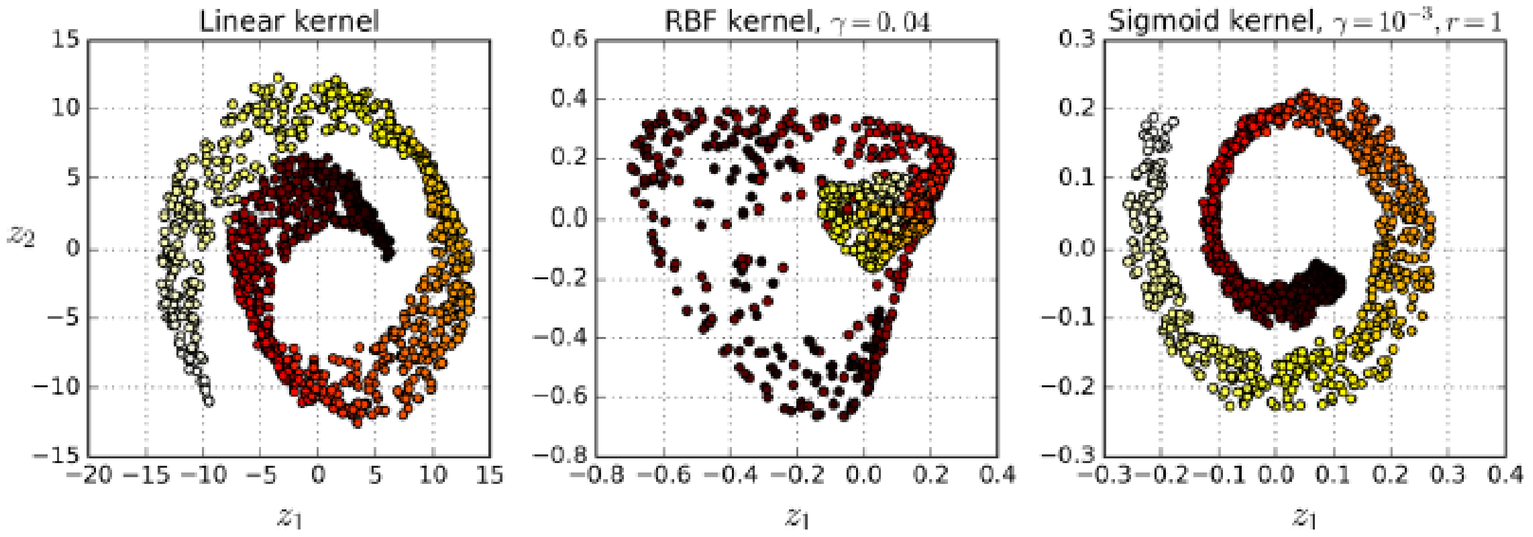
- 투영된 후 샘플의 군집을 유지하거나 꼬인 매니폴드에 가까운 데이터셋을 펼칠 때도 유용

5.4 커널 PCA

KernelPCA를 사용해 RBF 커널로 kPCA 적용

```
from sklearn.decomposition import KernelPCA

rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

선형 커널, RBF 커널, 시그모이드 커널을 사용하여 2차원으로 축소시킨 스위스 롤



여러가지 커널의 kPCA를 사용해 2D로 축소시킨 스위스 롤

5.4 커널 PCA

- 커널 선택과 하이퍼파라미터 튜닝

kPCA는 비지도 학습이지만 종종 지도학습의 전처리 단계로 활용되므로, 그리드 탐색을 사용하여 주어진 문제에서 성능이 가장 좋은 커널과 하이퍼파라미터를 선택할 수 있음

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression(solver="lbfgs"))
    ])

param_grid = [{
        "kpca__gamma": np.linspace(0.03, 0.05, 10),
        "kpca__kernel": ["rbf", "sigmoid"]
    }]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('kpca', KernelPCA(n_components=2)),
                                       ('log_reg', LogisticRegression())]),
             param_grid=[{'kpca__gamma': array([0.03      , 0.03222222, 0.03444444, 0.036666
67, 0.03888889,
       0.04111111, 0.04333333, 0.04555556, 0.04777778, 0.05      ]),
                          'kpca__kernel': ['rbf', 'sigmoid']}])
```

가장 좋은 커널과 하이퍼파라미터 확인

```
print(grid_search.best_params_)
```

```
{'kpca__gamma': 0.043333333333333335, 'kpca__kernel': 'rbf'}
```

06. LDA(Linear Discriminant Analysis)

EWHA EURON

6.1 LDA(Linear Discriminant Analysis)

LDA(Linear Discriminant Analysis)

: 입력 데이터 세트를 저차원 공간으로 투영해 차원을 축소하는 기법으로,

개별 클래스를 분류할 수 있는 기준을 최대한 유지하면서 차원을 축소하기 때문에 지도학습에서 사용됨.

(+ 분류 알고리즘 중 하나)

PCA?

비지도 학습에서 사용됨

6.1 LDA(Linear Discriminant Analysis)

LDA는 특정 공간상에서 클래스 분리를 최대화하는 축을 찾기 위해

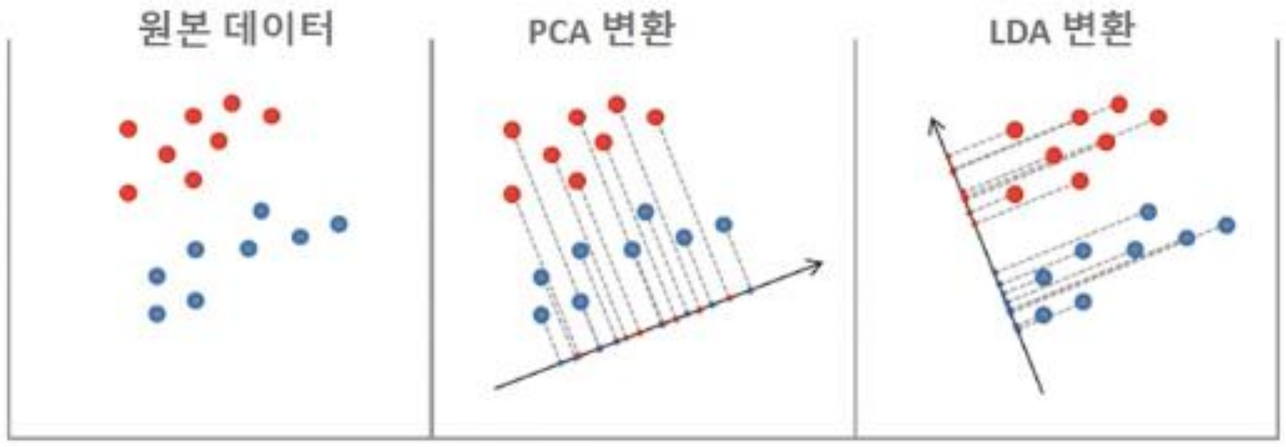
클래스 간 분산(between-class scatter)과 클래스 내부 분산(within-class scatter) 의 비율을 최대화하는 방식으로 차원을 축소함.

✓ 클래스 내의 분산은 최소가 되도록

✓ 클래스 간 분산은 최대가 되도록

➔ LDA를 통해 하나의 축으로 transformation(변형)된 데이터들이

같은 클래스 내에는 그 값의 차가 최소가 되도록 하며, 다른 클래스끼리는 그 값이 차가 크게 하는 축을 찾는 것



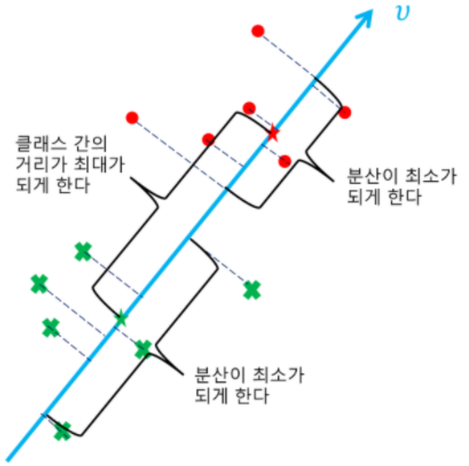


그림 1. LDA는 클래스 간의 거리가 최대가 되게 하면서, 동시에 클래스 내의 분산이 최소가 되게 하는 벡터를 찾는다.

6.1 LDA(Linear Discriminant Analysis)

✓ LDA 내부 동작 방식

1. d차원 데이터셋을 표준화한다. (d = 특성개수)
2. 각 클래스 대해 d차원 평균 벡터를 계산한다.
3. 클래스 간의 산포 행렬(scatter matrix) S(B)와
   클래스 내부의 산포행렬 S(W)를 구성한다.
4. S(W)의 역행렬과 S(B)의 곱행렬의 고유벡터와 고윳값을 계산한다.
5. 고윳값을 내림차순으로 정렬해 순서를 정한다.
6. 고윳값이 가장 큰 k개의 고유벡터를 선택해 d * k 차원의 변환행렬 W를 구한다.
7. 변환 행렬 W를 사용해 새로운 특성 부분 공간으로 투영한다.

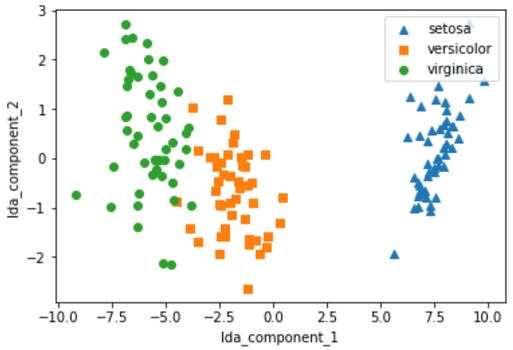
6.2 붓꽃 데이터 세트에 LDA 적용

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

iris = load_iris()
iris_scaled = StandardScaler().fit_transform(iris.data)  # 표준 정규 분포로 스케일링
```

LDA 수행시키기 전에 스케일링 해줘야 함.

```python
# 2개의 클래스로 구분하기 위한 LDA 생성
lda = LinearDiscriminantAnalysis(n_components=2)

# 지도학습 분류이므로 fit할 때, target값을 입력해주어야 함.
lda.fit(iris_scaled, iris.target)
iris_lda = lda.transform(iris_scaled)
print(iris_lda.shape)
```

```python
# LDA 변환된 입력 데이터 값을 2차원 평면에 품종별로 표현.

import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

lda_columns=['lda_component_1','lda_component_2']
irisDF_lda = pd.DataFrame(iris_lda,columns=lda_columns)
irisDF_lda['target']=iris.target

#setosa는 세모, versicolor는 네모, virginica는 동그라미로 표현
markers=['^', 's', 'o']

#setosa의 target 값은 0, versicolor는 1, virginica는 2. 각 target 별로 다른 shape으로 sca
tter plot
for i, marker in enumerate(markers):
    x_axis_data = irisDF_lda[irisDF_lda['target']==i]['lda_component_1']
    y_axis_data = irisDF_lda[irisDF_lda['target']==i]['lda_component_2']

    plt.scatter(x_axis_data, y_axis_data, marker=marker,label=iris.target_names[i])

plt.legend(loc='upper right')
plt.xlabel('lda_component_1')
plt.ylabel('lda_component_2')
plt.show()
```



→ LDA: 클래스를 나누는 것에 초점

6.3 PCA vs. LDA

PCA vs. LDA

PCA	LDA
투영 사용	
비지도 학습	지도 학습 (다른 클래스에 속하는 데이터들을 분류 가능하도록 데이터 포인트들을 투영시키는 머신러닝 알고리즘)
투영들의 분산이 큰 벡터들을 찾음 → 투영들의 분산이 적게 만드는 벡터 제거	데이터를 하나의 선으로 투영

➔ PCA와 LDA 거의 유사하지만,
LDA가 분류 문제에 최적화 되어 있는 차원 축소 방법이라고 생각하면 됨!

07.SVD(특이값 분해)

EWHA EURON

7.1 SVD(특이값 분해)

SVD(특이값 분해)

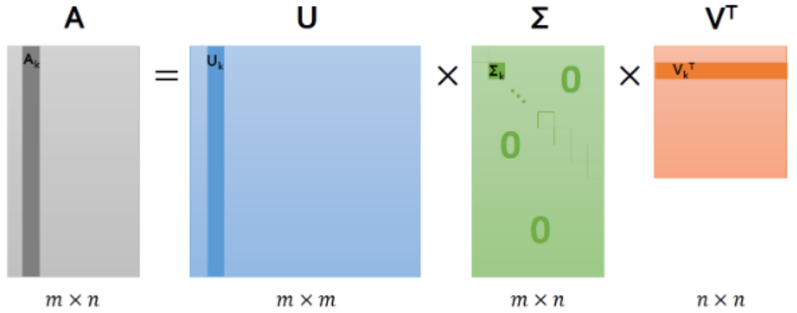
: m x n 크기의 데이터 (일반)행렬 A를 아래와 같이 분해하는 걸 의미함.

(행렬이 어떻게 분해되는가~를 보고 차원을 축소시키는 방법)

PCA?

m x m의 정방행렬 분해

$$A = U \sum V^T$$



- 행렬 U와 V에 속한 벡터는 특이벡터(singular vector)이며,

  모든 특이 벡터는 서로 직교하는 성질을 가짐

- Σ는 대각행렬(행렬의 대각에 위치한 값 외의 모든 값이 0)이다

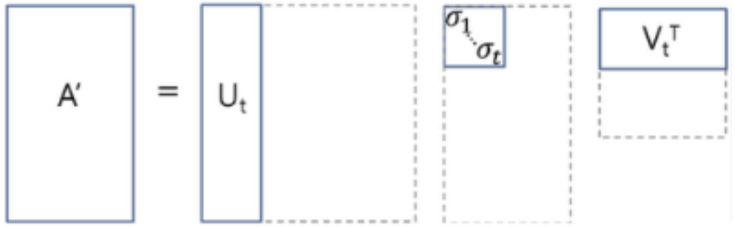
  → Σ이 위치한 0이 아닌 값이 행렬 A의 특이값

- SVD는 A의 차원이 m x n 일 때, U의 차원이 m x m, V^T의 차원이 n x n으로 분해함

https://blog.naver.com/PostView.naver?blogId=paragonyun&logNo=222465847517&from=search&redirect=Log&widgetTypeCall=true&directAccess=false
https://darkpgmr.tistory.com/106

7.1 SVD(특이값 분해)

Truncated SVD



Σ의 대각원소 중 상위 t개만 골라낸 형태.

→ 이렇게 하면 행렬 A를 원래의 상태로 회복할 수는 없지만,

U와 V의 원소도 함께 제거해 더욱 차원을 줄인 형태로 분해 가능해짐.

→ 데이터 정보를 상당히 압축했음에도 행렬 A를 근사할 수 있게 됨.

SVD 파이썬 구현

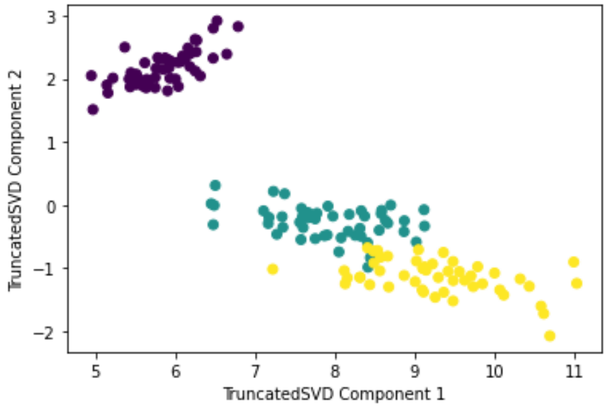
SVD → numpy.linalg.svd

Truncated SVD → scipy.sparse.linalg.svds, 사이킷런

7.2 사이킷런 TruncatedSVD 클래스 이용한 변환 (붓꽃)

```
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
%matplotlib inline

iris = load_iris()
iris_ftrs = iris.data
# 2개의 주요 component로 TruncatedSVD 변환
tsvd = TruncatedSVD(n_components=2)
tsvd.fit(iris_ftrs)
iris_tsvd = tsvd.transform(iris_ftrs)

# Scatter plot 2차원으로 TruncatedSVD 변환 된 데이터 표현. 품종은 색깔로 구분
plt.scatter(x=iris_tsvd[:,0], y= iris_tsvd[:,1], c= iris.target)
plt.xlabel('TruncatedSVD Component 1')
plt.ylabel('TruncatedSVD Component 2')
```

- 사이킷런의 TruncatedSVD 클래스는 Truncated SVD 연산을 수행해 원본 행렬을 분해한 U, sigma, Vt 행렬을 반환하지는 않음.
- PCA 클래스와 유사하게 fit()와 transform()을 호출해 원본 데이터를 몇 개의 주요 컴포넌트로 차원을 축소해 변환함.

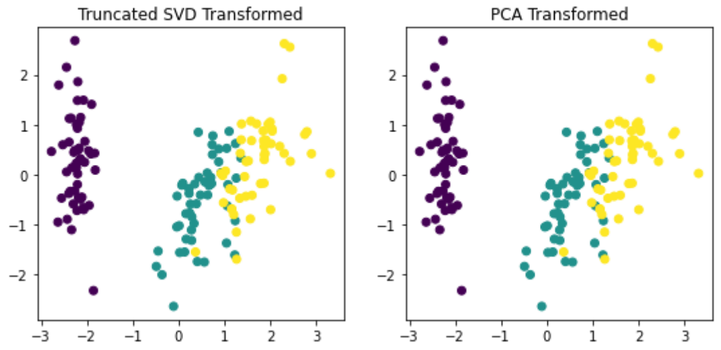


→ 변환 후, 품종별로 어느 정도 클러스터링이 가능할 정도로 각 변환 속성 뛰어난 고유성 가지고 있음.

7.2 TruncatedSVD vs. PCA

붓꽃 데이터 스케일링으로 변환한 뒤에 TruncatedSVD와 PCA 클래스 변환

```python
from sklearn.preprocessing import StandardScaler

# iris 데이터를 StandardScaler로 변환
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_ftrs)

# 스케일링된 데이터를 기반으로 TruncatedSVD 변환 수행
tsvd = TruncatedSVD(n_components=2)
tsvd.fit(iris_scaled)
iris_tsvd = tsvd.transform(iris_scaled)

# 스케일링된 데이터를 기반으로 PCA 변환 수행
pca = PCA(n_components=2)
pca.fit(iris_scaled)
iris_pca = pca.transform(iris_scaled)

# TruncatedSVD 변환 데이터를 왼쪽에, PCA변환 데이터를 오른쪽에 표현
fig, (ax1, ax2) = plt.subplots(figsize=(9,4), ncols=2)
ax1.scatter(x=iris_tsvd[:,0], y= iris_tsvd[:,1], c= iris.target)
ax2.scatter(x=iris_pca[:,0], y= iris_pca[:,1], c= iris.target)
ax1.set_title('Truncated SVD Transformed')
ax2.set_title('PCA Transformed')
```




```python
print((iris_pca - iris_tsvd).mean())
print((pca.components_ - tsvd.components_).mean())
```

```
2.3475954976278264e-15
-5.551115123125783e-17
```

두 개의 변환 행렬 값과 원본 속성별 컴포넌트 비율값을 실제로 서로 비교

→ 모두 0에 가까운 값 ➔ 2개의 변환이 서로 동일함

즉, 데이터 세트가 스케일링으로 데이터 중심이 동일해지면

사이킷런의 SVD와 PCA는 동일한 변환을 수행함.

→ PCA가 SVD 알고리즘으로 구현됐음을 의미함.

하지만 PCA는 밀집 행렬에 대한 변환만 가능하며

SVD는 희소 행렬에 대한 변환도 가능함.

7.3 SVD 활용

컴퓨터 비전 영역에서 이미지 압축을 통한 패턴 인식과 신호 처리 분야에 사용됨

<그림 7> 100개의 singular value로 근사 (t = 100)

<그림 8> 50개의 singular value로 근사 (t = 50)

<그림 9> 20개의 singular value로 근사 (t = 20)

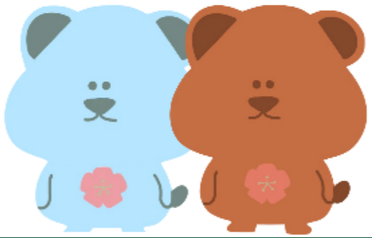
특이값 ↑ → 이미지의 전체적 구조 나타냄

특이값 ↓ → 이미지의 디테일한 부분

+ 추천 엔진(다양한 피처들을 분석해서 장르 분류),

문서의 잠재적 의미 파악 등에도 활용됨

08.NMF(Non-Negative Matrix Factorization)



EWHA EURON

8.1 NMF(Non-Negative Matrix Factorization)

NMF(Non-Negative Matrix Factorization)

: 음수를 포함하지 않는 행렬의 행렬 분해

(원본 행렬의 값들이 모두 양수면, 간단하게 행렬이 분해될 수 있음)

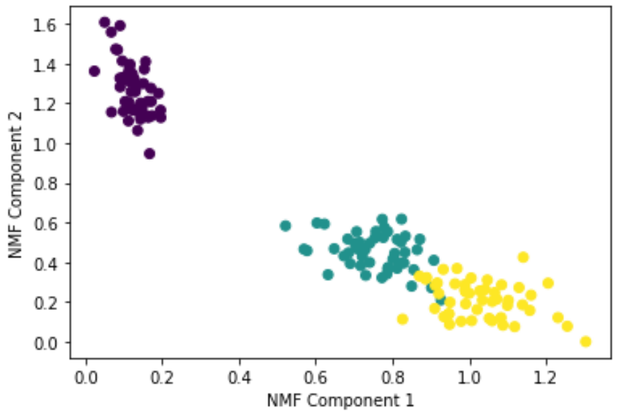
→ 이렇게 분해된 행렬은 잠재 요소를 특성으로 가지게 됨.

W: 원본 행에 대해서 이 잠재 요소의 값이 얼마나 되는지에 대응

H: 이 잠재요소가 원본 열(즉, 원본 속성)로 어떻게 구성되었는지를 나타내는 행렬

8.2 붓꽃 데이터 세트에 NMF 적용

```
from sklearn.decomposition import NMF
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
%matplotlib inline

iris = load_iris()
iris_ftrs = iris.data
nmf = NMF(n_components=2)
nmf.fit(iris_ftrs)
iris_nmf = nmf.transform(iris_ftrs)
plt.scatter(x=iris_nmf[:,0], y= iris_nmf[:,1], c= iris.target)
plt.xlabel('NMF Component 1')
plt.ylabel('NMF Component 2')
```

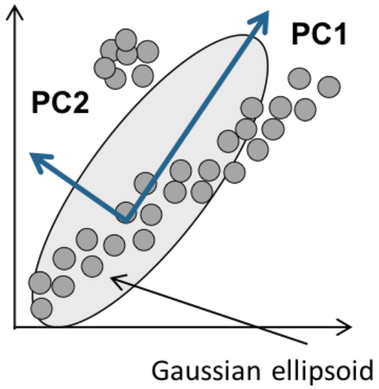


8.3 SVD 활용

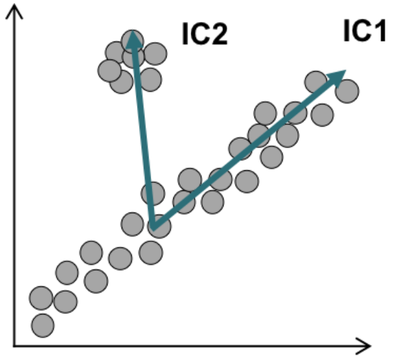
- SVD와 유사하게 이미지 압축을 통한 패턴 인식, 텍스트의 토픽 모델링 기법, 문서 유사도 및 클러스터링에 잘 사용됨.
- 영화 추천과 같은 추천 영역에서 활발하게 적용 됨

**PCA와 SVD는 모두 음수까지 반영해서 데이터가 모두 양수인 경우엔 오히려 NMF가 더 잘 축소시킴.

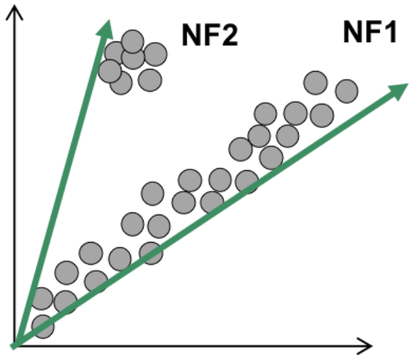
PCA does not 'sees' the data structure



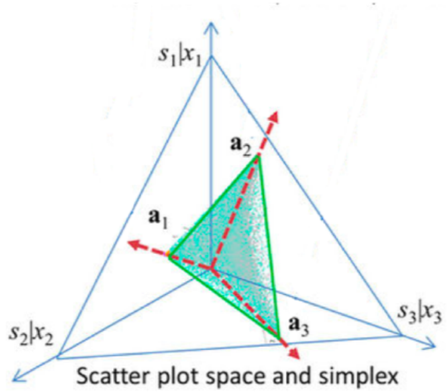
Independent components are directions of **non-gaussianity**



NMF components are **non-negative**



Fitting data to a **convex hull** shape

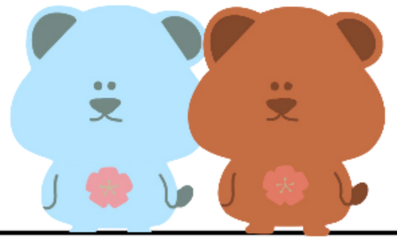


09. 다른 차원 축소 기법

9.1 다른 차원 축소 기법

LLE(지역 선형 임베딩)	- 비선형 차원 축소 기법, 투영 아닌 매니폴드 학습 - 서로 인접한 데이터들을 보존(neighborhood-preserving)하면서 고차원인 데이터셋을 저차원으로 축소하는 방법
다차원 스케일링(MDS)	- 샘플 간의 거리를 보존하면서 차원 축소
isomap	- 각 샘플을 가장 가까운 이웃과 연결하는 식으로 그래프를 만듦 - 샘플 간의 거리를 보존하면서 차원 축소
t-SNE	- 시각화에 많이 사용 - 고차원 공간에 있는 샘플의 군집을 시각화할 때 사용
랜덤 투영(random projection)	- 선형 투영을 사용해 데이터를 저차원 공간으로 투영 - 실제로 거리를 잘 보존하는 것으로 밝혀짐 - 차원 축소 품질은 샘플 수와 목표 차원수에 따라 다름

THANK YOU



EWHA EURON