강의계획표

주	해당 장	주제
1	1장 2장, 3장	머신러닝이란
2		머신러닝을 위한 기초지식, 구현을 위한 도구
3	4장	선형 회귀로 이해하는 지도학습
4	5장	분류와 군집화로 이해하는 지도 학습과 비지도 학습
5	6장	다양한 머신러닝 기법들 - 다항 회귀, Logistic Regression - 정보이론, 결정트리 - SVM, Ensemble
6		
7		
8		중간고사 (04-20)
9	7장	인공 신경망 기초 - 문제와 돌파구
10	8장	고급 인공 신경망 구현
11	9장	신경망 부흥의 시작, 합성곱 신경망
12	10장	순환 신경망
13	11장	차원축소와 매니폴드 학습
14	12장	오토인코더와 잠재표현 학습
15	13장	AI의 현재와 미래
16		기말고사

11장 차원 축소와 매니폴드 학습

13 주차

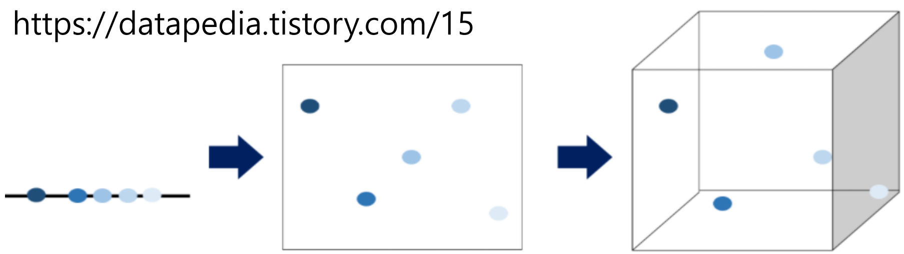
- 데이터의 특징이 많아질 때 발생하는 문제
- 특이값 분해와 커널 트릭을 이용한 주성분 분석
- 매니폴더 학습을 이용한 차원축소, 선형대수 기법
- 특징을 추출하여 데이터를 압축하고 복원하는 방법

차원, 차원의 저주 (1)

❖ 차원dimension
- 어떤 공간에 존재하는 데이터들을 식별하는 데에 필요한 최소 수의 좌표값
- 2차원 공간의 점들은 2개의 값을 가진 좌표로 표현하고, 3차원 공간의 점들은 3개의 값을 가진 좌표로 표현할 수 있음
- 데이터를 다룰 때 각 데이터를 표현하는 특징feature의 수

❖ 차원이 높아지면서 발생하는 문제
- 차원 수가 증가함에 따른 파라미터 항의 수에 비해 학습차원 수보다 적어져서 성능이 저하되는 현상. (인스턴스 수는 동일)
  - K-NN의 경우 이웃하는 node가 점점 멀어짐

https://datapedia.tistory.com/15



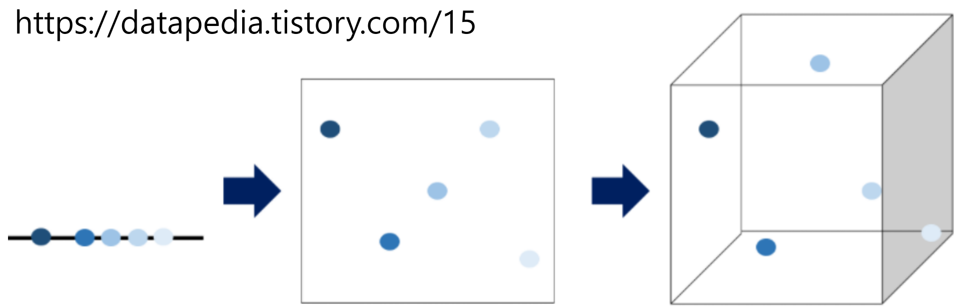
- 급격히 커지는 계산 복잡도
  - 6개의 독립변수의 8차항 다항식의 경우: 3,003 개의 항
  - 6개의 독립변수의 15차항 다항식의 경우 54,264개의 항

차원, 차원의 저주 (2)

❖ **차원의 저주**Curse of Dimensionality :

- 차원: 데이터들을 식별하는 데에 필요한 값의 개수, **각 데이터를 표현하는 특징**feature**의 종류**.
  - 6개의 독립변수의 8차항 다항식의 경우: 3,003 개의 항
  - 6개의 독립변수의 15차항 다항식의 경우 54,264개의 항
- 차원 수가 증가함에 따른 파라미터 항의 수에 비해 학습차원 수보다 적어져서 성능이 저하되는 현상. (인스턴스 수는 동일)
  - K-NN의 경우 이웃하는 node가 점점 멀어짐

https://datapedia.tistory.com/15

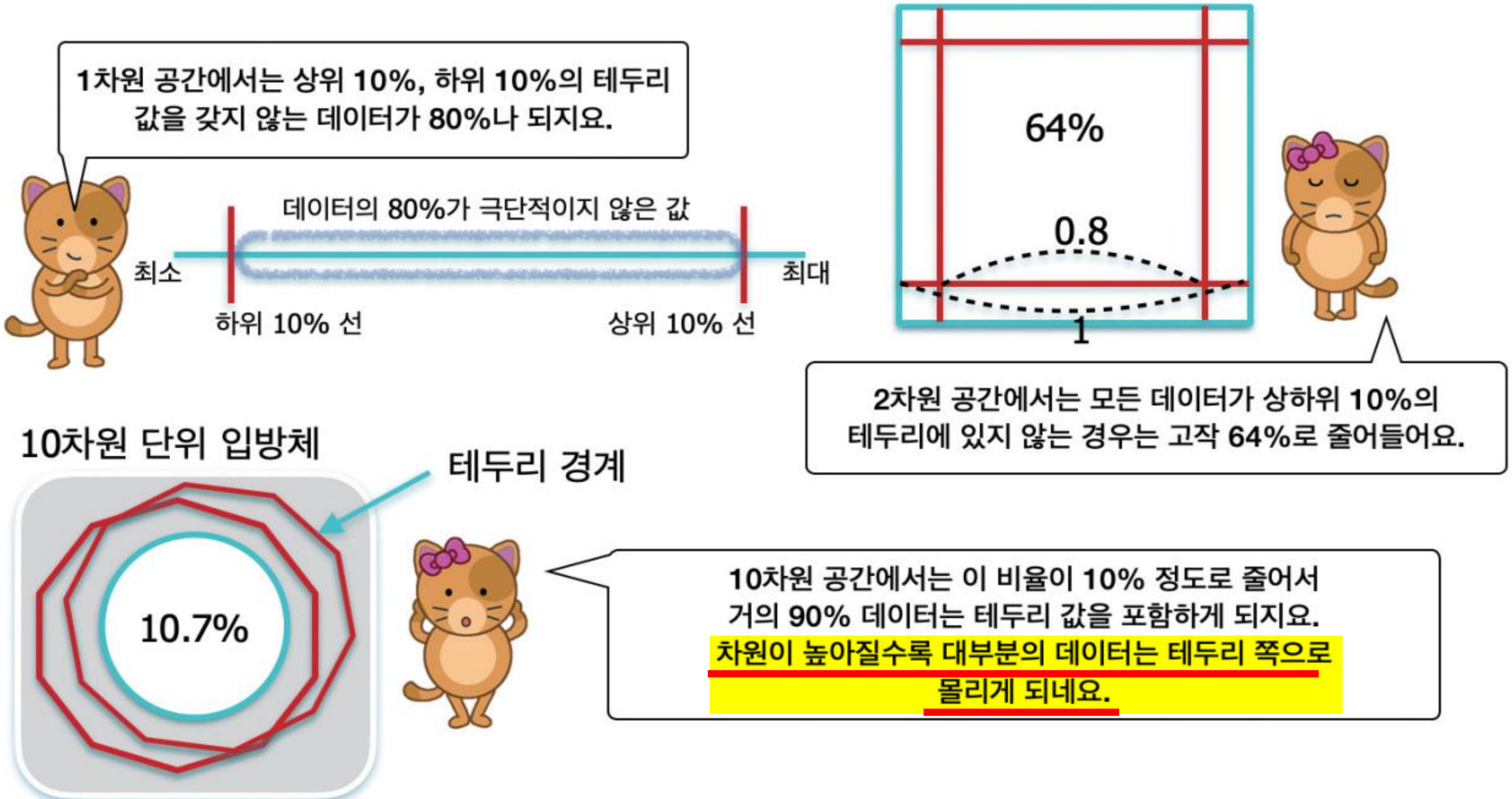


  - n 차원의 초입방체에서의 무작위로 선택한 두 점의 거리: $\sqrt{\frac{n}{6}}$
- 급격히 커지는 계산 복잡도

차원, 차원의 저주 (3)

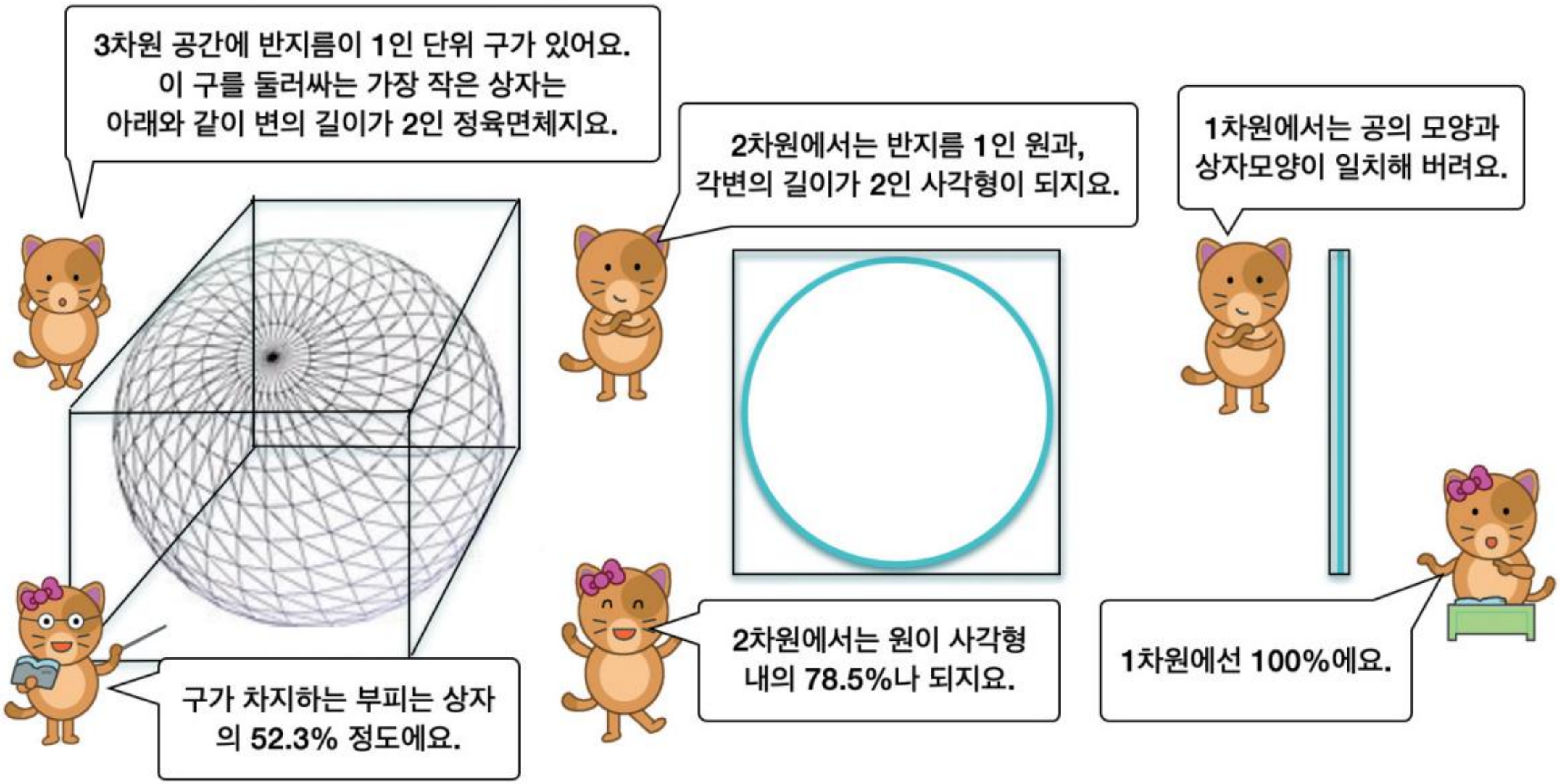
❖ **차원의 저주**Curse of Dimensionality :

- 데이터가 균일 분포로 흩어져 있다고 가정
- 데이터의 밀도가 낮아져 데이터 사이의 관계를 파악하기 힘들어짐



차원, 차원의 저주 (4)

❖ **차원의 저주**Curse of Dimensionality :

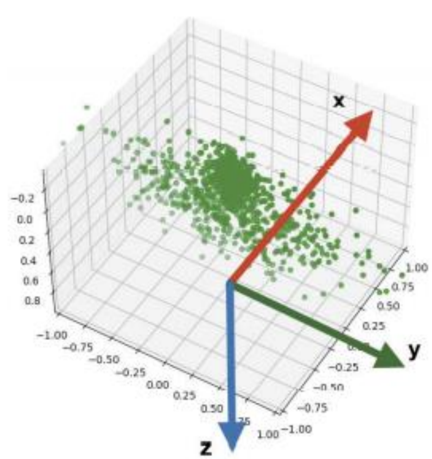


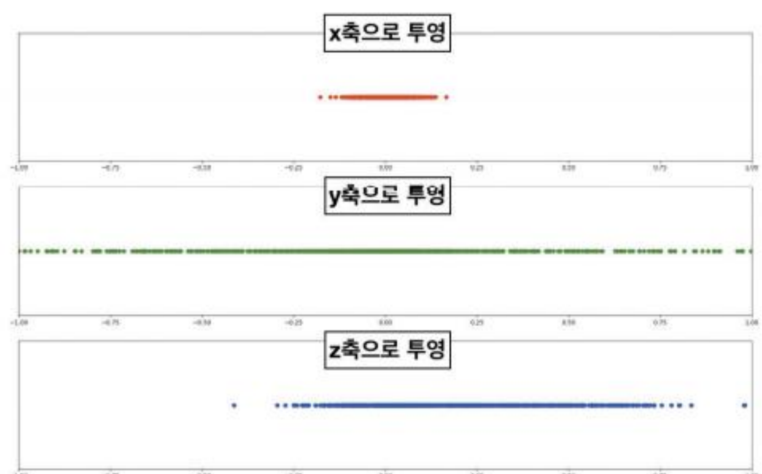
7

차원 축소 (1)

❖ 차원 축소dimensionality reduction
- 데이터를 표현하기 위한 특징의 수를 줄이는 것
- 높은 차원의 데이터는 많은 양의 데이터를 가지고 있지만, 높은 차원의 데이터를 다룰 때는 중요하지 않은 차원을 생략하고 중요한 차원만 남김
- **정보의 손실을 최소로 하면서 특징의 수를 줄이는 방법 필요**

❖ 특징 선택feature selection
- 변동이 거의 없는 차원의 특징은 데이터 구분에 덜 중요한 특징
- 관련성이 낮고 중복되거나 불필요한 정보를 담은 차원을 버림



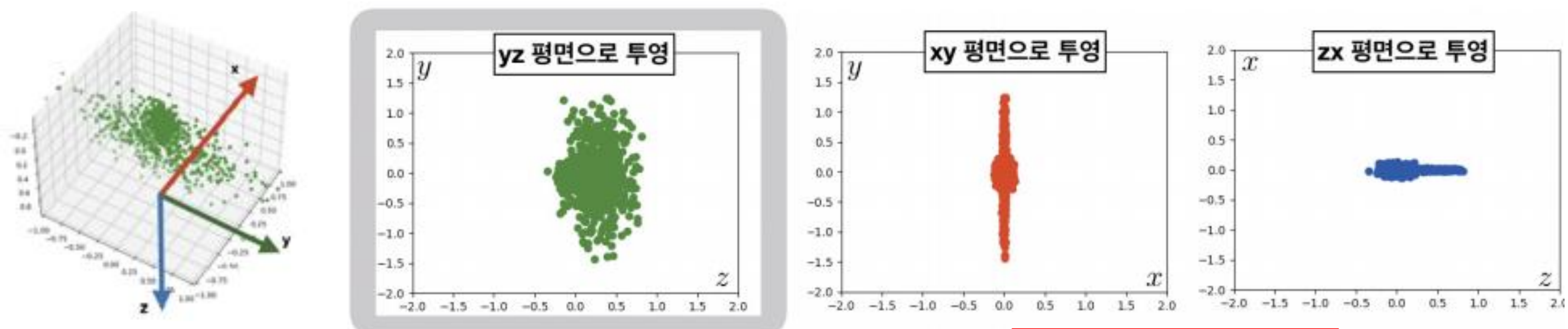


각 축의 분산에 의해 중요한 축을 파악: $y > z > x$

차원 축소 (2)

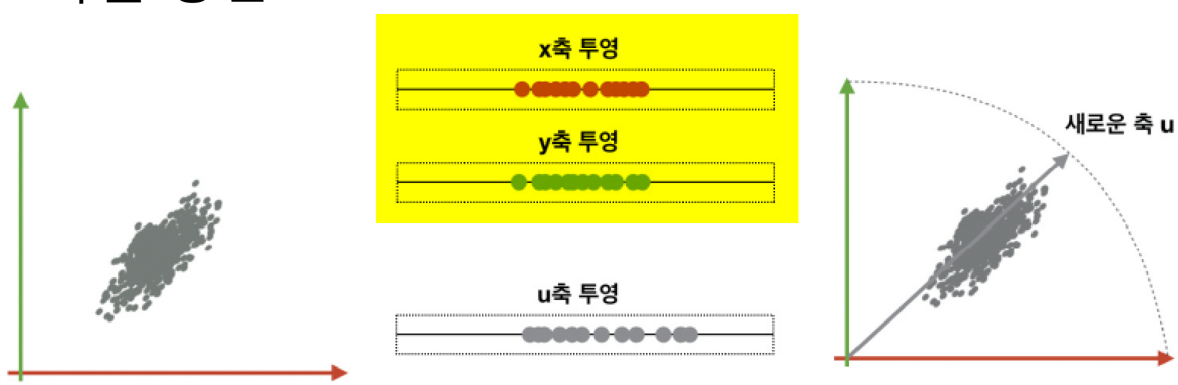
❖ 투영projection: 데이터를 낮은 차원에 떨어뜨리는 것

❖ 특징 투영projection: 원본 데이터의 분산을 최대로 유지하는 방향으로 투영이 일어나게 하는 것(축을 하나 제외하는 직교 투영)



zx 평면으로 투영하면
정보손실이 가장 많다

❖ 2차원 공간



새로운 축$(1/\sqrt{2}, 1/\sqrt{2})$에 대해 투영하면 분산이 잘 유지됨(정보손실이 적음)
= 가장 좋은 축 : 주성분

주성분 분석과 특이값 분해 (1)

❖ 주성분principal component
- 데이터의 분산을 가장 잘 유지하는 축들
- k번째 주성분은 이전의 $k-1$개의 주성분과 모두 직교하면서 데이터의 분산을 가장 잘 보존하는 축
- 원본데이터셋과 투영된 데이터 셋 사이의 평균제곱거리를 최소화하는 축
- '데이터 공분산 행렬'의 고유값 분해 혹은 '데이터 행렬'의 SVD(특이값 분해) 로 계산

❖ 주성분 분석principal component analysis:PCA
- 데이터의 특징을 유지하며 차원을 축소할 수 있는 가장 대표적인 차원 축소 알고리즘
- 데이터에 가장 가까운 초평면hyperplane을 찾아 데이터를 투영
- 분산을 가장 잘 유지하는 축을 찾아, 이들을 이용하여 투영면을 구성
- 알고리듬
  1. 분산을 최대로 유지하는 축을 찾기
  2. 찾은 축에 직교하면서 남은 분산을 최대로 유지하는 축 찾기
  3. 원하는 투영면의 차원(d<n)에 도달할 때까지 2번을 반복

주성분 분석과 특이값 분해 (2)

❖ 공분산covariance 행렬

- 공분산과 공분산 행렬(covariance)
- 공분산: 2개 변수가 함께 변하는 정도(joint variability)를 측정하는 척도
  - $Cov(X, Y) = \sigma_{xy} = E[(X-\mu_x)(Y-\mu_y)] = E(XY) - \mu_x\mu_y$
  - 상관계수: X, Y의 공분산에서 단위 크기 고려

$$Corr(X, Y) = \rho = \frac{Cov(X, Y)}{S(X)S(Y)} = \frac{E[(X-\mu_x)(Y-\mu_y)]}{\sqrt{E(X-\mu_x)^2 \cdot E(Y-\mu_y)^2}}$$

- 공분산covariance 행렬: 변수들 사이의 공분산을 행렬 형태로 나타낸 것
  - 정방행렬square matrix, 대칭행렬symmetric matrix
  - 대각항은 단일변수의 분산 σ^2

$$\Sigma = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & \cdots & Cov(X_1, X_p) \\ Cov(X_2, X_1) & Var(X_2) & \cdots & Cov(X_2, X_p) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(X_p, X_1) & Cov(X_p, X_2) & \cdots & Var(X_p) \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_p^2 \end{bmatrix}$$

❖ 참고sites

- 머신러닝 - 19. 고유값(eigenvalue), 고유벡터(eigenvector ...
- 데이터사이언스스쿨

주성분 분석과 특이값 분해 (3)

❖ 고유값 분해eigen decomposition

- 고유값과 고유벡터를 찾는 작업
  - A: 정방행렬
  - λ : 고유값
  - v : 고유벡터

$$Av = \lambda v$$

  - cv (c: 실수) v의 방향과 같은 벡터도 모두 고유벡터

$$A(cv) = cAv = c\lambda v = \lambda(cv)$$

  - 고유벡터는 길이가 1인 단위벡터로 표현 $\dfrac{v}{\|v\|}$

예제

$$A = \begin{bmatrix} 1 & -2 \\ 2 & -3 \end{bmatrix} \quad v = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \lambda = -1$$

$$Av = \begin{bmatrix} 1 & -2 \\ 2 & -3 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} = (-1)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \lambda v$$

단위벡터

$$v = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \approx \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}$$

- 특성방정식
  - $\det(A - \lambda I) = 0$
  - 예제 $A = \begin{bmatrix} 1 & -2 \\ 2 & -3 \end{bmatrix}$

$$\begin{aligned} \det(A - \lambda I) &= \det\left(\begin{bmatrix} 1 & -2 \\ 2 & -3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right) \\ &= \det\begin{bmatrix} 1-\lambda & -2 \\ 2 & -3-\lambda \end{bmatrix} \\ &= (1-\lambda)(-3-\lambda) + 4 \\ &= \lambda^2 + 2\lambda + 1 = 0 \end{aligned}$$

$$\lambda^2 + 2\lambda + 1 = (\lambda+1)^2 = 0$$

고유값: -1 (중복고유값)

$$\det(A) = \prod_{i=1}^{N} \lambda_i$$

$$\mathrm{tr}(A) = \sum_{i=1}^{N} \lambda_i$$

- 고유벡터 계산

$$(A - \lambda I)v = 0$$

$$\begin{bmatrix} 1+1 & -2 \\ 2 & -3+1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} 2 & -2 \\ 2 & -2 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$2v_1 - 2v_2 = 0$$

$$v_1 = v_2 \qquad v = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad v = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$

주성분 분석과 특이값 분해 (4)

❖ 대각화diagonalization & 행렬분해

- N 차원의 정방행렬 A 가 N개의 고유값(복소수)과 고유벡터를 가짐

  $\lambda_1, \lambda_2, \cdots, \lambda_N \quad v_1, v_2, \cdots, v_N$

- 고유벡터행렬 $V = [v_1 \cdots v_N]$
- 고유값행렬 $\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{bmatrix}$

- **행렬과 고유벡터행렬의 곱은 고유벡터행렬과 고윳값행렬의 곱과 같다.**

$$
\begin{aligned}
AV &= A[v_1 \cdots v_N] \\
&= [Av_1 \cdots Av_N] \\
&= [\lambda_1 v_1 \cdots \lambda_N v_N] \\
&= [v_1 \cdots v_N] \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{bmatrix} \\
&= V\Lambda
\end{aligned}
$$

예제

$$B = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$$

$$V = \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{1}{\sqrt{2}} \\ \frac{2}{\sqrt{13}} & \frac{1}{\sqrt{2}} \end{bmatrix} \quad \Lambda = \begin{bmatrix} 4 & 0 \\ 0 & -1 \end{bmatrix}$$

$$V^{-1} = \frac{1}{5} \begin{bmatrix} \sqrt{13} & \sqrt{13} \\ -2\sqrt{2} & 3\sqrt{2} \end{bmatrix}$$

- V의 역행렬이 존재한다면

$$A = V\Lambda V^{-1}$$

$$B = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} = V\Lambda V^{-1} = \frac{1}{5} \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{1}{\sqrt{2}} \\ \frac{2}{\sqrt{13}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \sqrt{13} & \sqrt{13} \\ -2\sqrt{2} & 3\sqrt{2} \end{bmatrix}$$

주성분 분석과 특이값 분해 (5)

❖ 주성분 벡터

- 입력 데이터들의 공분산 행렬에 대한 고유값분해 때 나오는 고유벡터.
- 데이터의 분포에서 분산이 큰 방향을 나타내고, 대응되는 고유값이 그 분산의 크기를 나타냄

PCA

- C : covariance matrix of x
- $C = P\Sigma P^T$ (P: orthogonal, Σ: diagonal)

$$C = \begin{pmatrix} \boxed{e_1} & \cdots & \boxed{e_n} \end{pmatrix} \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \begin{pmatrix} \boxed{e_1^T} \\ \vdots \\ \boxed{e_n^T} \end{pmatrix}$$

- P : n×n orthogonal matrix
- Σ : n×n diagonal matrix
- $Ce_i = \lambda_i e_i$
  - e_i : eigenvector of C, direction of variance
  - λ_i : eigenvalue, e_i 방향으로의 분산
  - $\lambda_1 \geq \ldots \geq \lambda_n \geq 0$
- e_1: 가장 분산이 큰 방향
- e_2: e_1에 수직이면서 다음으로 가장 분산이 큰 방향
- e_k: $e_1, \ldots, e_{k-1}$에 모두 수직이면서 가장 분산이 큰 방향

주성분 분석과 특이값 분해 (6)

❖ 특이값 분해singular value decomposition:SVD

- $M = U\Sigma V^T$, $M \in \mathbb{R}^{m\times n}$, 랭크rank $r = \min(m,n)$
  - $U \in \mathbb{R}^{\mathrm{m\times m}}$, $V \in \mathbb{R}^{n\times n}$ 직교orthogonal 행렬
  - $UU^T = I \in \mathbb{R}^{m\times m}$, $VV^T = I \in \mathbb{R}^{n\times n}$: 이 행렬들의 각 행과 열이 단위 벡터이고, 서로 직교하는 축
  - Σ: $m \times n$의 대각diagonal 행렬, 대각행렬의 i번째 대각 성분(특이값)은 $\sigma_i > \sigma_{i+1}$이 되도록 한다. 즉, 큰 값이 먼저 나오도록 한다.
  - $U_{*,\mathrm{i}}$는 U행렬의 i열 벡터이고, $V_{*,\mathrm{i}}(V_{\mathrm{i},*}^{\mathrm{T}})$는 행렬 V의 i열 벡터

m > n

$$\mathbf{M} = \begin{pmatrix} | & | & \cdots & | \\ U_{*,1} & U_{*,2} & \cdots & U_{*,m} \\ | & | & \cdots & | \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} - & V_{1,*}^{\mathrm{T}} & - \\ - & V_{2,*}^{\mathrm{T}} & - \\ \vdots & \vdots & \vdots \\ - & V_{n,*}^{\mathrm{T}} & - \end{pmatrix}$$

m < n

$$\mathbf{M} = \begin{pmatrix} | & | & \cdots & | \\ U_{*,1} & U_{*,2} & \cdots & U_{*,m} \\ | & | & \cdots & | \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots & \vdots & 0 \\ 0 & 0 & \cdots & \sigma_m & \cdots & 0 \end{pmatrix} \begin{pmatrix} - & V_{1,*}^{\mathrm{T}} & - \\ - & V_{2,*}^{\mathrm{T}} & - \\ \vdots & \vdots & \vdots \\ - & V_{n,*}^{\mathrm{T}} & - \end{pmatrix}$$

주성분 분석과 특이값 분해 (7)

❖ 특이값 분해와 주성분벡터

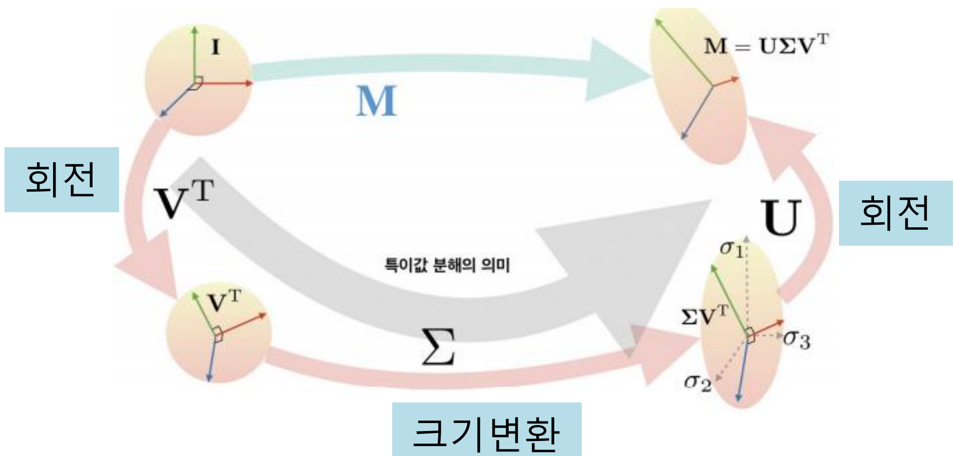
- $M = U\Sigma V^T$, $M \in \mathbb{R}^{m\times n}$
- 공분산 행렬 $C = M^T M/(m-1)$

Λ로 축약

$$\mathbf{C} = \frac{(\mathbf{V\Sigma U^T})(\mathbf{U\Sigma V^T})}{m-1} = \frac{\mathbf{V\Sigma^2 V^T}}{m-1} = \mathbf{V}\boxed{\frac{\mathbf{\Sigma}^2}{m-1}}\mathbf{V^T} = \mathbf{V\Lambda V^T} = \mathbf{W\Lambda W^{-1}}$$

- 특이값 분해를 통해 얻은 V가 바로 주성분 벡터, 공분산의 고유벡터로 구성된 W와 동일

❖ 특이값 분해의 기하학적 의미



행렬 M을 표현하기 위해
- n차원의 공간을 회전시키는 동작이 V에 의해 이루어지고,
- M행렬이 가진 크기에 맞춰 변형하는 작업이 Σ에 의해 이루어지고,
- 최종적인 일치가 U에 의해 이루어짐

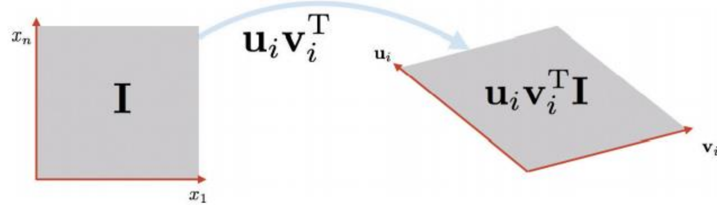
주성분 분석과 특이값 분해 (7)

❖ 특이값 분해의 기하학적 의미

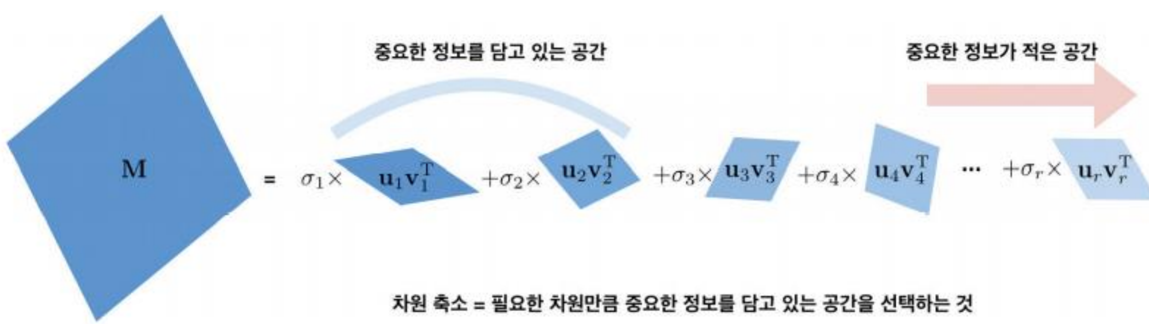
- 행렬 M의 랭크rank r은 $\min(m, n)$이다.
- 행렬 U의 i번째 기저 벡터 $U_{*,i}$와 행렬 V의 i번째 기저 벡터 $V_{*,i}$를 간단히 u_i와 v_i로 표현하면 특이값 분해를 나타내는 식 $M = U\Sigma V^T$은 다음과 같이 랭크를 이용하여 표현

$$\mathbf{M} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^{\mathrm{T}} + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^{\mathrm{T}} + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^{\mathrm{T}}$$

- 두 단위 벡터의 외적outer product인 $u_i v_i^T$는 두 벡터가 만들어내는 공간



- 특이값 분해는 행렬 M을 이러한 공간의 가중합weighted sum으로 표현하는 것
- 가중치는 특이값이며, 이 특이값이 클수록 더욱 중요한 데이터의 정보를 담고 있는 공간이 되는 것

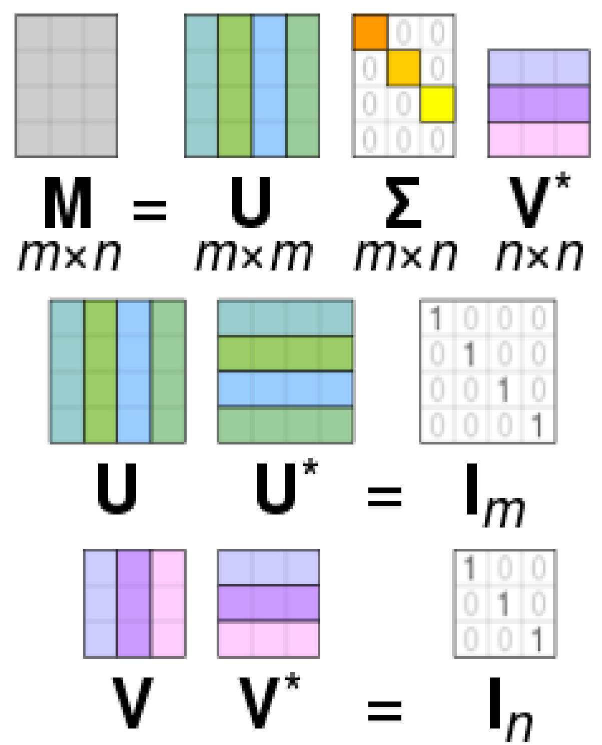


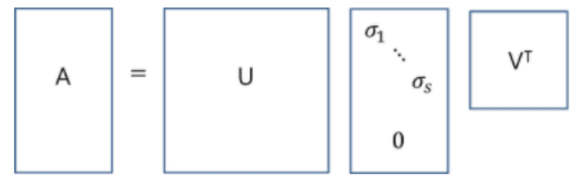
주성분 분석과 특이값 분해 (8)

[선형대수학 #4] 특이값 분해의 활용

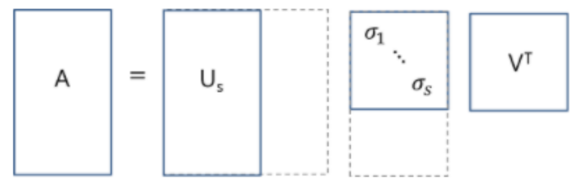
❖ Reduced SVD와 행렬근사, 데이터 압축

$M = U\Sigma V^*$

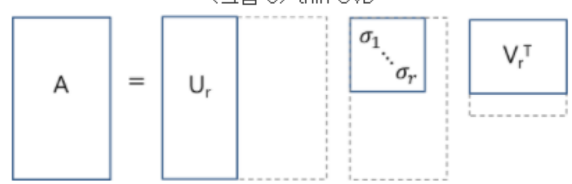




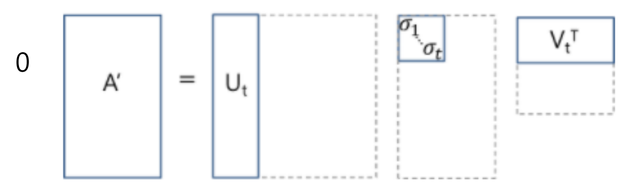
<그림 2> full SVD



<그림 3> thin SVD



<그림 4> Compact SVD



<그림 5> Truncated SVD


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


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


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

K차원으로의 축소(압축): $\mathbf{M}_k = \mathbf{M}\mathbf{V}_{*,:k}$

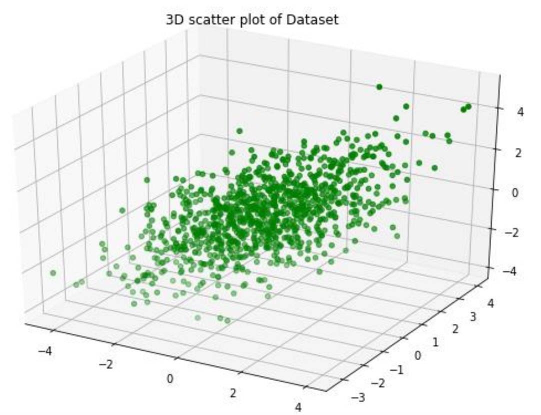
Program (1)

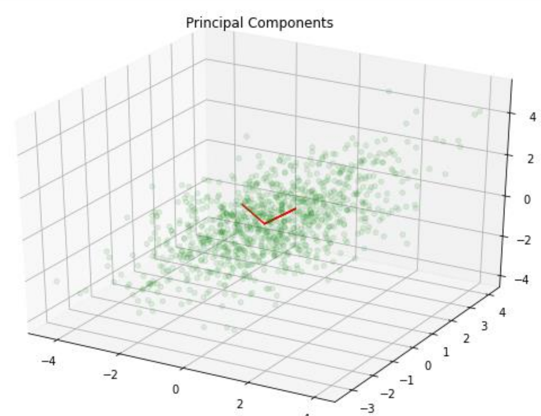
❖ **LAB**[11-1] 3차원 공간의 데이터에서 주성분

실습 목표

3차원 공간에 $\mathbf{u}=\left[\frac{1}{\sqrt{3}}\frac{1}{\sqrt{3}}\frac{1}{\sqrt{3}}\right]^T$ 와 $\mathbf{v}=\left[\frac{1}{\sqrt{2}}0-\frac{1}{\sqrt{2}}\right]^T$ 를 축으로 하는 2차원 부분 공간에 약간의 잡음을 더한 데이터를 생성하자. 이 데이터에 주성분 분석을 한 뒤 찾은 주성분이 잡음을 생성한 축과 일치하는지 확인해 보자.

- https://colab.research.google.com/drive/1Z7rky4SzC4_qbbPk_aR6zp2mtj4LipEx

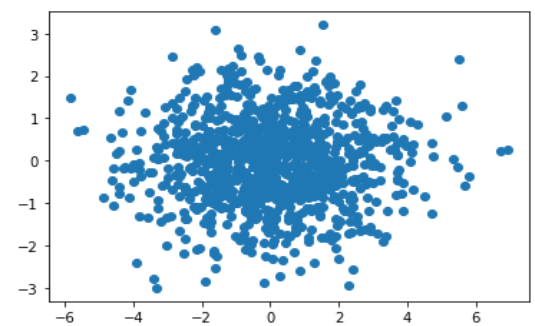




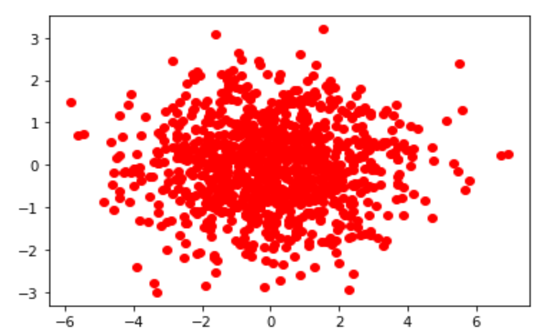

```
X_adj = X - X.mean(axis = 0)    # 데이터 중심을 원점으로 옮김
U, S, Vt = np.linalg.svd(X_adj) # 특이값 분해를 실행함
Vt.T                            # 주성분을 담고 있는 행렬

array([[ 0.58733034, -0.69956107,  0.40701029],
       [ 0.57727521,  0.00962061, -0.81649298],
       [ 0.56727101,  0.71450805,  0.40948973]])
```

```
S

array([60.91282589, 31.54280584,  0.89591028])
```

$$X_{2d} = X\, W_{2d} = X\, V_{*,:2}$$



np.linalg.svd 결과



Sklearn.decomposition.PCA 결과

Kernel PCA (1)

❖ 커널 트릭kernel trick

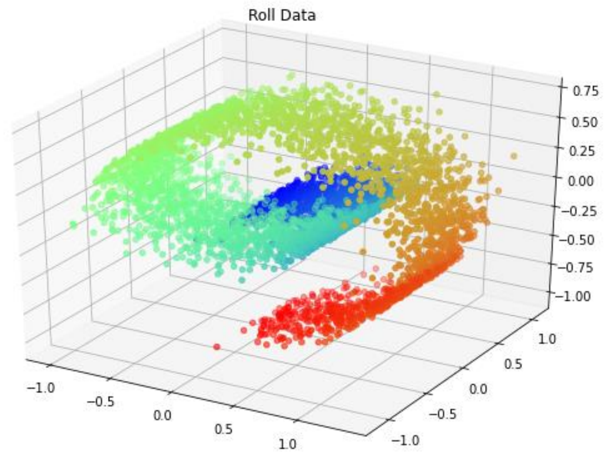
- 특징 공간을 다항화하지 않고도 비슷한 효과를 얻을 수 있는 방법
- 주성분 분석에도 똑같이 적용

커널 이름	커널 함수
선형 커널	$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \mathbf{x}^{(i)}{}^{\mathrm{T}}\mathbf{x}^{(j)}$
다항 커널	$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\gamma\mathbf{x}^{(i)}{}^{\mathrm{T}}\mathbf{x}^{(j)} + r)^d$
방사 기저 함수 커널	$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = e^{\gamma\|\mathbf{x}^{(i)}-\mathbf{x}^{(j)}\|^2}$
시그모이드 커널	$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\gamma\mathbf{x}^{(i)}{}^{\mathrm{T}}\mathbf{x}^{(j)} + r)$
코사인 커널	$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \mathbf{x}^{(i)}{}^{\mathrm{T}}\mathbf{x}^{(j)}/\|\mathbf{x}^{(i)}\|\|\mathbf{x}^{(j)}\|$

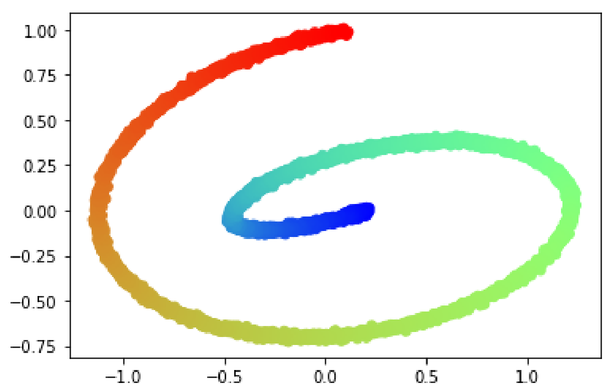
```
from sklearn.decomposition import PCA, KernelPCA
```

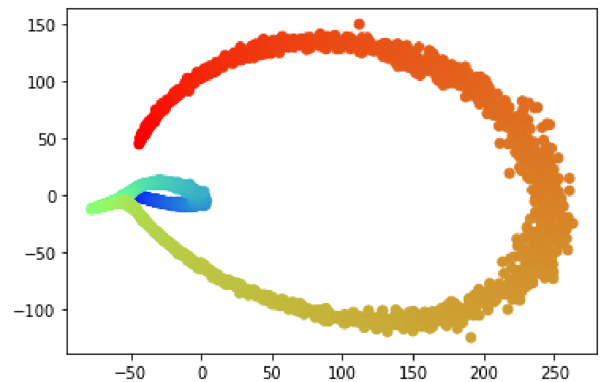
Kernel PCA (2)

❖ 커널 트릭kernel trick



❖ 주성분 분석과 커널 트릭을 사용한 주성분 분석 비교



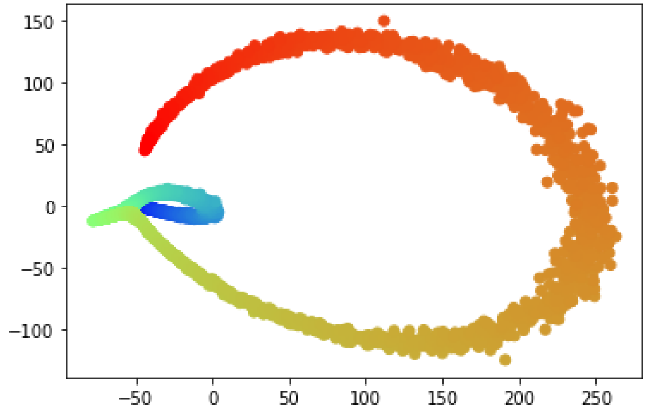



```
pca = PCA(n_components = 2)
X_2d = pca.fit_transform(X)
```

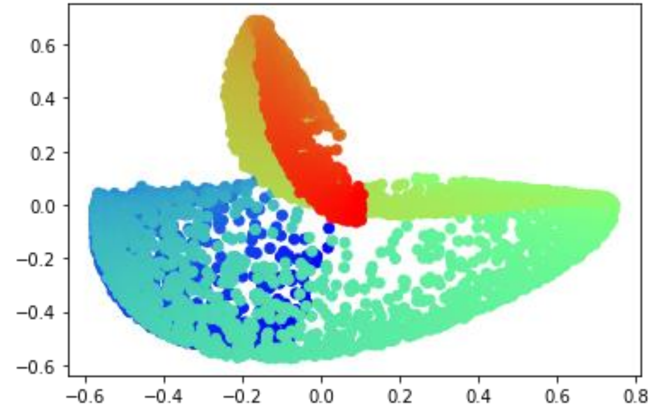
```
krn_pca = KernelPCA(n_components = 2, kernel='polynomial',
          gamma = 3.5, degree = 5, coef0 = 3.5)
X_2d_kernel = krn _pca.fit_transform(X)
```

Kernel PCA (3)

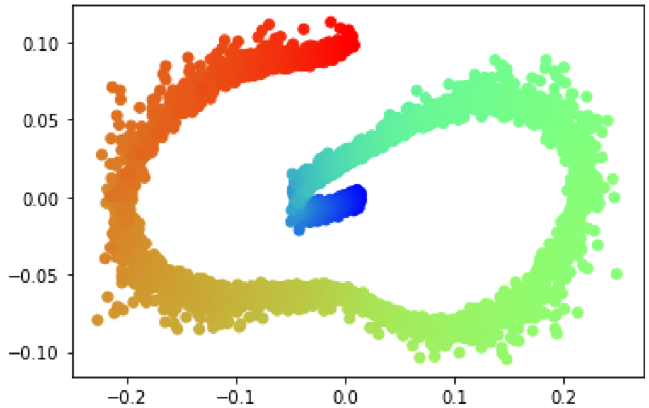
- Polynomial 커널



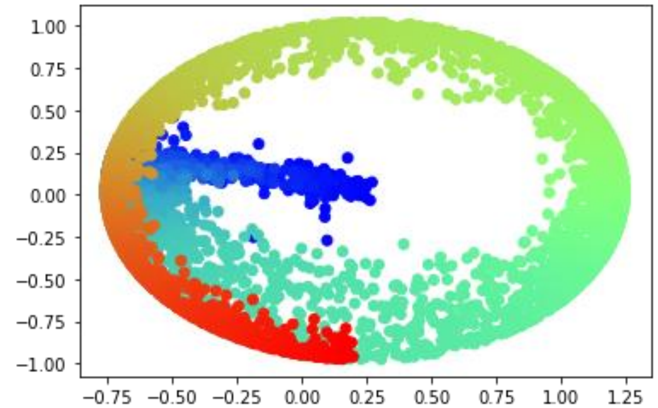
- rbf 커널



- Sigmoid 커널



- cosine 커널



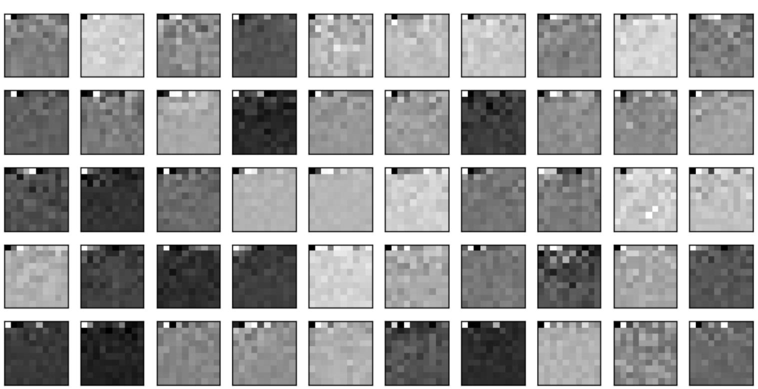
Program (2)

❖ LAB11-2 주성분을 추출해 이미지 압축

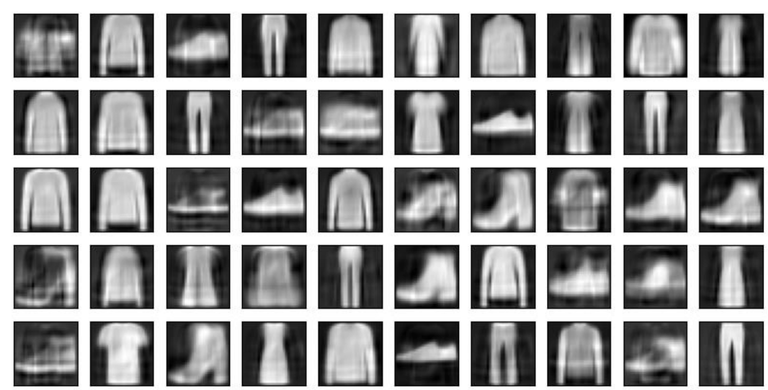
- 28x28 = 784 차원 => (PCA) 100차원
- https://colab.research.google.com/drive/1Z7rky4SzC4_qbbPk_aR6zp2mtj4LipEx

원본 이미지

$X_{recoverd} = X_{reduced} V_{100}^{T}$



$X_{reduced} = X_{train} V_{100}$



$X_{recoverd} = X_{reduced} V_{25}^{T}$

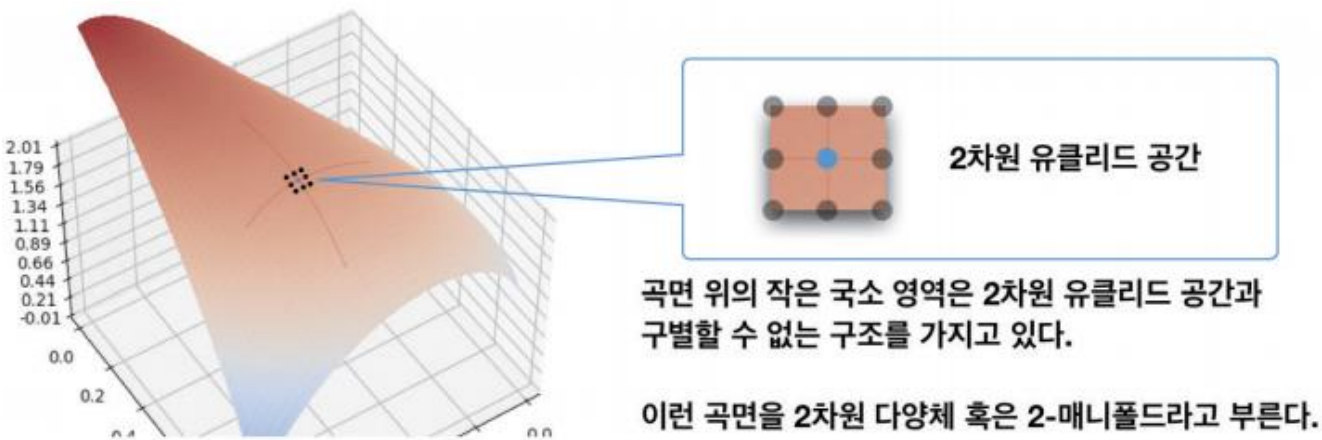
매니폴드 학습 (1)

- 주성분 분석PCA
  - PCA는 데이터를 새로운 직교 좌표계로 옮겨 놓는 선형 차원 축소
- 매니폴드(다양체)manifold
  - 데이터 분포의 **비선형(non-linear)** 구조
  - 매우 작은 영역을 관찰할 때 이웃 점들 사이의 관계를 유클리드 공간으로 표현할 수 있는 공간
  - n차원 다양체는 국소적으로 볼 때 각 점들이 이웃 점들과 n차원의 유클리드 공간을 이룸
- 매니폴드 차원 축소 기법
  - 국소적 선형 임베딩locally linear embedding, LLE
  - MDSMulti-Dimension Scaling : 데이터 포인트 간의 거리를 보존하면서 차원을 축소
  - **Isomap**: 각 샘플에 대해 가장 가까운 샘플을 연결하여 그래프를 만들어 **측지선geodesic** 거리의 대소 관계를 유지하며 차원을 축소하는 기법
  - t-SNEt-distributed Stochastic Neighbor Embedding: 비슷한 데이터 인스턴스들끼리 가까운 거리를 유지하도록 하면서 더 낮은 차원의 임베딩 공간을 찾는 분석
  - **LDA**Linear Discriminant Analysis: Supervised learning이며, 분류 알고리즘에 속한다. LDA는 학습 단계에서 클래스를 가장 잘 구분하는 축을 학습하며, 이 축은 데이터가 투영되는 초평면을 정의하는 데 사용할 수 있다. 이러한 초평면으로 데이터를 투영하게 되면 클래스 간의 거리를 멀리 떨어지게 축소
- 참고 sites
  - [Manifold Learning] IsoMap, LLE, t-SNE 설명 – Wordtory
  - Embedding for Word Visualization (LLE, ISOMAP, MDS, t-SNE)

매니폴드 학습 (2)

❖ 국소적 선형 임베딩locally linear embedding, LLE

- 데이터의 차원 축소를 위해 인접한 데이터들이 차지하는 작은 영역을 선형관계로 보고 이러한 관계를 잘 유지하는 임베딩 방법



매니폴드 학습 (3)

❖ LLE 알고리듬

1. 모든 점 x_i 에 대해 k개의 이웃 지점들을 찾음, $\mathrm{x}_j, j \in N_i$ (N_i: 이웃 지점 k개의 인덱스 집합)
2. 모든 점 x_i 에 대해 이웃 지점들과의 선형 관계를 찾음
   - $\mathbf{x}_i = \sum_{j \in N_i} w_{ij} \mathbf{x}_j$
   - 오차를 최소화하는 가중치 $\underset{\mathbf{w}_i}{\operatorname{argmin}} \left| \mathbf{x}_i - \sum_{j \in N_i} w_{ij} \mathbf{x}_j \right|^2$
   - 가중치의 합이 1이 되도록 함. $\sum_{j \in N_i} w_{ij} = 1$
3. 앞에서 찾은 선형관계가 유지되는 임베딩 공간 y 로 옮김
   - $\underset{\mathbf{y}}{\operatorname{argmin}} \sum_{i=1}^{m} \left| \mathbf{y}_i - \sum_{j \in N_i} w_{ij} \mathbf{y}_j \right|^2$
   - 제약조건1: 데이터의 평균이 원점 $\sum_{i=1}^{m} \mathbf{y}_i = 0$
   - 제약조건2: 임베딩이 이루어진 데이터가 단위 공분산을 가짐

$$\frac{1}{m} \sum_{i=1}^{m} \mathbf{y}\mathbf{y}^T = \mathbf{I}$$

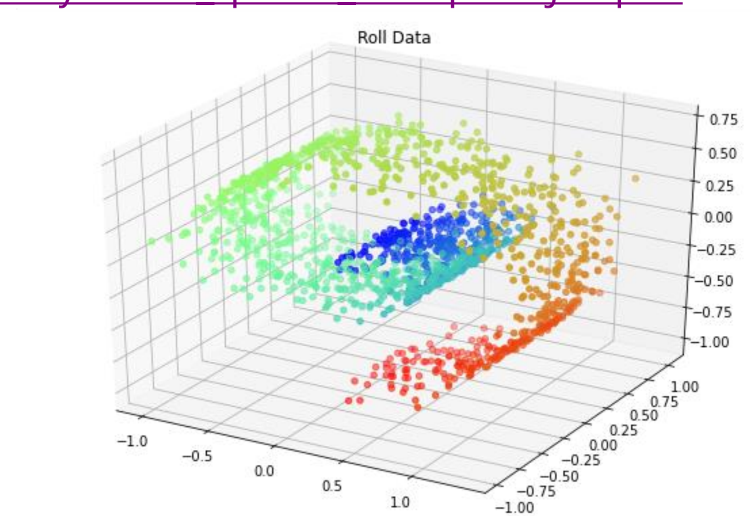
제약조건이 있는 최적화 해:
라그랑주 승수법Lagrange multiplier

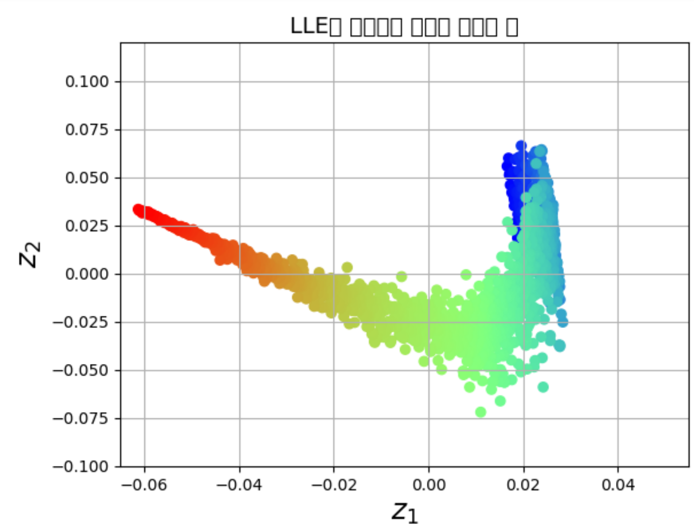
Program (1)

❖ **LAB11-3** LLE를 넘파이로만 구현해 보자

- https://colab.research.google.com/drive/1Z7rky4SzC4_qbbPk_aR6zp2mtj4LipEx

```
from sklearn.manifold import LocallyLinearEmbedding

# create sample dataset
X, color = make_a_roll(m)

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10,
            random_state=42)
lle.fit(X)

X_reduced = lle.transform(X)

plt.title("LLE를 사용하여 펼쳐진 스위스 롤", fontsize=14)
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=color,
cmap=plt.cm.hot)
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$z_2$", fontsize=18)
plt.axis([-0.065, 0.055, -0.1, 0.12])
plt.grid(True)

plt.show()
```





매니폴드 학습 (4)

❖ Isomap

- 각 샘플에 대해 가장 가까운 샘플을 연결하여 그래프를 만들어 **측지선** **geodesic** 거리의 대소 관계를 유지하며 차원을 축소하는 기법
- 측지선 거리는 그래프를 구성하는 노드들 사이의 최단경로를 계산하여 구할 수 있음

Isomap 알고리즘

1단계: 각 데이터 인스턴스에 대해 이웃 데이터를 결정한다.
  – k 개의 이웃을 찾는다.
2단계: 이웃 그래프를 구성한다.
  – 모든 점에 대해 1단계에서 얻은 이웃 데이터와 연결하는 간선을 만들어 전체 데이터를 하나의 그래프로 만든다.
  – 간선의 길이 혹은 가중치는 유클리드 거리로 한다.
3단계: 데이터들 사이의 측지선 거리, 즉 그래프에서의 최단경로를 구한다.
  – 플로이드-워셜Floyd-Warshall 알고리즘을 사용해 모든 쌍에 대한 최단 거리를 구하거나, 모든 점에 대해 다익스트라Dijkstra 알고리즘을 적용한다.
4단계: 앞에서 구한 측지선 거리를 유지하는 임베딩 공간을 찾는다.

매니폴드 학습 (5)

❖ t-SNE t-distributed Stochastic Neighbor Embedding

- 비슷한 데이터 인스턴스들끼리 가까운 거리를 유지하도록 하면서 더 낮은 차원의 임베딩 공간을 찾는 분석
- 고차원 공간에 있는 데이터의 군집을 시각화할 때 사용
- 통계적 근접점 임베딩 기법(SNE)에 t-분포를 적용하여 개선한 방법

t-SNE 알고리즘

1단계: 각 데이터 인스턴스에 대해 상호 유사도를 아래의 확률로 구한다.

$$- \; p_{j|i} = \frac{e^{-|\mathbf{x}_i - \mathbf{x}_j|^2/2\sigma_i^2}}{\sum_{k \neq i} e^{-|\mathbf{x}_i - \mathbf{x}_j|^2/2\sigma_i^2}}$$

⇐ Softmax 함수

2단계: 임의의 초기 임베딩 해 $\mathbf{Y}^{(0)} = [\mathbf{y}_1, \cdots, \mathbf{y}_m]$를 구한다.
3단계: 임베딩 해에 대해 상호 유사도를 구한다.

$$- \; q_{j|i} = \frac{e^{-|\mathbf{y}_i - \mathbf{y}_j|^2}}{\sum_{k \neq i} e^{-|\mathbf{y}_i - \mathbf{y}_j|^2}}$$

4단계: 쿨백-라이블리 발산값에 경사 하강법을 반복해 작용하여 최적의 $\mathbf{Y}$를 얻는다.

$$- \; \mathbf{Y} \leftarrow \mathbf{Y} - \eta \nabla_{\mathbf{Y}} \sum_i \mathcal{KL}(P_i \| Q_i)$$

Program (2)

- **LAB11-4** 매니폴드 학습을 이용한 차원 축소
  - https://colab.research.google.com/drive/1Z7rky4SzC4_qbbPk_aR6zp2mtj4LipEx

```
from sklearn import manifold

methods= {}
methods['LLE'] = manifold.LocallyLinearEmbedding(n_neighbors=10,
                                                  n_components=2)
methods['Isomap'] = manifold.Isomap(n_neighbors=10, n_components=2)
methods['t-SNE'] = manifold.TSNE(n_components=2)
```