

Preprocessing & Feature extraction

Hyunjoong Kim

soy.lovit@gmail.com

github.com/lovit

-
- 데이터를 테이블 형식으로 표현하는 경우가 많습니다.
 - 하나의 데이터 (row, data, point, instance, record, ...) 는 이를 설명하는 변수 (column, feature, column, variables) 로 구성되어 있습니다.

	datetime	temperature	humidity	wind speed	demand
row 1	2016. 06. 25	33	45	3.5	254
row 2	2016. 05. 21	23	21	4.3	761
row 3	2016. 05. 23	27	57	2.1	340
row 4

- 데이터의 종류는 다양합니다.

- 테이블 외에도 "이미지", "텍스트", "음성", "영상", "그래프" 등이 있습니다.
- 앞서 살펴본 선형회귀나 로지스틱 분류모델을 이용하기 위해서는 데이터를 벡터로 표현해야 합니다.

- 때로는 변수를 "정보력이 더 좋은 벡터"로 가공해야 합니다.

- 명목형 변수는 dummy variables 로 치환이 가능합니다.

- 범위가 다른 연속형 변수들은 스케일링 (scaling) 이 필요할 때도 있습니다.

- 연속형 변수를 명목형 변수로 묶을 수도 있습니다 (binning)

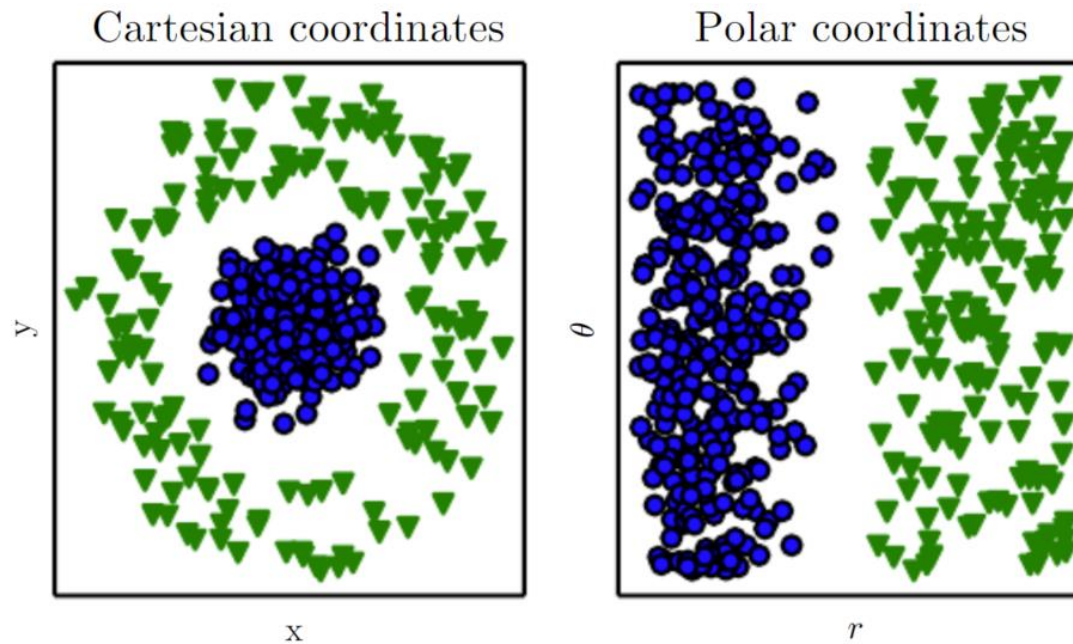
	categorical variable			different scaled numerical variable	
	Survived	Pclass	Sex	Age	Fare
PassengerId					
1	0	3	male	22.0	7.2500
2	1	1	female	38.0	71.2833
3	1	3	female	26.0	7.9250
4	1	1	female	35.0	53.1000
5	0	3	male	35.0	8.0500
6	0	3	male	NaN	8.4583
7	0	1	male	54.0	51.8625
8	0	3	male	2.0	21.0750
9	1	3	female	27.0	11.1333
10	1	2	female	14.0	30.0708

Feature engineering

- 벡터 공간을 이용하는 머신러닝 알고리즘이 잘 작동하기 위해서는 과업에 적합한 형태로 정보를 가공할 필요가 있습니다.
 - Feature engineering 은 정보력이 좋은 변수들을 가공하여 과업의 성능을 향상하기 위한 일련의 과정입니다.
 - feature generation / vectorizing : 새로운 변수를 생성하거나 데이터를 벡터로 표현
 - feature selection : 변수 중 필요한 것을 선택
 - feature extraction/transformation : 주어진 변수를 이용하여 새로운 변수를 생성

Feature engineering

- 벡터 공간을 이용하는 머신러닝 알고리즘이 잘 작동하기 위해서는 과업에 적합한 형태로 정보를 가공할 필요가 있습니다.

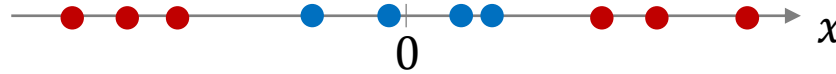


직교좌표계에서 선형 모델로 두 클래스를 구분할 수 없지만,
동일한 데이터를 극좌표계로 표현하면 구분이 가능하기도 합니다.

Feature engineering

- 벡터 공간을 이용하는 머신러닝 알고리즘이 잘 작동하기 위해서는 과업에 적합한 형태로 정보를 가공할 필요가 있습니다.

Linear inseparable with x



Linear separable with (x, x^2)

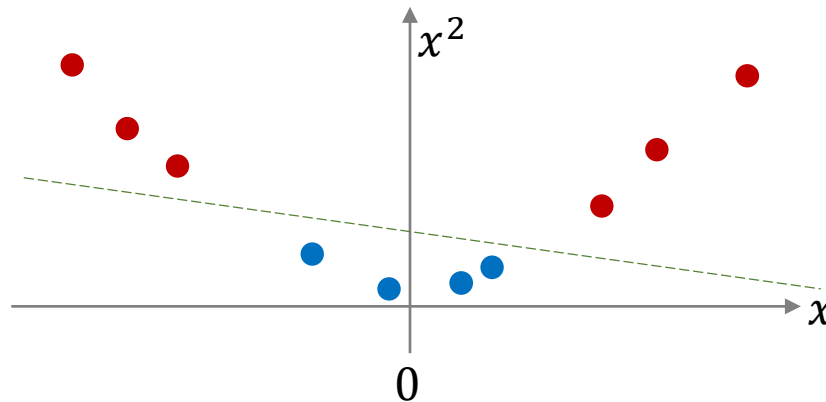


Image as vector

- 이미지는 그 자체로 벡터로 인식될 수 있습니다.

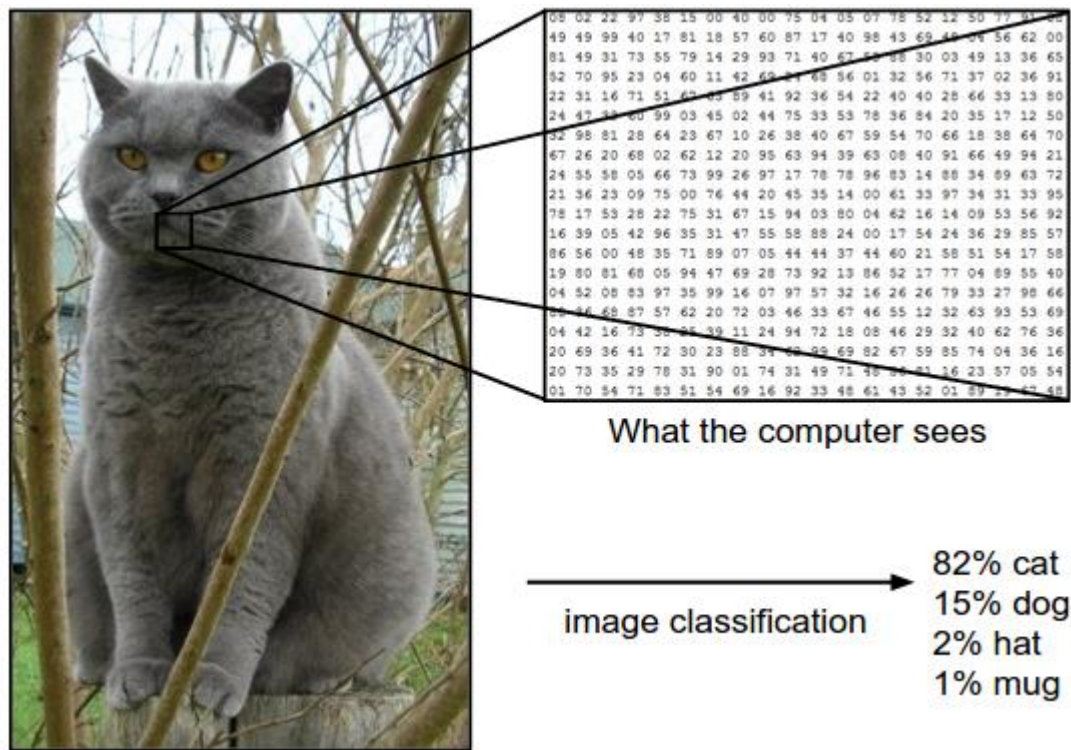


Image as vector

- 각 행렬을 flatten 한 다음 이어붙이면 벡터로 표현됩니다.



$$(640 \times 480) \\ = 307,200$$

$$(640 \times 480) \\ = 307,200$$

$$(640 \times 480) \\ = 307,200$$

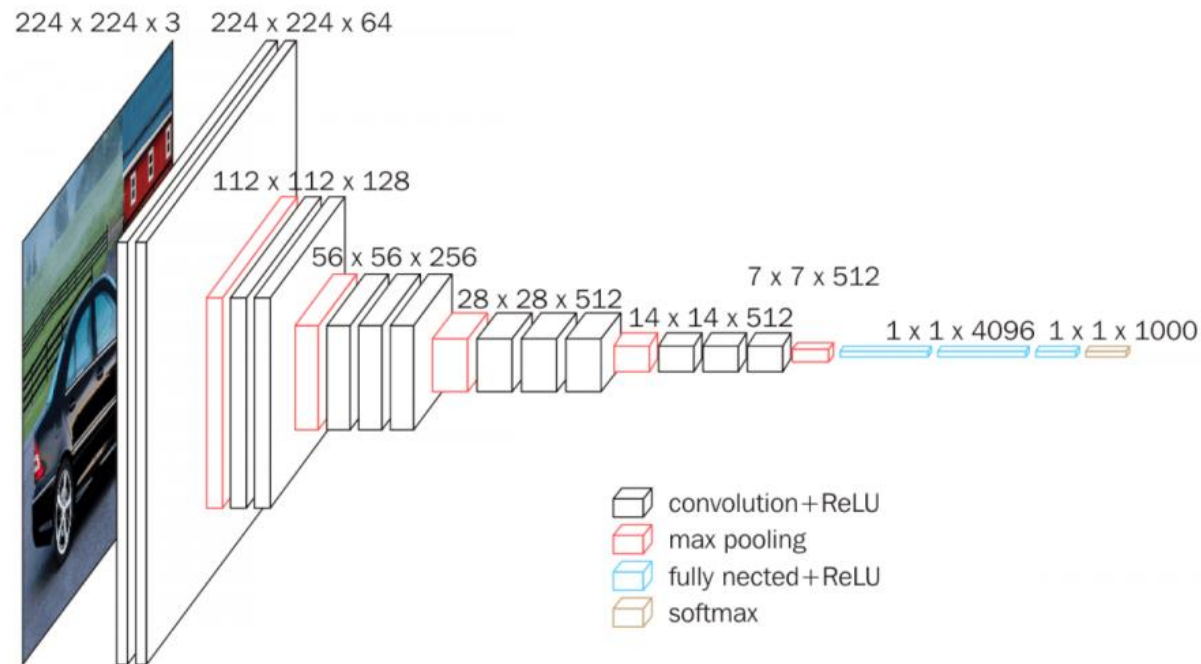


→
concatenation



Image as vector

- 이미지는 locality 를 표현하는 더 좋은 벡터 표현들이 있습니다.



Text as vector

- 문서/문장은 등장한 단어의 빈도를 이용하여 벡터로 표현할 수 있습니다.

	기계	학습	은	텍스트	마이닝	는
Doc 1	3	2	5	0	0	0
Doc 2	0	0	0	3	5	5
...

Doc 1 = [(0, 3), (1, 2), (2, 5)]

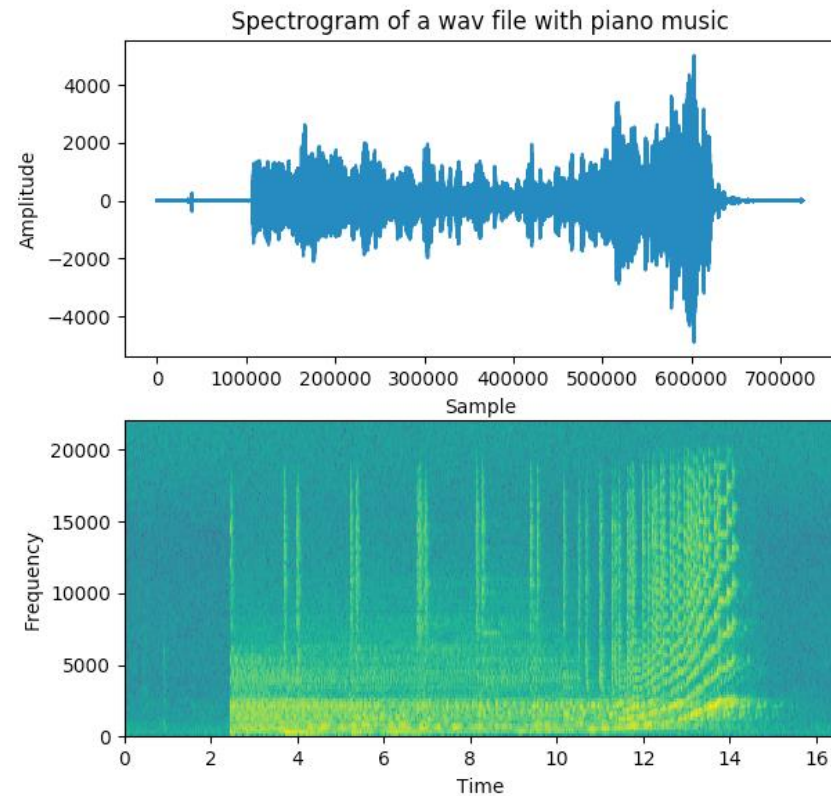
Doc 2 = [(3, 3), (4, 5), (5, 5)]



	0	1	2	3	4	5
Doc 1	3	2	5	0	0	0
Doc 2	0	0	0	3	5	5
...

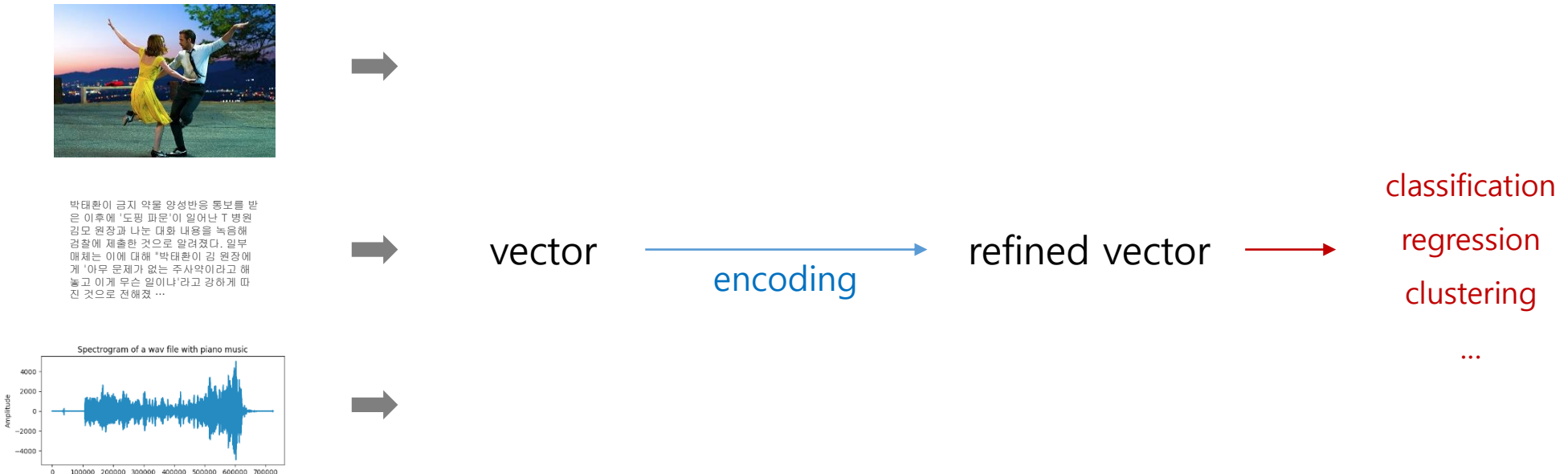
Sound as vector

- 음성은 한 시점 t 에서의 주파수 별 에너지량을 이용하여 시계열 형식의 벡터로 표현할 수 있습니다.



Feature engineering

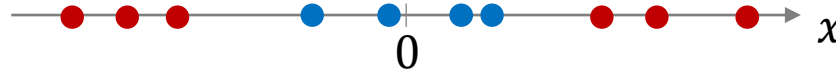
- 이미지/텍스트/음성 등 복잡한 데이터는 특징을 더 잘 표현하는 벡터로 변환하기 위하여 인코딩 과정을 거치는 경우가 많습니다.
- 데이터마다 적합한 인코딩 과정은 대체로 다릅니다.



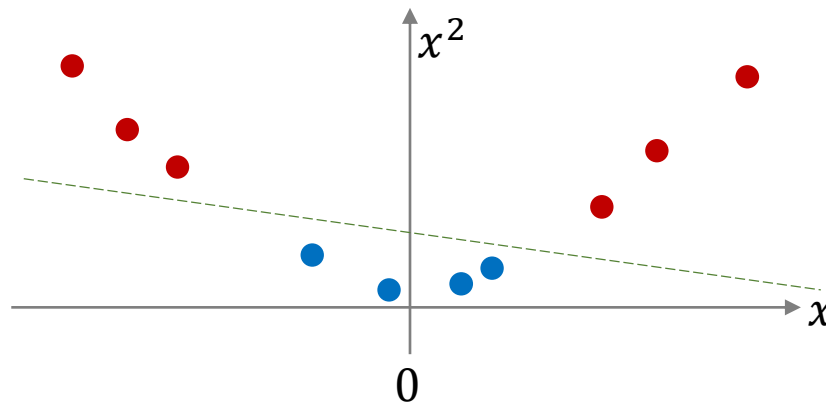
Feature engineering

- 이미지/텍스트/음성 등 복잡한 데이터는 특징을 더 잘 표현하는 벡터로 변환하기 위하여 인코딩 과정을 거치는 경우가 많습니다.
 - 공통된 목적은 각 과업을 더 쉬운 문제로 만들기 위함입니다.

Linear inseparable with x



Linear separable with (x, x^2)



Feature scaling

- 변수 X^j 의 크기/분포를 조절하는 과정입니다.
 - centering : $X^j \leftarrow X^j - \text{mean}(X^j)$
 - Standardization : $X^j \leftarrow \frac{X^j - \text{mean}(X^j)}{\sigma_{X^j}}$
 - min-max scaling : 특정 범위내에 값이 존재하도록 특정합니다.
 - $X^j \leftarrow \alpha_j \times (X^j - \beta_j)$
 - [0, 1] scaling : $X^j \leftarrow \frac{X^j - \min(X^j)}{\max(X^j) - \min(X^j)}$
 - [-1, 1] scaling : $X^j \leftarrow 2 \times \left(\frac{X^j - \min(X^j)}{\max(X^j) - \min(X^j)} - 0.5 \right)$

Feature scaling

- 변수 x^j 의 크기/분포를 조절하는 과정입니다.
 - 변수 별 scale 이 다를 경우, 점들 간 거리를 정의할 때 특정 변수의 영향력이 큼니다.
 - 첫번째 변수의 범위는 [150, 200], 두번째 변수의 범위는 [20.5, 22.5] 라면 두번째 변수가 변화량이 큰 경우이지만, 거리 척도가 이를 반영하지 못합니다.

$$d = |(170, 22) - (160, 21)|_2$$

- 변수 별 정규화를 통하여 두번째 변수의 영향력이 크도록 벡터를 수정합니다.

$$d = |(0.5, 0.75) - (0.45, 0.25)|_2$$

Feature scaling

- 변수 X^j 의 크기/분포를 조절하는 과정입니다.
 - 변수 별 scale 이 다를 경우, 내적을 취할 때 특정 변수의 영향력이 큼니다.
 - $\cos(u, v) = \frac{u \cdot v}{|u|_2 \cdot |v|_2} = \left(\frac{u}{|u|_2} \right) \cdot \left(\frac{v}{|v|_2} \right)$
 - $\cos((3, 0, 4), (4, 3, 0)) = \frac{12}{5 \times 5} = 0.48$
 - $\cos((300, 0, 4), (400, 3, 0)) \cong 1$

Feature scaling

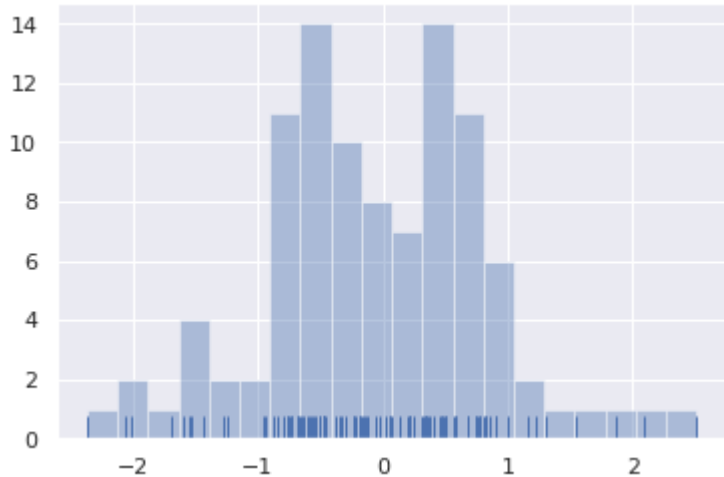
- 변수 x^j 의 크기/분포를 조절하는 과정입니다.
 - L2 regularization 은 변수의 스케일에 영향을 받습니다. 또한 경사하강법을 이용할 때 변수의 스케일에 따라 gradient 의 값이 달라집니다.
 - 뉴럴 네트워크 모델들은 변수 별 스케일이 다르면 학습이 잘 이뤄지지 않기도 합니다.

$$\text{NLL} : \sum_{i=1}^n \sum_{j=1}^K I(y_i = j) \left(-x_i^T \beta_j + \log \sum_{j=1}^K \exp(x_i^T \beta_j) \right)$$

$$\frac{\partial l(x_i, y_i)}{\partial \beta_{jq}} = x_{iq} \frac{\exp(x_i^T \beta_j)}{\sum_{j=1}^K \exp(x_i^T \beta_j)} - x_{iq} = \mathbf{x}_{iq}(\pi_i - 1)$$

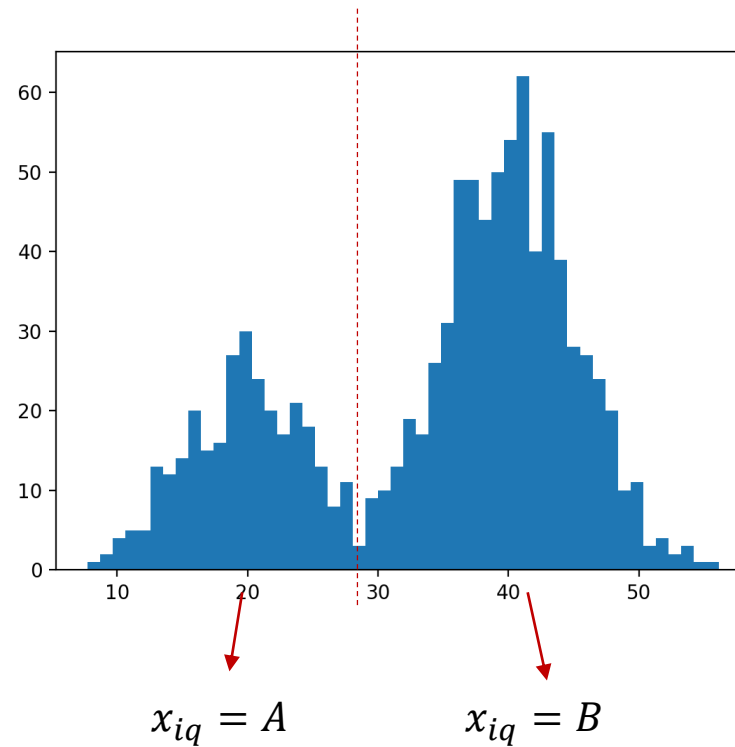
Binning

- 연속형 변수라 하더라도 유형별로 구분할 때가 있습니다. 이때는 범위를 지정하여 연속형 변수를 이산형으로 변형합니다.
 - (예시) if age < 15 children, else adult
 - (예시) histogram



Binning

- Multimodal distribution 의 경우에도 standardization 보다 변수의 특징을 잘 표현할 수 있습니다.



Feature extraction

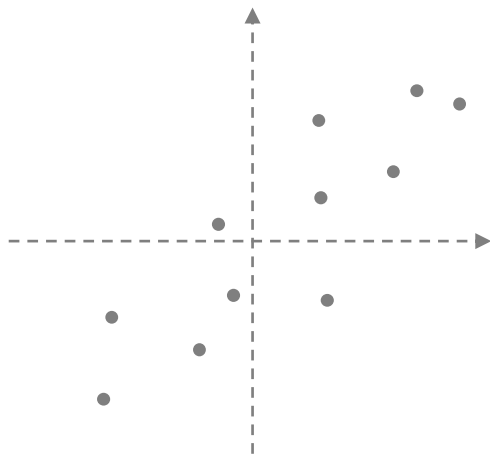
- 변수 추출 (extraction) 은 중복된 변수를 제거하는데 이용될 수 있습니다.
 - PCA 는 경향이 비슷한 여러 features 를 하나의 새로운 feature 로 묶을 수 있습니다.
 - Features 의 개수가 줄어들기 때문에 모델의 regularization cost 가 감소합니다.

Feature extraction

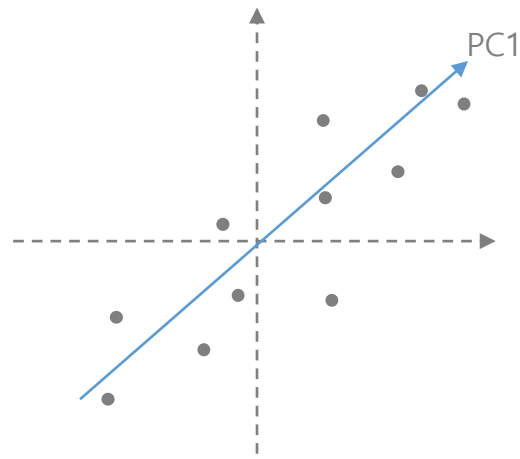
- 변수 추출은 고차원의 데이터를 2, 3차원으로 압축하여 고차원을 시각적으로 확인하는데 이용될 수 있습니다.
 - t-SNE, UMAP 은 새로운 features 를 만드는 용도가 아닙니다.
 - Input space 에서 가까이 위치한 점들을 2 차원 (임베딩) 공간에서도 가깝게 위치시켜 고차원 공간의 모습을 짐작할 수 있도록 도와줍니다.

Principal Component Analysis (PCA)

- p 차원 X 의 **방향적 분포를 잘 설명**하는 새로운 ($q \leq p$) 차원 직교 좌표를 학습합니다.
- 데이터는 평균이 원점이라 가정합니다



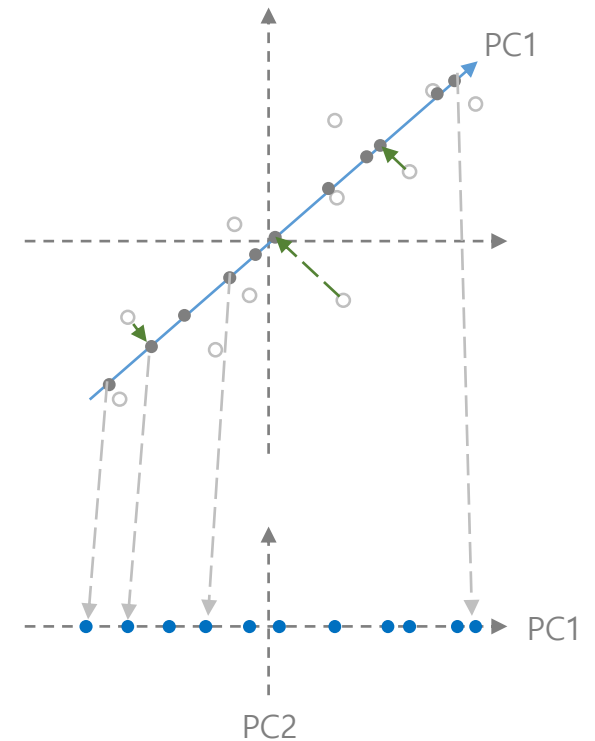
input data



find the most principal component (PC1)

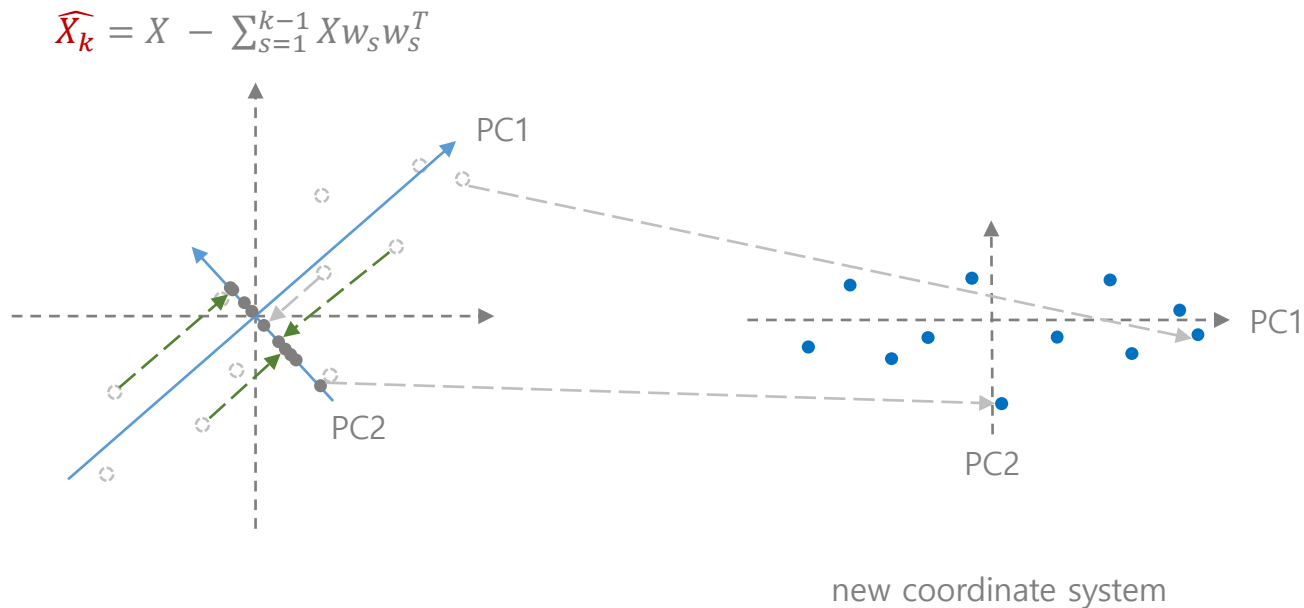
$$w_1 = \underset{|w|=1}{\operatorname{argmax}} \left\{ \frac{w^T X^T X w}{w^T w} \right\}$$

$w^T X^T X w$ (Covariance) 이 가장 큰 방향 벡터 w 를 찾습니다



Principal Component Analysis (PCA)

- PC1 을 제외한 값에서 Covariance 가 가장 큰 방향 벡터 w_{i+1} 를 찾습니다.



$$w_k = \underset{|w|=1}{\operatorname{argmax}} \left\{ \frac{w^T \widehat{X}_k^T \widehat{X}_k w}{w^T w} \right\}$$

$w^T \widehat{X}_k^T \widehat{X}_k w$ (Covariance) 이 가장 큰 방향 벡터 w 를 찾습니다

Principal Component Analysis (PCA)

- PCA 에서 variance 가 큰 방향벡터를 탐색하는 방법으로 Singular Vector Decomposition (SVD) 가 이용됩니다.
 - PCA 는 데이터 평균이 원점이라 가정하기 때문에, 주어진 학습데이터에서 평균값을 뺀 뒤, SVD 를 적용합니다.

Singular Value Decomposition

- SVD 는 행렬 A 를 다음처럼 세 개의 행렬로 분해합니다.

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

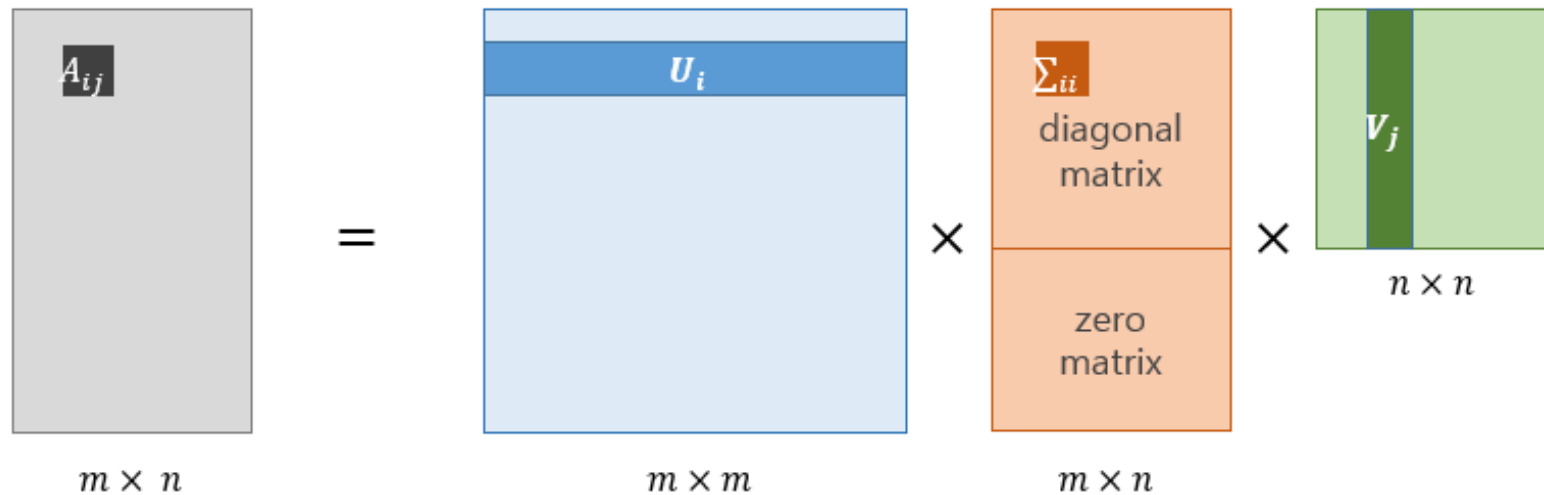
The diagram illustrates the Singular Value Decomposition (SVD) of matrix A . Matrix A (dimensions $m \times n$) is decomposed into three matrices:

- U (dimensions $m \times m$): A light blue square matrix with a dark blue horizontal band labeled U_i .
- Σ (dimensions $m \times n$): An orange rectangle divided into two sections: a top section labeled Σ_{ii} (diagonal matrix) and a bottom section labeled zero matrix.
- V^T (dimensions $n \times n$): A light green rectangle with a dark green vertical band labeled V_j .

The decomposition is represented by the equation $A = U \Sigma V^T$.

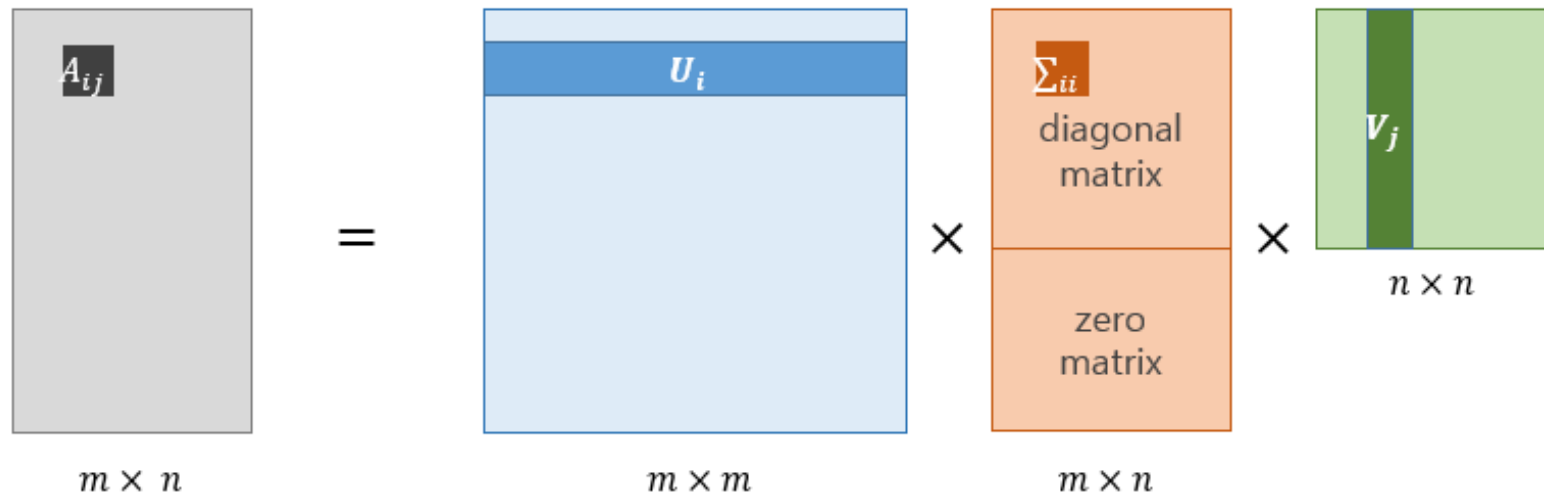
Singular Value Decomposition

- U, V 는 각각 orthonormal matrix 입니다.
 - 각 행과 열의 벡터의 크기는 1 이며 (normal)
 - $U_i \times U_j^T = 0, U_i \cdot U_i^T = 1$ 입니다.
 - V 도 동일합니다.



Singular Value Decomposition

- Σ 는 PCA 에서의 각 component 별 중요도 (variance proportion) 입니다.
 - $\min(m, n)$ 개의 components 에 대한 중요도로 해석할 수 있습니다.



Singular Value Decomposition

- truncated SVD 는 Σ 에서 중요한 k 개의 components 만 이용합니다.
- U, V 를 k 차원으로 축소한 것으로 해석할 수 있습니다.

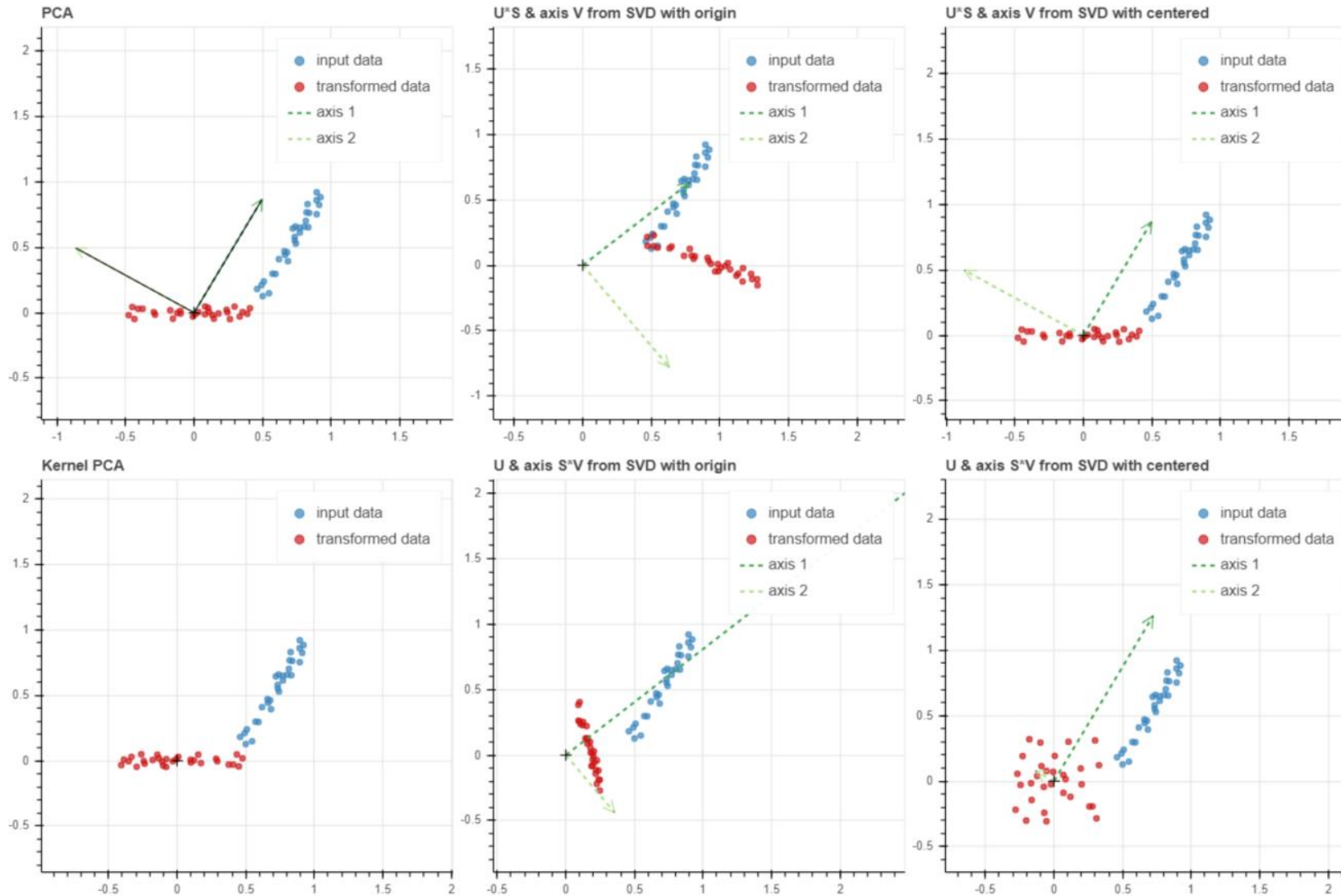
The diagram illustrates the truncated SVD decomposition of a matrix A_{ij} (size $m \times n$) into three components: U_i (size $m \times k$), Σ_{ii} (size $k \times k$), and V_j (size $k \times n$). The matrix A_{ij} is shown as a gray rectangle. The matrix U_i is shown as a light blue rectangle with a darker blue header. The matrix Σ_{ii} is shown as an orange rectangle. The matrix V_j is shown as a light green rectangle with a dark green header. The decomposition is represented by the equation: $A_{ij} = U_i \times \Sigma_{ii} \times V_j$.

$$\begin{matrix} \text{Gray box } A_{ij} \\ m \times n \end{matrix} = \begin{matrix} \text{Blue box } U_i \\ m \times k \end{matrix} \times \begin{matrix} \text{Orange box } \Sigma_{ii} \\ k \times k \end{matrix} \times \begin{matrix} \text{Green box } V_j \\ k \times n \end{matrix}$$

Singular Value Decomposition

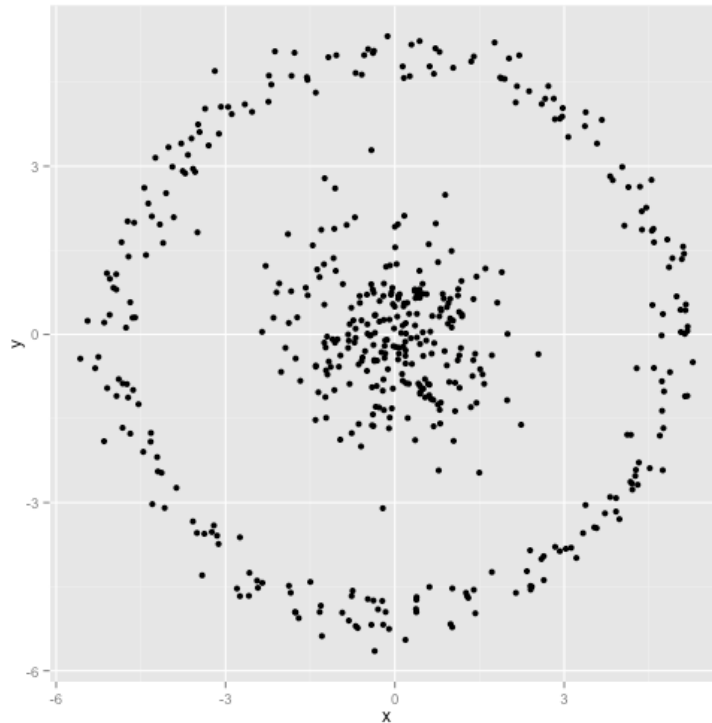
- Top principal components 에서 중요한 정보들은 추출이 되었기 때문에 정보의 손실이 적습니다.
- 비슷한 points 에서 함께 등장한 features 의 정보가 각 components 에 저장됩니다.

SVD vs PCA



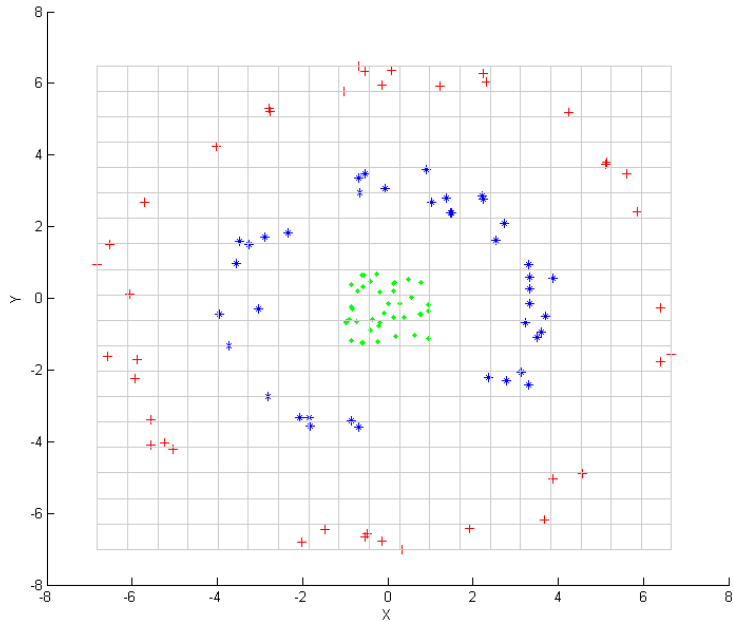
Principal Component Analysis (PCA)

- PCA 는 데이터의 방향적인 경향이 있을 때 잘 작동합니다.
- 데이터의 경향이 방향적이지 않는 경우는 주요 축을 찾을 수 없습니다.

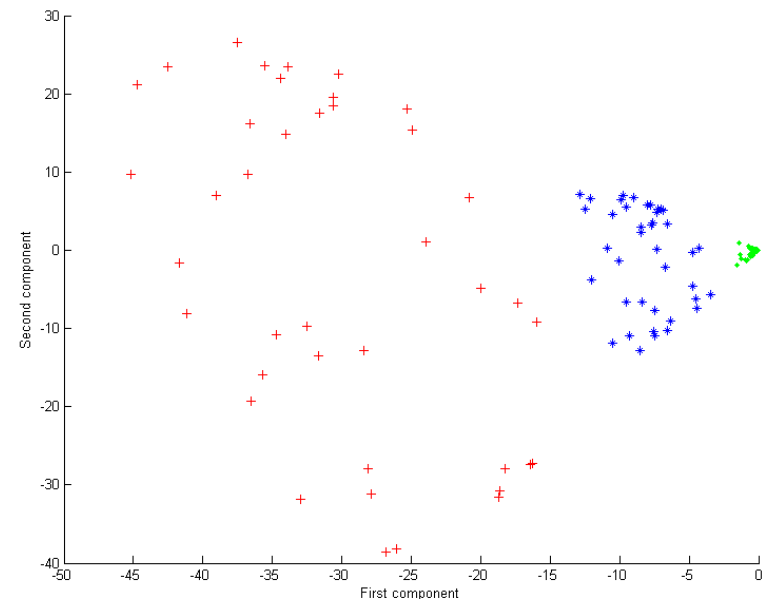


Kernel Principal Component Analysis (KPCA)

- Kernel PCA 는 분포의 경향을 보존하는 새로운 직교 좌표를 학습합니다.
 - 데이터의 개수가 n 일 때, n 보다 작은 q 차원의 공간을 학습합니다.
- $w_1 = \operatorname{argmax}_{||w||=1} \left\{ \frac{w^T \mathbf{K}(\mathbf{X}, \mathbf{X}) w}{w^T w} \right\}$



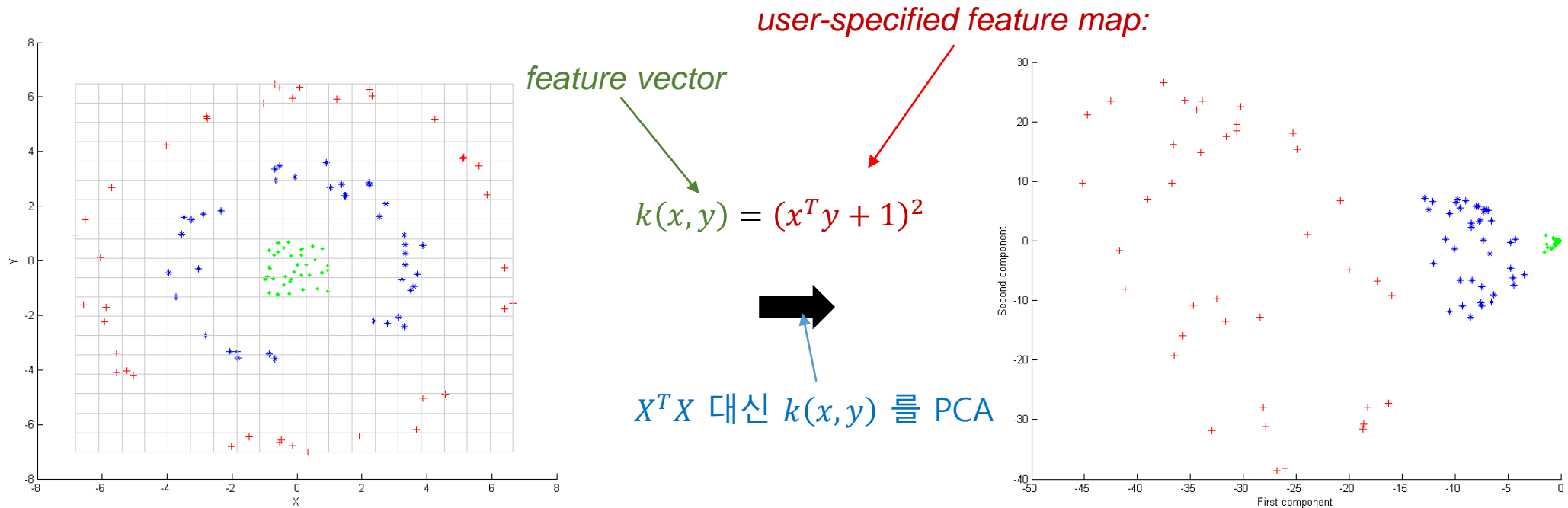
$$k(x, y) = (x^T y + 1)^2$$



Kernel Principal Component Analysis (KPCA)

- Kernel 은 데이터 간의 유사성 (proximity) 으로 해석할 수 있습니다.

*For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into **feature vector** representations via a **user-specified feature map**:*



Kernel Principal Component Analysis (KPCA)

- Kernel 은 데이터 간의 유사성 (proximity) 으로 해석할 수 있습니다.
 - n 개의 데이터를 유사도 벡터로 representation 을 변환한 뒤, PCA 를 적용한 것과 같습니다.
 - $n \times n$ 크기의 kernel matrix 는 점들 간의 유사도 행렬과 같습니다.
 - “유사한 점들이 비슷한 점들”은 kernel PCA 변환 뒤에도 유사한 벡터를 지닙니다.

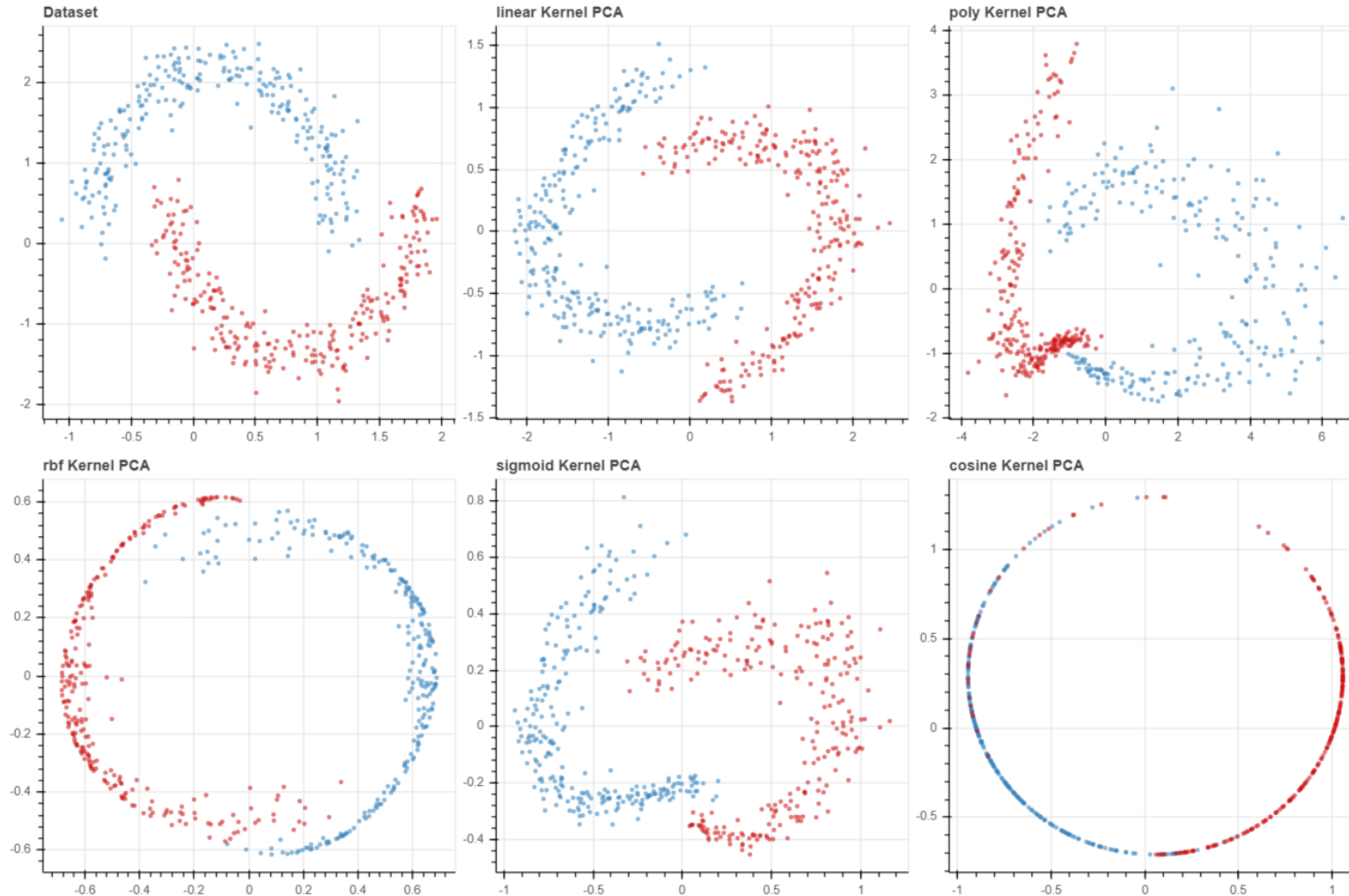
Kernel Principal Component Analysis (KPCA)

- 일반적으로 이용되는 kernels 은 아래와 같습니다.

kernel name	$K(x, y)$
Linear	$x^T y + c$
Polynomial	$(x^T y + a)^d$
RBF kernel	$\exp(-\beta x - y _2^2)$
Sigmoid	$\tanh(x^T y + c)$
Cosine	$\cos(x, y)$
etc ...	

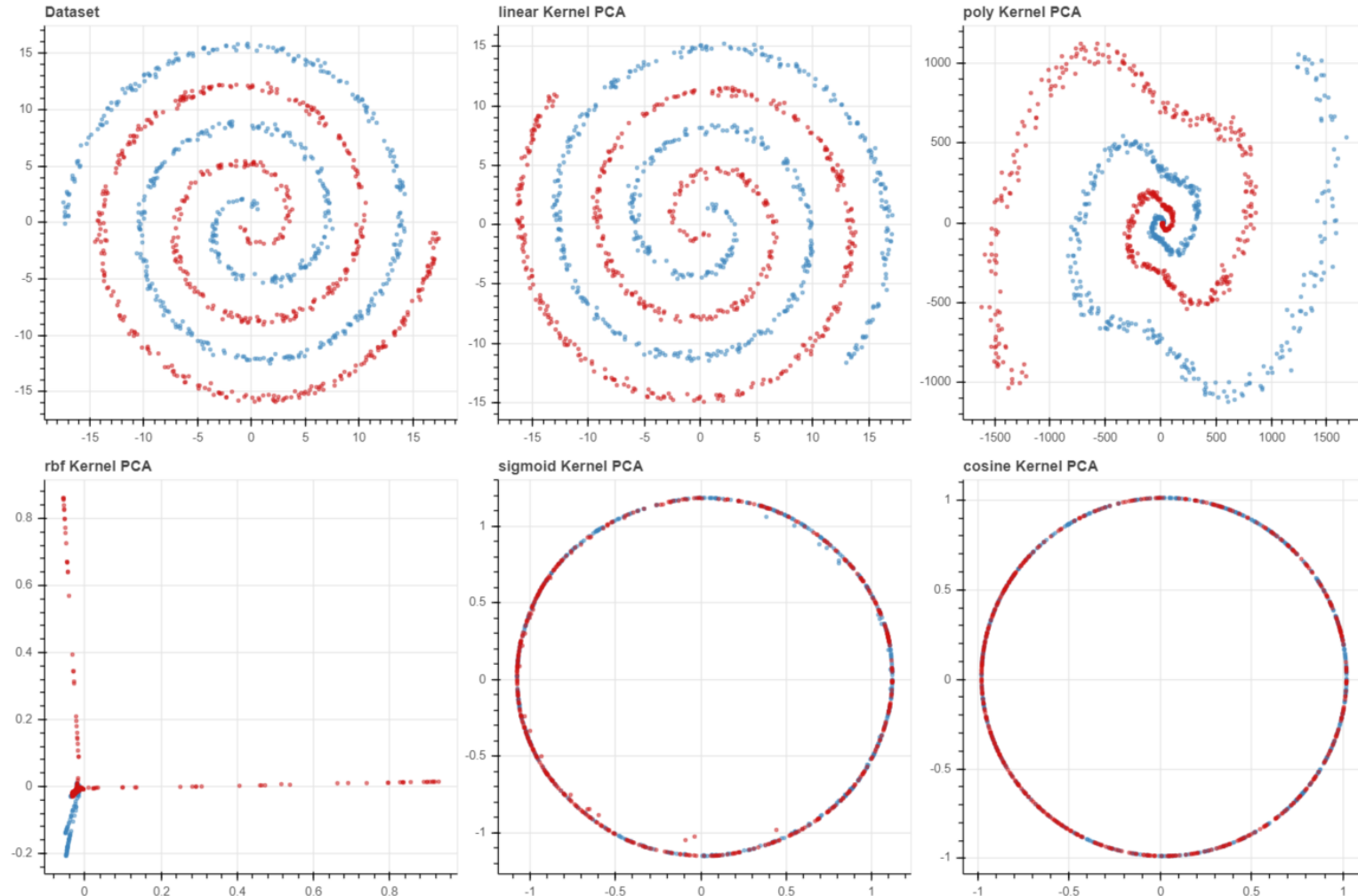
Kernel Principal Component Analysis (KPCA)

- Kernel 의 종류에 따라 같은 데이터도 서로 다른 features 로 변환됩니다.



Kernel Principal Component Analysis (KPCA)

- Kernel 의 종류에 따라 같은 데이터도 서로 다른 features 로 변환됩니다.



t-Stochastic Neighbor Embedding (t-SNE)

- t-SNE 는 SNE 알고리즘의 문제를 개선한 방법이며, 이후 LargeVis, UMAP 으로 발전합니다.
 - SNE → t-SNE → LargeVis → UMAP 의 목적은 모두 input space 에서 가까운 두 점 x_i, x_j 가 embedding space 에서 가깝고, input space 에서 먼 두 점은 embedding space 에서도 멀리 떨어지도록 좌표값 y_i, y_j 를 학습하는 것입니다.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

$$v_{j|i} = \exp\left(-\beta_i |x_i - x_j|_2^2\right)$$

$$p_{j|i} = \frac{v_{j|i}}{\sum_{k \neq i} v_{k|i}}, p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

$$w_{ij} = \frac{1}{1 + |y_i - y_j|_2^2}$$

$$q_{ij} = \frac{w_{ij}}{\sum_{p, q \neq p} w_{pq}}$$

$$C_{tSNE} = \sum_{i, j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Embedding space 에서의 점들 간 유사도를 재정의합니다.
학습이 더욱 안정적으로 이뤄집니다.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

$$C_{tSNE} = \sum_{i,j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

$$\frac{dC_{tSNE}}{dy_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) (y_i - y_j) \times \frac{1}{1 + |y_i - y_j|_2^2}$$

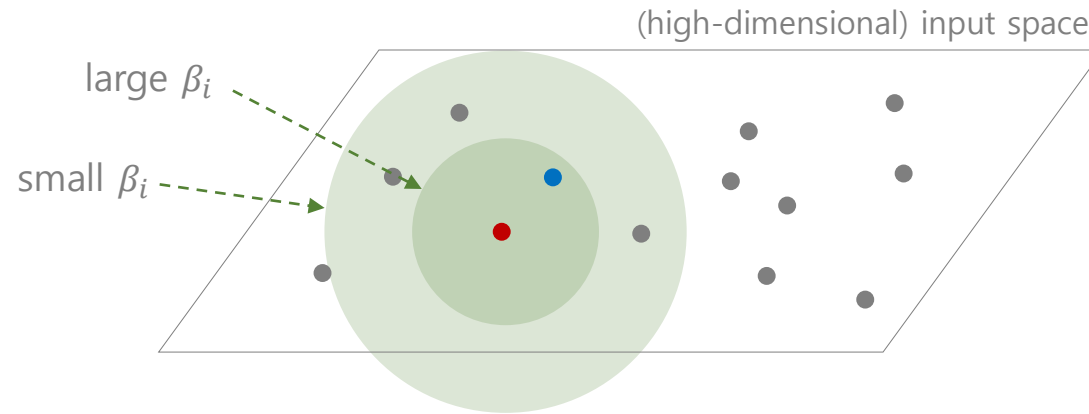
q_{ij} 가 크거나 p_{ij} 가 큰 점에 집중적으로 학습합니다.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Finding β_i for each point i to meet certain perplexity with $v_{j|i}$

$$v_{j|i} = \exp\left(-\beta_i |x_i - x_j|_2^2\right)$$

$$\text{Perplexity} = 2^{H(p)}, 2^{\text{entropy}}$$



- β_i 에 의하여 $v_{j|i}$ 분포가 변화합니다. β_i 가 크면 가까운 점들만 큰 $v_{j|i}$ 가 만들어집니다.

-
- Entropy 는 확률 분포의 불확실성을 정의하는 방법입니다.

- $entropy(p) = -\sum_p p \log p$

- $P(x) = \{a: 0.3, b: 0.4, c: 0.3\}$ 에서 하나를 선택할 때, 결과를 예상하기 어렵습니다.

$$entropy(P(x)) = -(0.3 \times \log 0.3 + 0.4 \times \log 0.4 + 0.3 \times \log 0.3)$$

-
- Entropy 는 확률 분포의 불확실성을 정의하는 방법입니다.

- $entropy(p) = -\sum_p p \log p$

- $P(x) = \{a: 0.99, b: 0.005, c: 0.005\}$ 에서 하나를 선택하면 거의 a 일 것입니다.

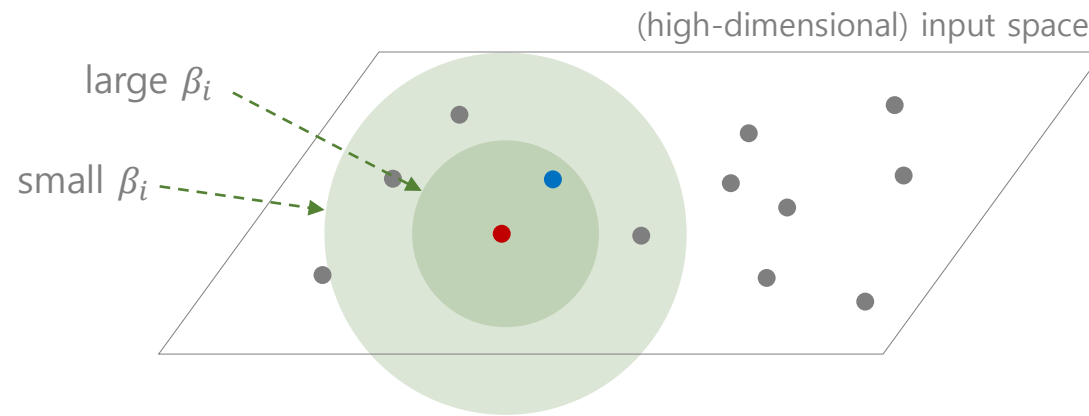
- $entropy(P(x)) = -(0.99 \times \log 0.99 + 0.005 \times \log 0.005 + 0.005 \times \log 0.005) = 0.063$

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Finding β_i for each point i to meet certain perplexity with $v_{j|i}$

$$v_{j|i} = \exp\left(-\beta_i \|\mathbf{x}_i - \mathbf{x}_j\|_2^2\right)$$

$$\text{Perplexity} = 2^{H(p)}, 2^{\text{entropy}}$$



- 사용자에게 의하여 특정 perplexity 가 주어지면 이를 맞추기 위해 β_i 를 조절합니다.
Perplexity 가 작을수록 주위의 (local) 점들만 집중하여 $v_{j|i}$ 가 큰 값이 만들어집니다.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- $v_{j|i} = \exp\left(-\beta_i |x_i - x_j|_2^2\right)$ 는 안정적인 임베딩 결과를 도출합니다.
 - 데이터 분포와 거리 척도에 관계없이 대체로 비슷한 $p_{j|i}$ 를 생성합니다.
 - 데이터에 관계없이 비슷한 input p_{ij} 가 만들어지기 때문에 비슷한 (안정적인) 임베딩 결과가 학습됩니다.
- 하지만 β_i 가 지나치게 크면 점들 간 유사도의 변별력이 사라집니다.
 - 임베딩이 제대로 학습되지 않습니다.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

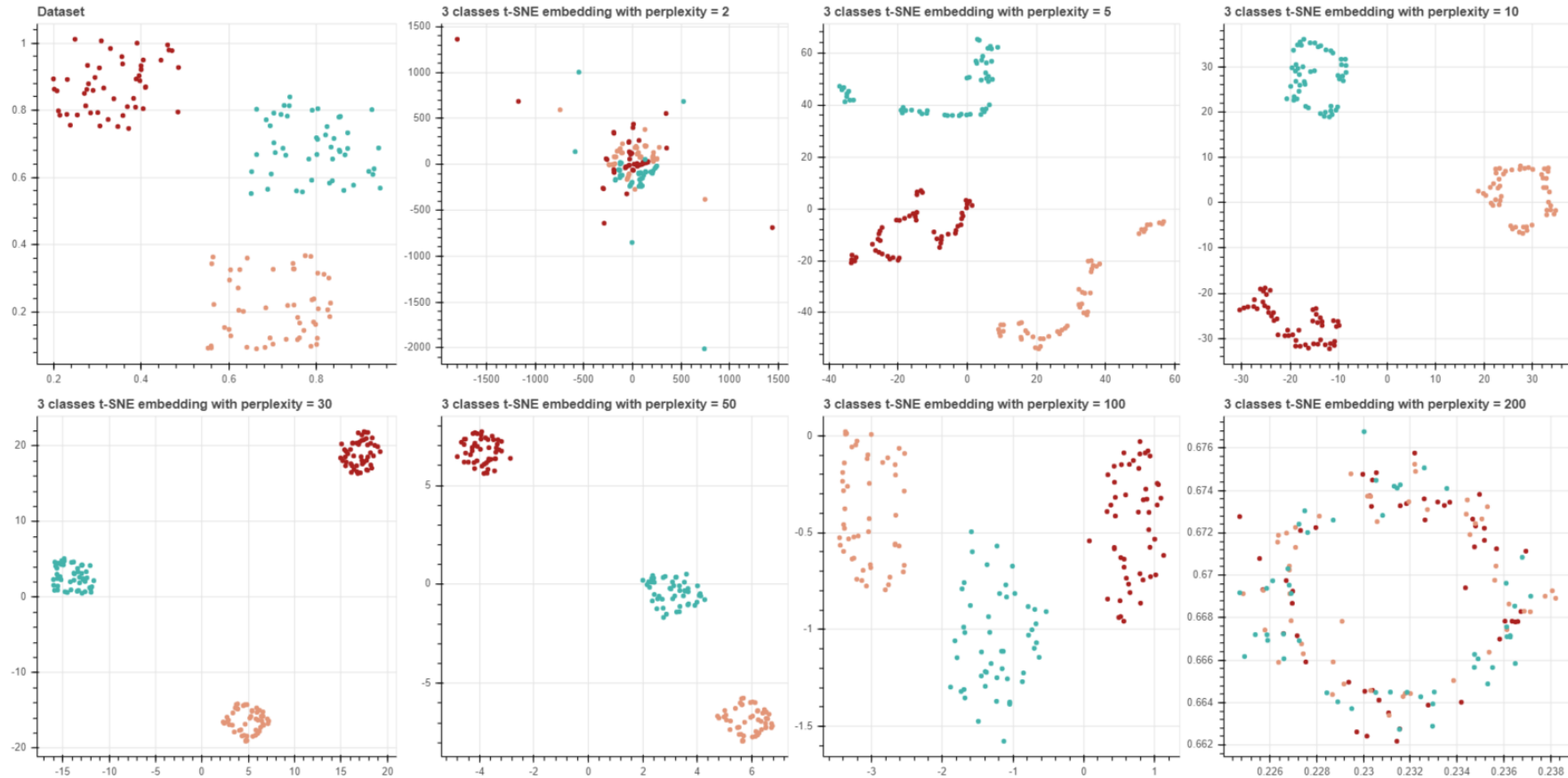
- $P_{j|i} \gg 0$ 인 점들의 개수는 perplexity 에 의하여 조절됩니다.
 - Perplexity 가 클수록 더 많은 점들을 고려합니다.
 - 그 값이 지나치게 크면 모든 점들 간의 $P_{j|i}$ 가 비슷한 값이 되어 점들 간 유사도가 uniform 하게 변환됩니다. 반대로 지나치게 작은 값은 한 두 개의 이웃만 고려합니다.
 - 적은 수의 데이터를 학습할 때에는 perplexity 를 신경써야 합니다.

`sklearn.manifold.TSNE`

```
class sklearn.manifold.TSNE(n_components=2, perplexity=30.0, early_exaggeration=4.0,  
learning_rate=1000.0, n_iter=1000, n_iter_without_progress=30, min_grad_norm=1e-07, metric='euclidean',  
init='random', verbose=0, random_state=None, method='barnes_hut', angle=0.5) ¶ \[source\]
```

t-Distributed Stochastic Neighbor Embedding (t-SNE)

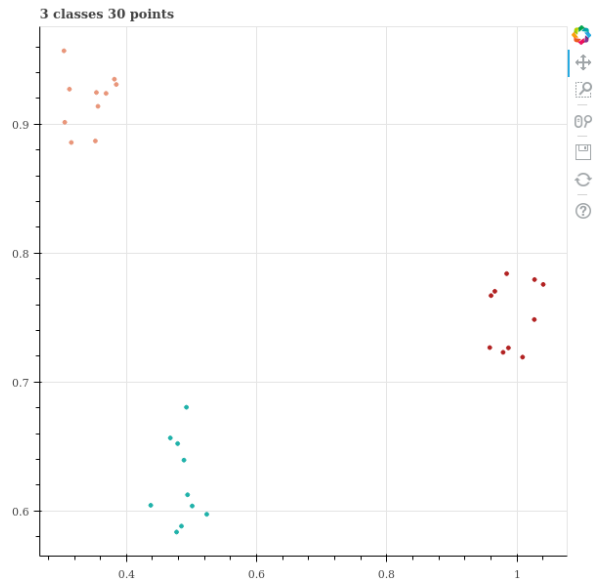
- 데이터의 개수에 따라 perplexity 를 조절해야 합니다.



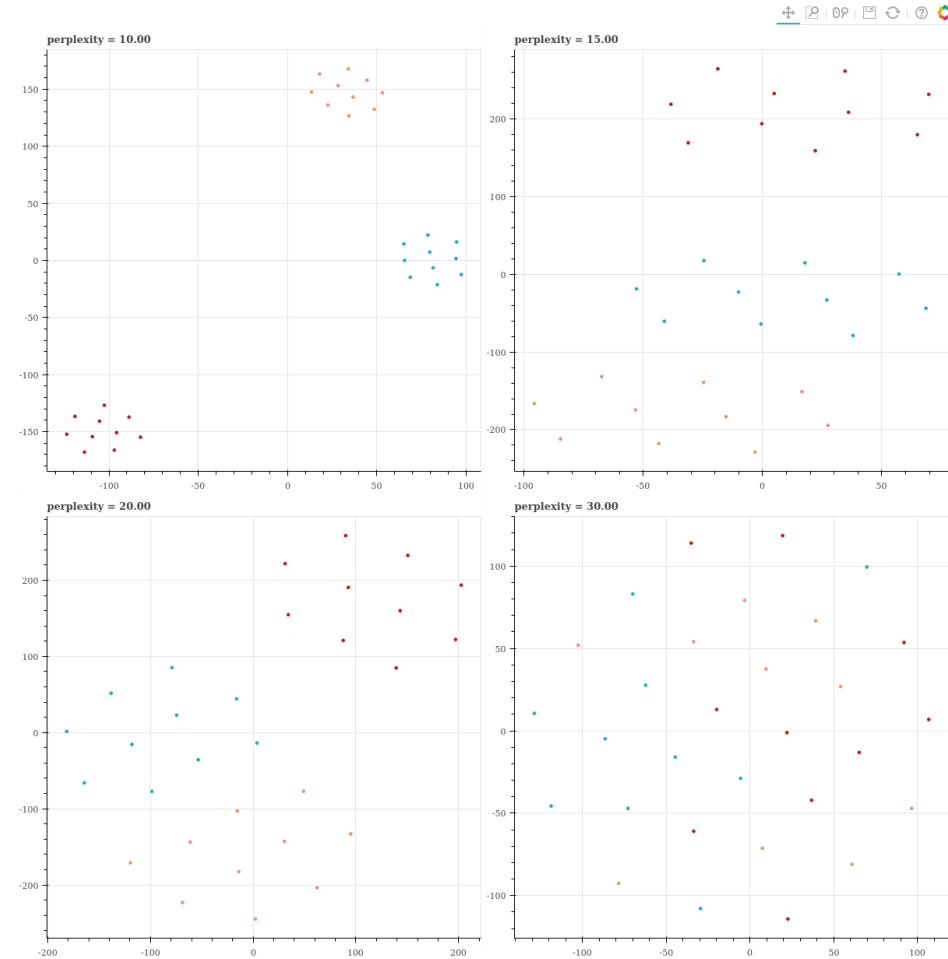
150 points 의 perplexity 에 따른 t-SNE 변화

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- 데이터의 개수에 따라 perplexity 를 조절해야 합니다.



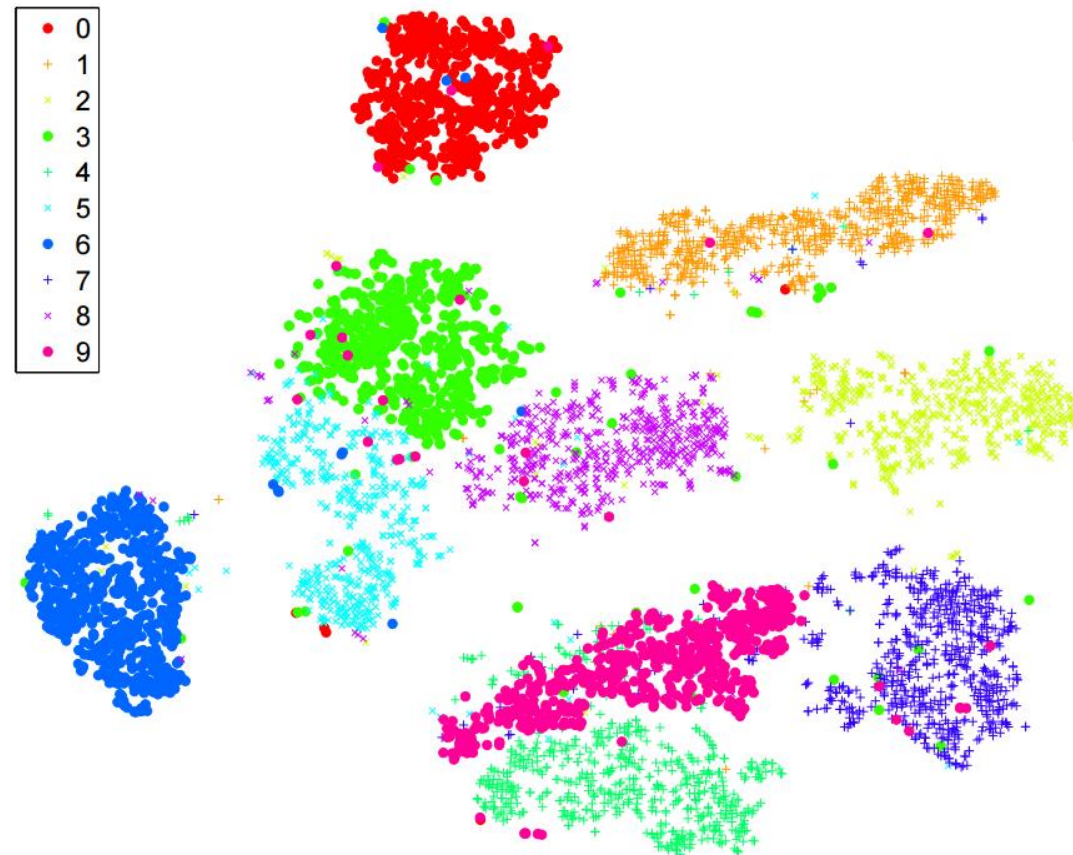
raw data, 30 points



embedding results

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- 손글씨 숫자 데이터 (MNIST)의 시각화 예시



0	4	7	9	2	1	3	1	4	3
5	3	6	1	7	2	8	6	9	4
0	9	7	1	2	4	3	2	7	3
8	6	9	0	5	6	0	7	6	1
8	7	9	3	9	8	5	9	3	3
0	7	4	9	8	0	9	4	7	4
4	6	0	4	5	6	1	0	0	1
7	1	6	3	0	2	1	1	7	9
0	2	6	7	8	3	9	0	4	6
7	4	6	8	0	7	8	3	1	5

t-Distributed Stochastic Neighbor Embedding (t-SNE), 한계점

- $w_{ij} = \frac{1}{1+|y_i - y_j|_2^2}$ $q_{ij} = \frac{w_{ij}}{\sum_{p,q \neq p} w_{pq}}$ 이므로 gradient descent 의 매 iteration step 마다 정규화를 포함한 q_{ij} 를 재계산 해야 합니다. 이 계산비용은 $O(n^2)$ 으로 큰 데이터에 t-SNE 를 적용하지 못하게 만듭니다.

Barnes hut t-SNE

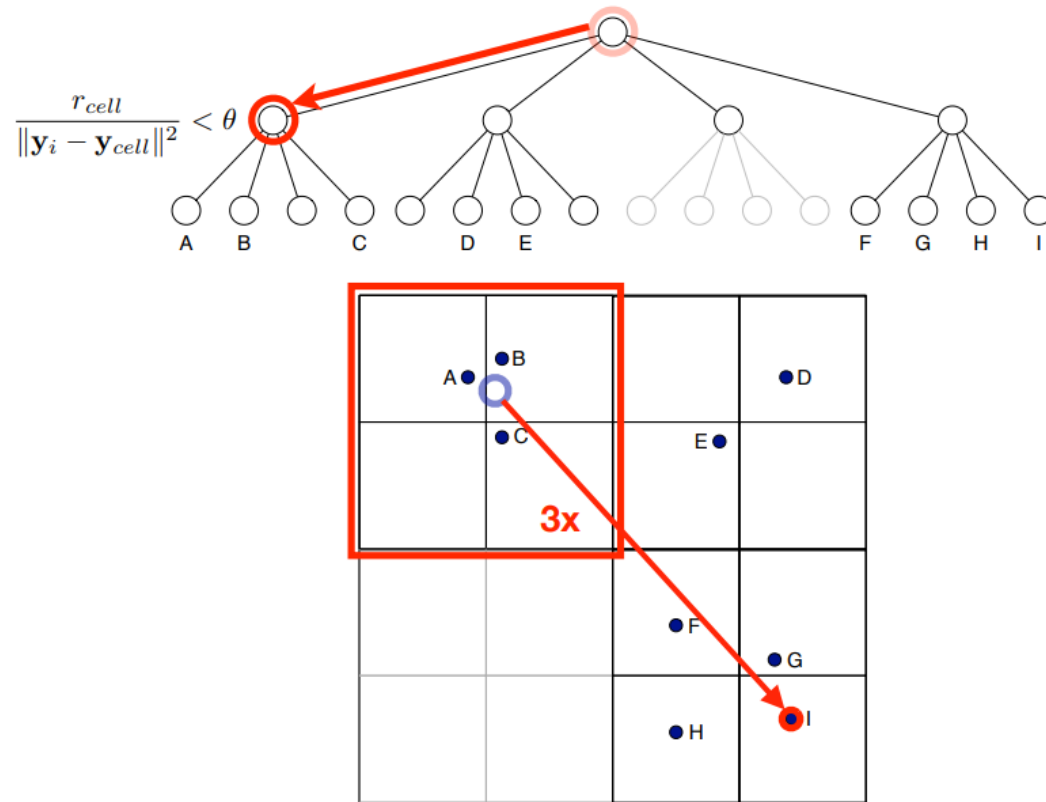
- t-SNE (Maaten & Hinton, 2008) 는 계산 복잡도가 높아서 큰 데이터의 시각화에 사용되지 못했습니다. 이후 개선된 Barnes hut t-SNE (Maaten, 2014)이 제안되었으며, 대부분의 패키지는 이 알고리즘을 쓰고 있습니다.

`sklearn.manifold.TSNE`

```
class sklearn.manifold.TSNE(n_components=2, perplexity=30.0, early_exaggeration=4.0,  
learning_rate=1000.0, n_iter=1000, n_iter_without_progress=30, min_grad_norm=1e-07, metric='euclidean',  
init='random', verbose=0, random_state=None, method='barnes_hut', angle=0.5) ¶ \[source\]
```

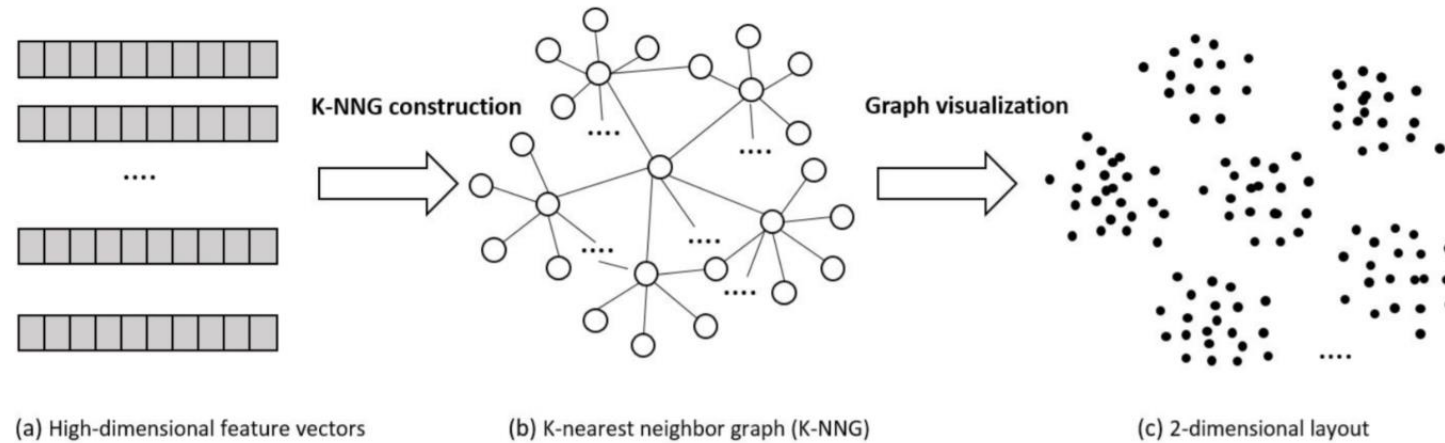
Barnes hut t-SNE

- 비슷한 위치에 존재하는 점들을 한번에 비슷한 방향으로 학습시킴으로써 계산비용을 줄였습니다.



UMAP

- UMAP 도 input data X 를 이용하여 k-nearest neighbor graph, G_x 를 생성한 뒤 이를 임베딩 학습에 이용합니다.



그림은 LargeVis

UMAP

$$v_{j|i} = \exp\left(-\beta_i |r_{ij} - \rho_i|_2^2\right)$$

$$v_{ij} = (v_{j|i} + v_{i|j}) - v_{j|i} \cdot v_{i|j}$$

ρ_i : distance between i and the most closest point

β_i such that $\sum_j v_{j|i} = \log k$, k is the number of neighbors in graph

- ρ_i 최인접이웃과의 거리를 뺀으로써 input space 의 x_i 주변의 거리 분포를 반영합니다.
- G_x 에 포함되지 않으면 $v_{j|i} = 0$ 입니다 (disconnected).

UMAP

$$v_{j|i} = \exp\left(-\beta_i |r_{ij} - \rho_i|_2^2\right)$$

$$v_{ij} = (v_{j|i} + v_{i|j}) - v_{j|i} \cdot v_{i|j}$$

$$w_{ij} = \frac{1}{1 + ad_{ij}^{2b}}$$

$$C_{UMAP} = \sum_{ij} v_{ij} \log\left(\frac{v_{ij}}{w_{ij}}\right) + (1 - v_{ij}) \log\left(\frac{1-v_{ij}}{1-w_{ij}}\right)$$

G_x 에 포함된 점들은 임베딩 공간에서 서로 가깝도록 당겨오고,
 G_x 에 포함되지 않은 점들은 어느 정도 멀어질때까지 밀어냅니다.

UMAP

$$C_{UMAP} = \sum_{ij} v_{ij} \log \left(\frac{v_{ij}}{w_{ij}} \right) + (1 - v_{ij}) \log \left(\frac{1-v_{ij}}{1-w_{ij}} \right)$$

$$\frac{dC_{UMAP}^+}{dy_i} = \frac{-2}{1+d_{ij}^2} v_{ij} (y_i - y_j)$$

$$\frac{dC_{UMAP}^-}{dy_i} = \frac{b}{(0.001 + d_{ij}^2)(1+d_{ij}^2)} (1 - v_{ij})(y_i - y_j)$$

$\frac{dC_{UMAP}^-}{dy_i}$ 에 이용할 y_i, y_j 는 분포를 이용하지 않고 임의로 선택합니다.

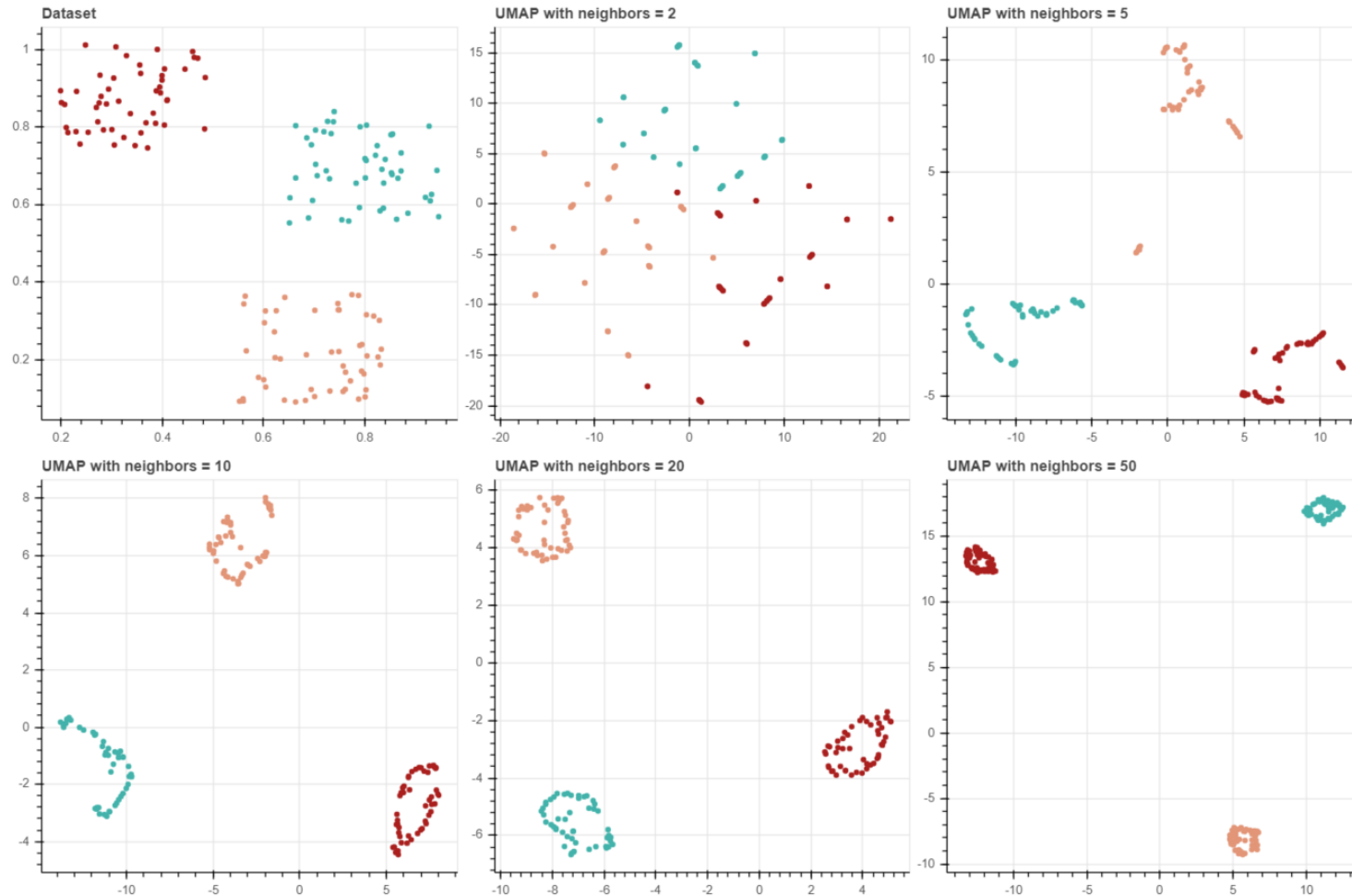
UMAP

- UMAP 은 t-SNE 보다 빠릅니다.

	UMAP	Fit-SNE	t-SNE	LargeVis	Eigenmaps	Isomap
Pen Digits (1797x64)	9s	48s	17s	20s	2s	2s
COIL20 (1440x16384)	12s	75s	22s	82s	47s	58s
COIL100 (7200x49152)	85s	2681s	810s	3197s	3268s	3210s
scRNA (21086x1000)	28s	131s	258s	377s	470s	923s
Shuttle (58000x9)	94s	108s	714s	615s	133s	–
MNIST (70000x784)	87s	292s	1450s	1298s	40709s	–
F-MNIST (70000x784)	65s	278s	934s	1173s	6356s	–
Flow (100000x17)	102s	164s	1135s	1127s	30654s	–
Google News (200000x300)	361s	652s	16906s	5392s	–	–

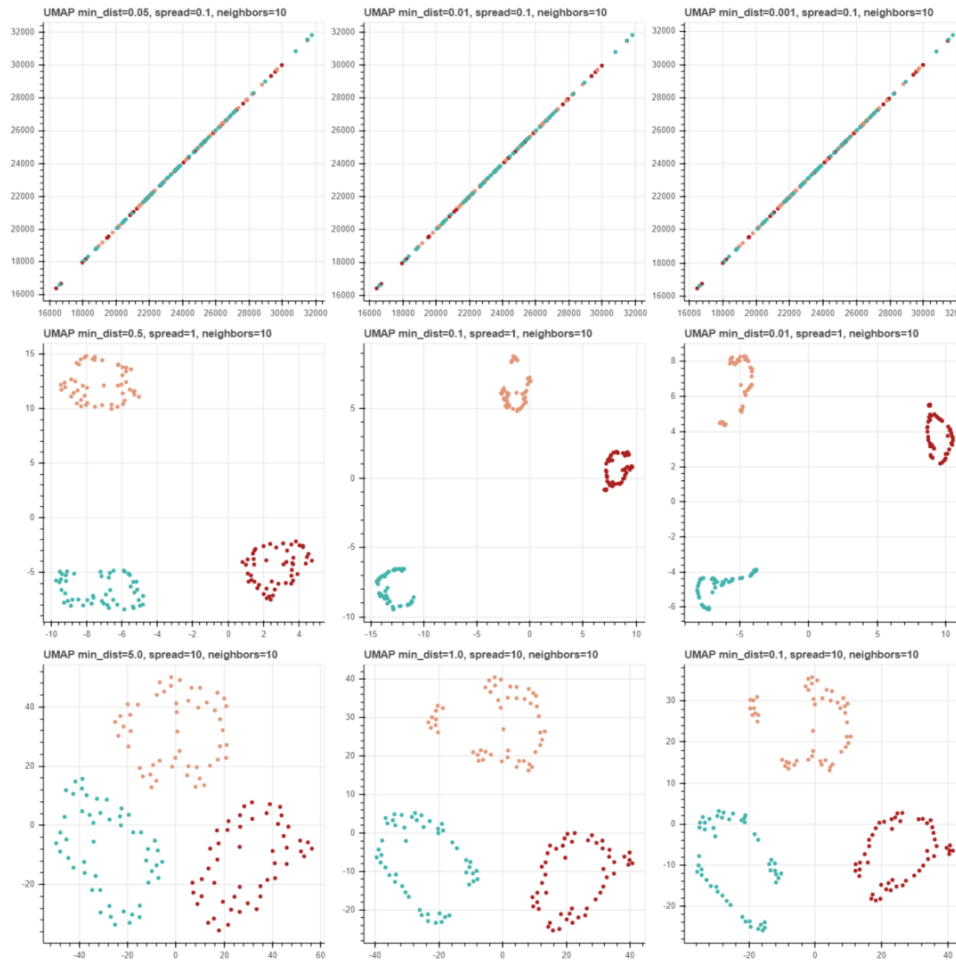
UMAP

- t-SNE 의 perplexity 의 역할을 UMAP 에서 n_neighbors 가 합니다.



UMAP

- `spread`, `min_dist` 에 의하여 경향이 달라질 수 있습니다.

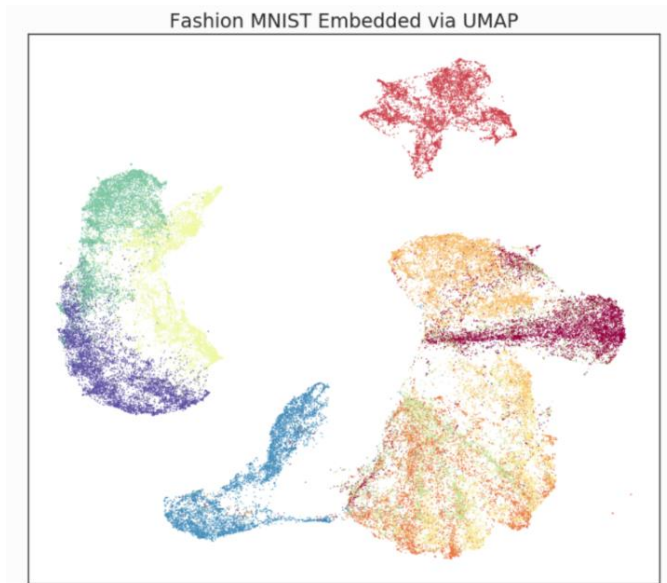


UMAP

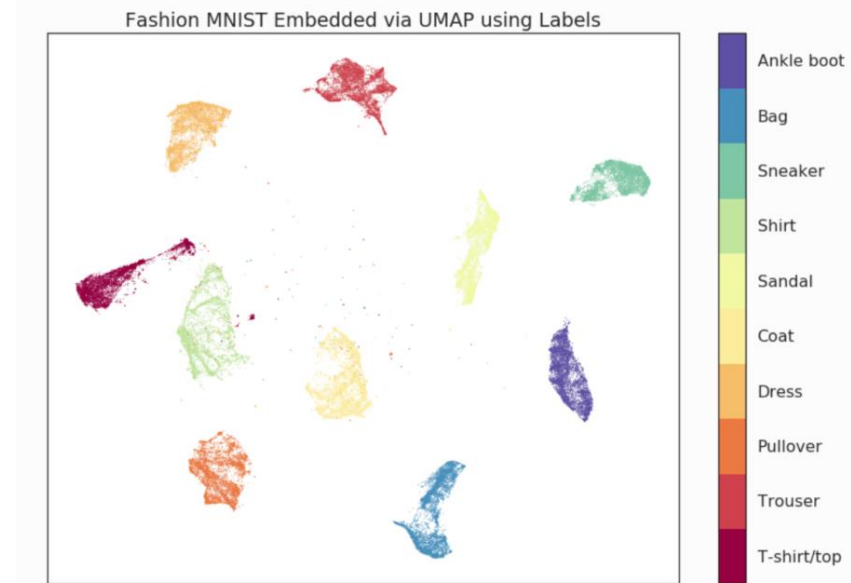
- UMAP 은 label 정보를 이용한 supervised embedding 을 지원합니다.
 - y 를 이용하여 G_y 를 만든 뒤, v_{ij} 를 만들 때 $\alpha G_x + (1 - \alpha)G_y$ 를 이용합니다.
 - y 가 명목변수이면 $\{0, 1\}$, 연속형변수이면 L1, L2 거리를 G_y 의 weight 로 이용합니다.
 - ``target_metric`` 은 'categorical', 'l1', 'l2' 중 선택, ``target_weight`` 는 α 입니다.

UMAP

- 레이블을 이용하면 서로 다른 클래스의 객체들이 구분되어 임베딩 됩니다.



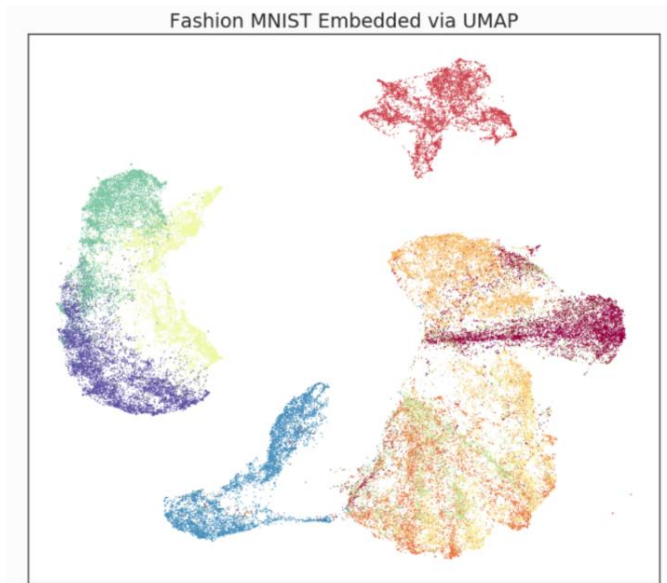
unsupervised embedding



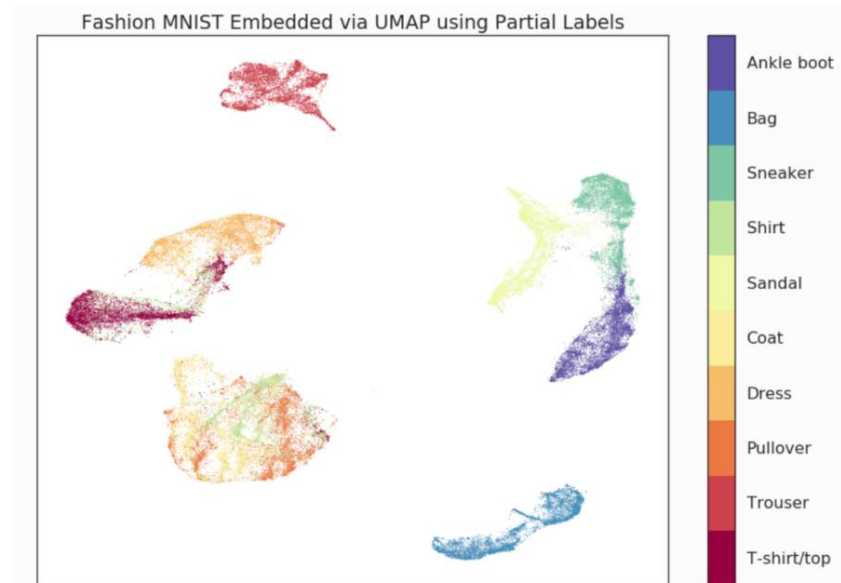
supervised embedding

UMAP

- Partial labeled (semi-supervised) embedding 도 가능합니다.
 - 클래스를 알지 못하는 데이터들은 label 을 -1 로 지정합니다.



unsupervised embedding



semi-supervised embedding

UMAP

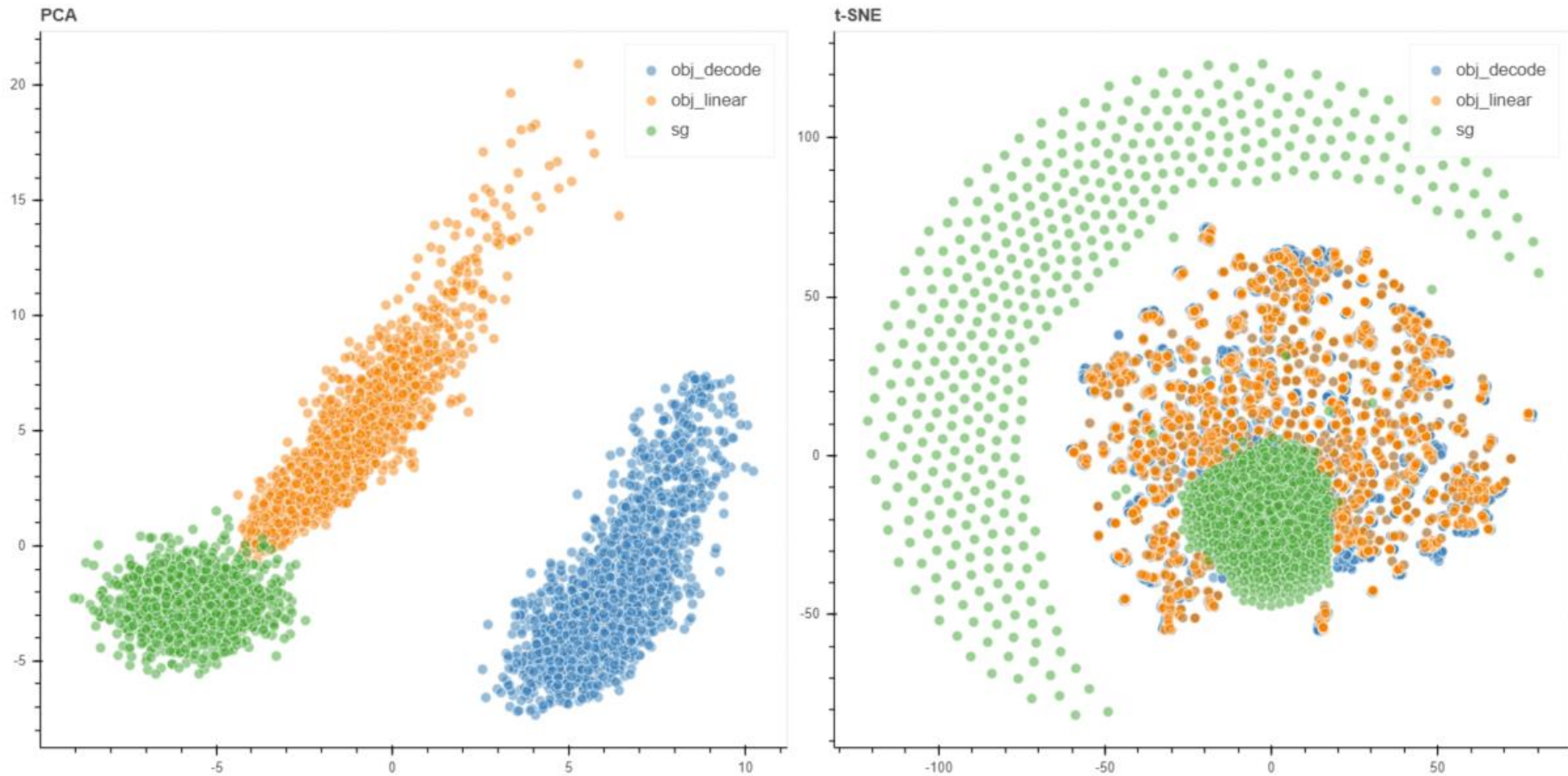
- UMAP 의 학습결과로 embedding mapper 를 얻기 때문에, 새로운 데이터에 대한 임베딩 벡터 추정이 가능합니다.

```
import umap

mapper = umap.UMAP(n_neighbors=10).fit(train_data, train_label)
test_embedding = mapper.transform(test_data)
```

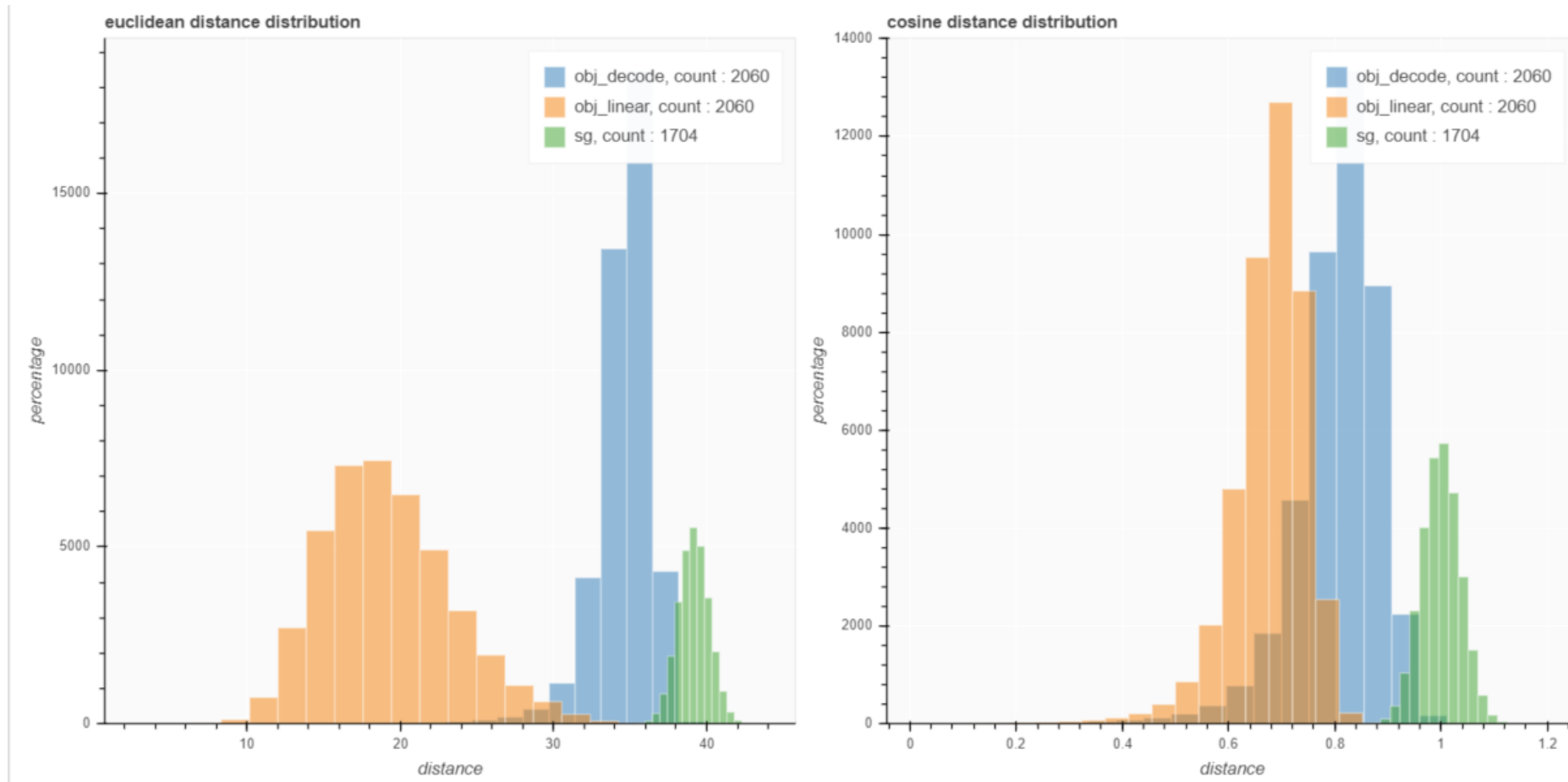

t-SNE vs PCA

- 동일한 데이터 (760 차원) 를 PCA 와 t-SNE 로 임베딩한 결과입니다. 서로 경향이 다릅니다.



t-SNE vs PCA

- 각 클래스별로 pairwise distance (Euclidean, cosine) 를 계산하여 분포를 살펴봅니다.



t-SNE vs PCA

- 녹색 클래스의 데이터가 전체 공간에 골고루 펼쳐져 있기 때문에 Cosine distance 가 1이 넘는 경우들이 많습니다. 하지만 그 개체수는 가장 적습니다.
- PCA 는 특정 방향에 데이터가 몰려있을 경우, 그 방향을 우선적으로 탐색합니다.
- PCA 임베딩 그림만으로는 녹색이 한 곳에 몰려있다고 곡해할 수 있습니다.

