Embedding for Vector Visualization

Hyunjoong Kim

soy.lovit@gmail.com

github.com/lovit

Embedding

- 임베딩은 데이터를 공간 x 에서 새로운 공간 y 로 보내는 것입니다.
  - 원하는 정보를 잘 저장하며 공간을 변환하는 것으로,
  - 어떤 정보를 보존할 것이냐에 따라서 다양한 임베딩 방법이 존재합니다.
- 각 임베딩 방법을 "어떤 정보를 보존"하려 하는지의 관점에서 살펴본다면, 각 알고리즘이 무엇을/어떻게 학습하는지 이해하기 쉽습니다.

Embedding?

- (벡터) 시각화는 고차원으로 표현되는 객체(단어/문서/어떤 것이든)를 2차원의 저차원 백터로 표현하는 것입니다.
  - 시각화를 위한 임베딩을 차원축소 방법으로 부르기도 합니다.
  - 임베딩 방법이기 때문에, 학습 과정에서 원 공간의 특정 정보를 보존합니다.

Multidimensional Scaling (MDS)

- Δ 공간 벡터 간의 거리 $\delta_{i,j}$를 Euclidean distance 으로 보존합니다.
  - 모든 $\delta_{i,j}$ 의 중요도가 동일합니다.
  - 먼 거리의 정보가 더 중요하게 여겨집니다.

minimizes $\sum_{i<j}\left(\left|x_i - x_j\right| - \delta_{i,j}\right)^2$

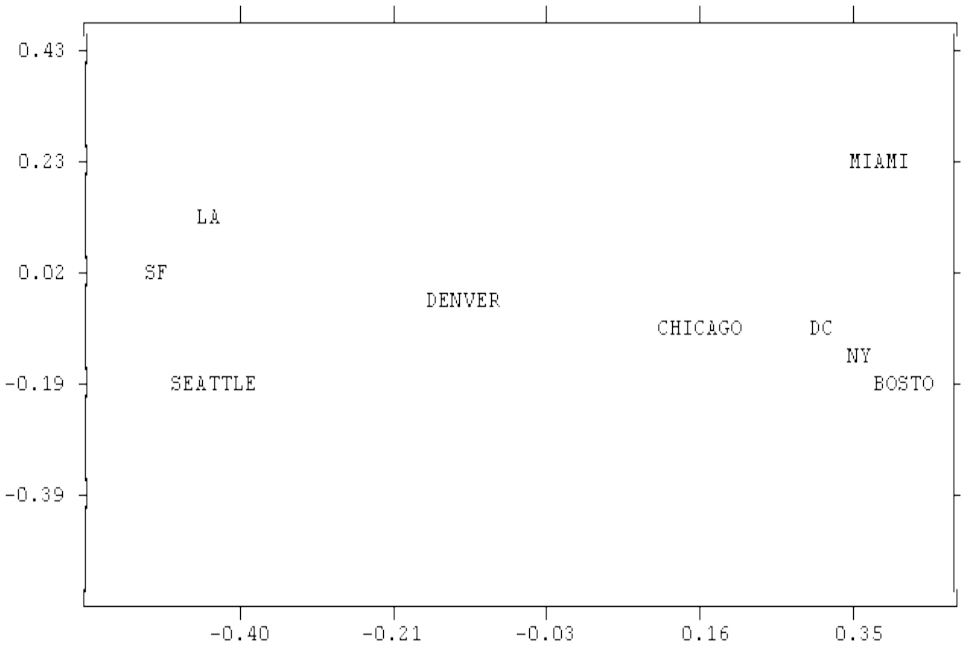
where $\Delta = \begin{pmatrix} \delta_{1,1} & \cdots & \delta_{1,n} \\ \vdots & \ddots & \vdots \\ \delta_{n,1} & \cdots & \delta_{n,n} \end{pmatrix}$

* Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, *29*(1), 1-27.

Multidimensional Scaling (MDS)

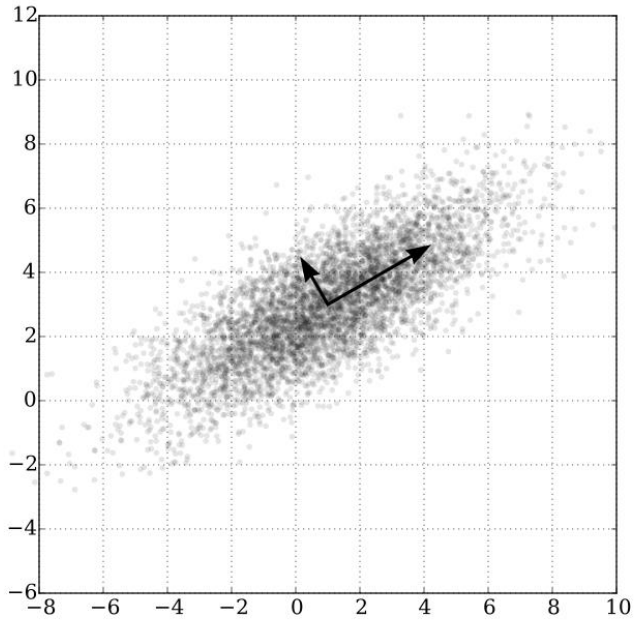
- 도시간 거리를 행렬로 만든 뒤 MDS 를 학습하면 지도가 복원됩니다.

		1	2	3	4	5	6	7	8	9
		BOST	NY	DC	MIAM	CHIC	SEAT	SF	LA	DENV
1	BOSTON	0	206	429	1504	963	2976	3095	2979	1949
2	NY	206	0	233	1308	802	2815	2934	2786	1771
3	DC	429	233	0	1075	671	2684	2799	2631	1616
4	MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
5	CHICAGO	963	802	671	1329	0	2013	2142	2054	996
6	SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
7	SF	3095	2934	2799	3053	2142	808	0	379	1235
8	LA	2979	2786	2631	2687	2054	1131	379	0	1059
9	DENVER	1949	1771	1616	2037	996	1307	1235	1059	0



Principal Component Analysis (PCA)

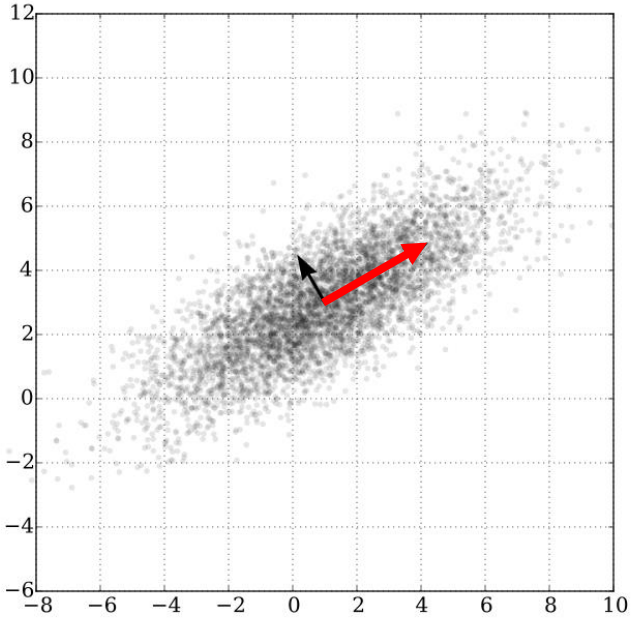
- PCA 는 p 차원의 데이터 X에 대하여, 방향적 분포를 가장 잘 설명하는 q 차원의 새로운 직교 좌표를 찾습니다. ($q \leq p$)
  - X를 q 차원의 공간으로 변환 (Z=XW) 하는 일종의 회전변환 입니다.



https://en.wikipedia.org/wiki/Principal_component_analysis

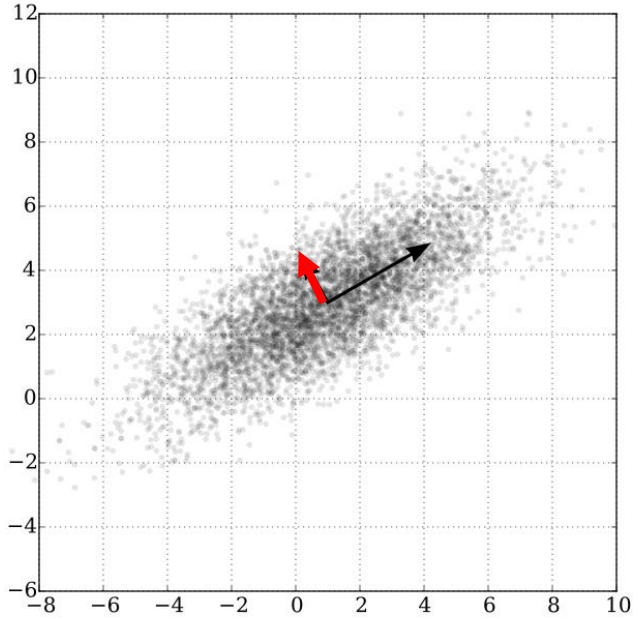
Principal Component Analysis (PCA)

- First component (PC_1)
  - $w_1 = argmax_{||w||=1} \left\{ \frac{w^T X^T X w}{w^T w} \right\}$
  - $X^T X$ (Covariance) 이 가장 큰 방향 벡터 w 를 찾습니다



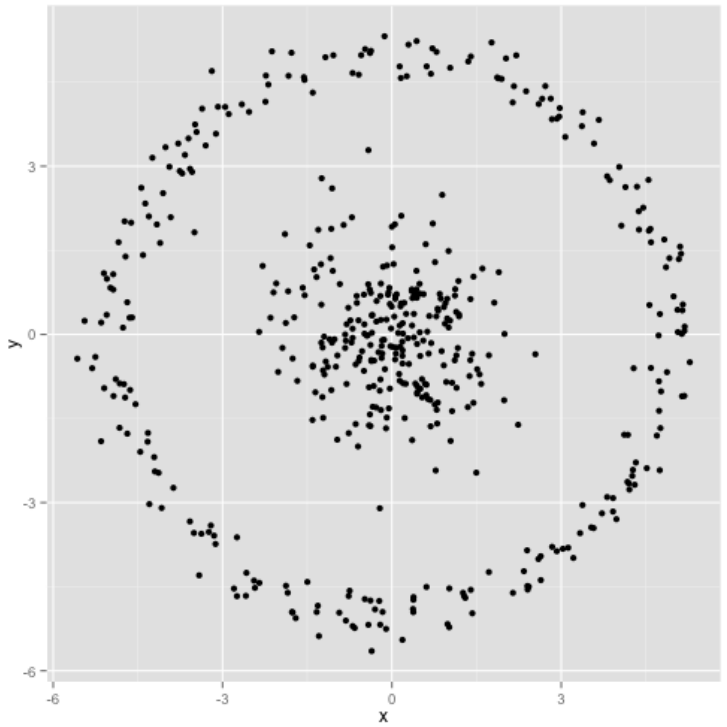
Principal Component Analysis (PCA)

- $\widehat{X_k} = X - \sum_{s=1}^{k-1} X w_s w_s^T$: 이전까지 고려한 데이터의 분포를 뺀 나머지
- $w_k = argmax_{||w||=1} \left\{ \frac{w^T \widehat{X_k}^T \widehat{X_k} w}{w^T w} \right\}$
- $X^T X$ (Covariance)이 가장 큰 방향 벡터 w



Principal Component Analysis (PCA)

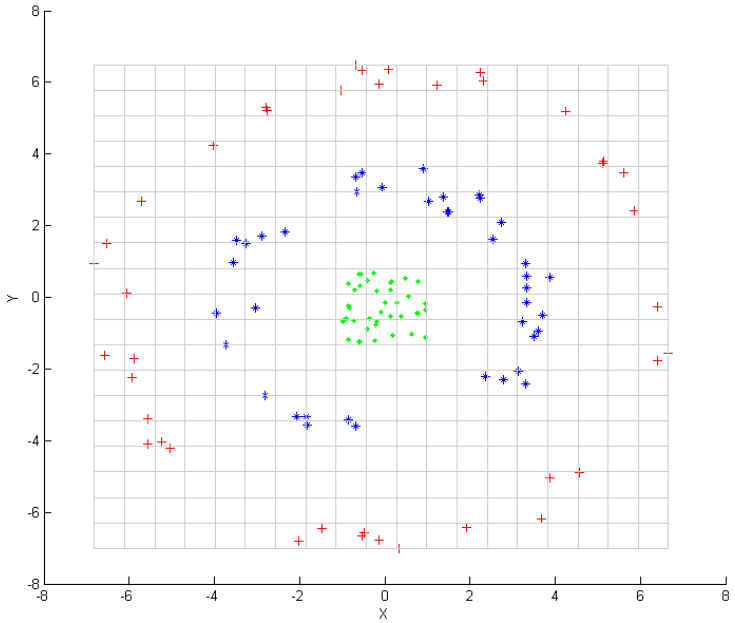
- PCA 는 데이터의 방향적인 경향이 있을 때 잘 작동합니다.
- 데이터의 경향이 방향적이지 않는 경우는 주요 축을 찾을 수 없습니다.



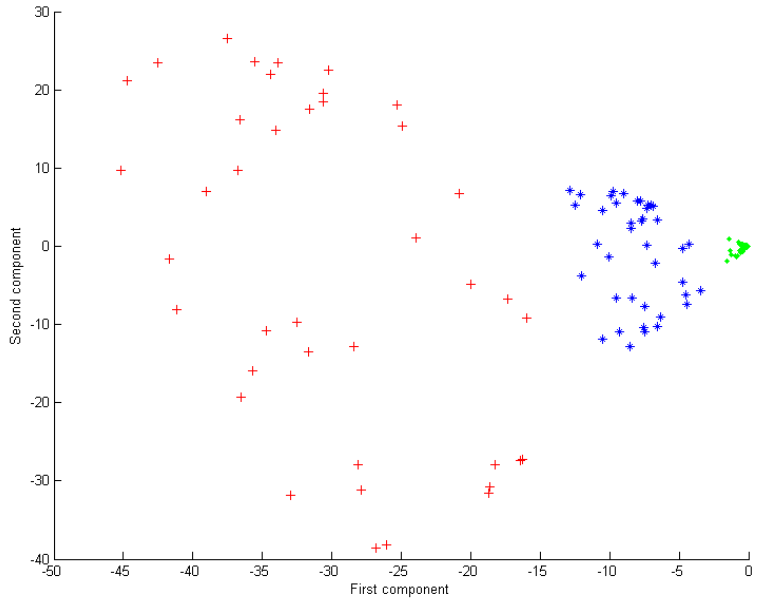
http://varianceexplained.org/

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.
  - $w_1 = argmax_{||w||=1} \left\{ \frac{w^T \boldsymbol{K(X,X)} w}{w^T w} \right\}$



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

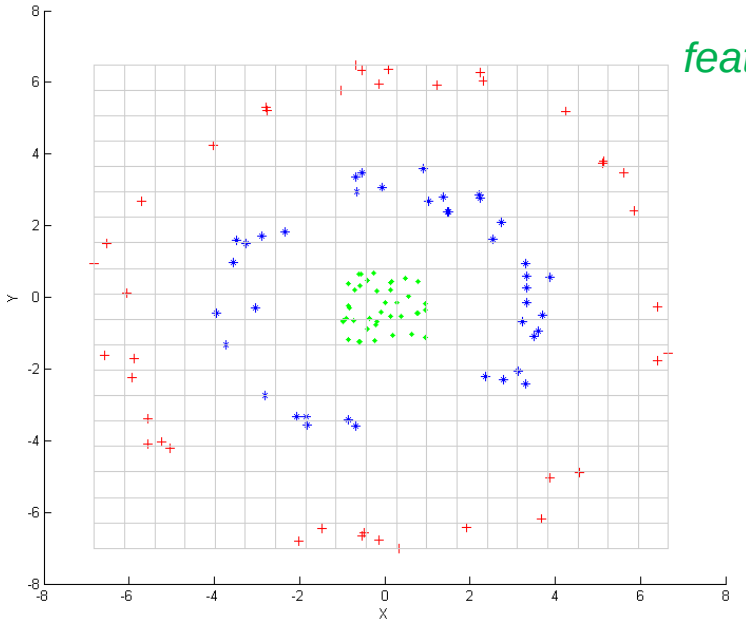


https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

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:

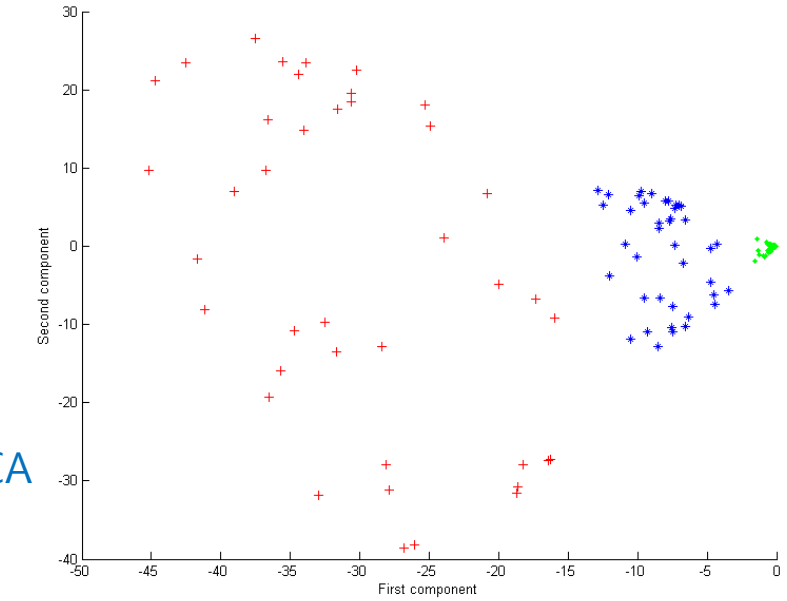


user-specified feature map:

feature vector

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

$X^T X$ 대신 $k(x, y)$ 를 PCA



https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

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.

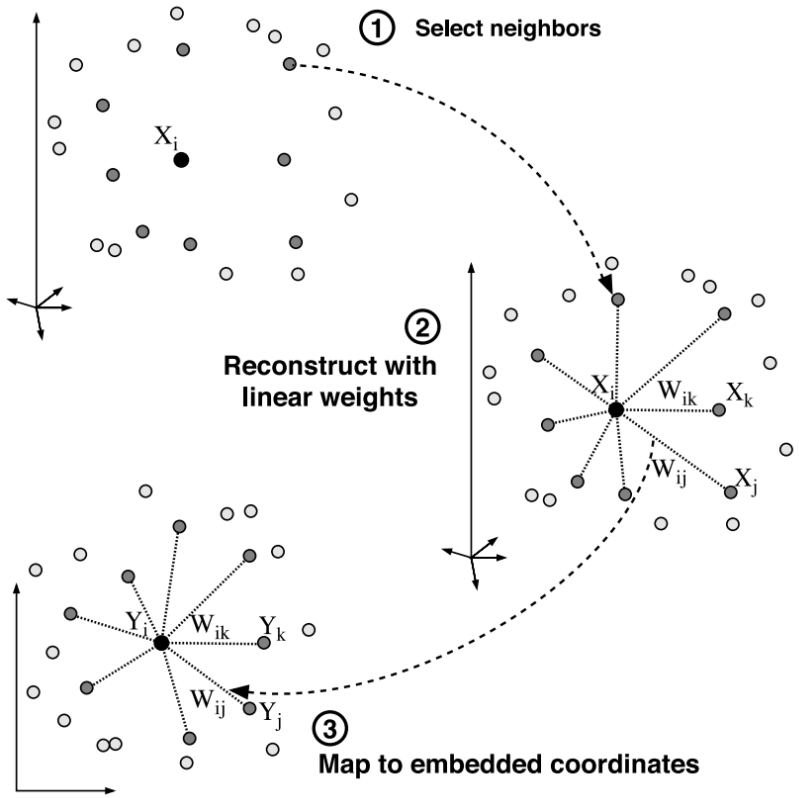
Locally Linear Embedding (LLE)

- LLE 는 x_i 의 주위 k개의 이웃 구조를 보존한 저차원 공간을 학습합니다.

  - 1단계: x_i와 가까운 k개의 이웃을 선택
  - 2단계: 본래 공간에서의 이웃간의 구조학습

    minimizes $\varepsilon(W) = \sum_i \left| x_i - \sum_j w_{ij} x_j \right|^2$
  - 3단계: W를 보존하는 y_i학습

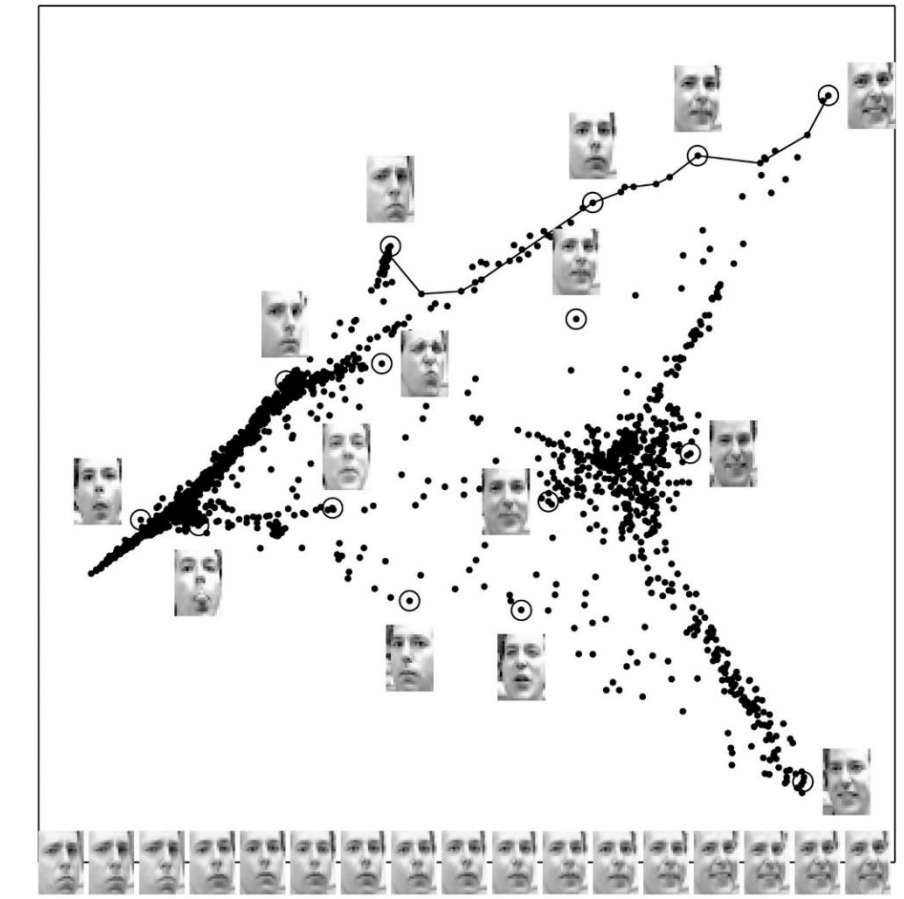
    $\varphi(Y) = \sum_i \left| y_i - \sum_j w_{ij} y_j \right|^2$



* Roweis, S. T., & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*(5500)

Locally Linear Embedding (LLE)

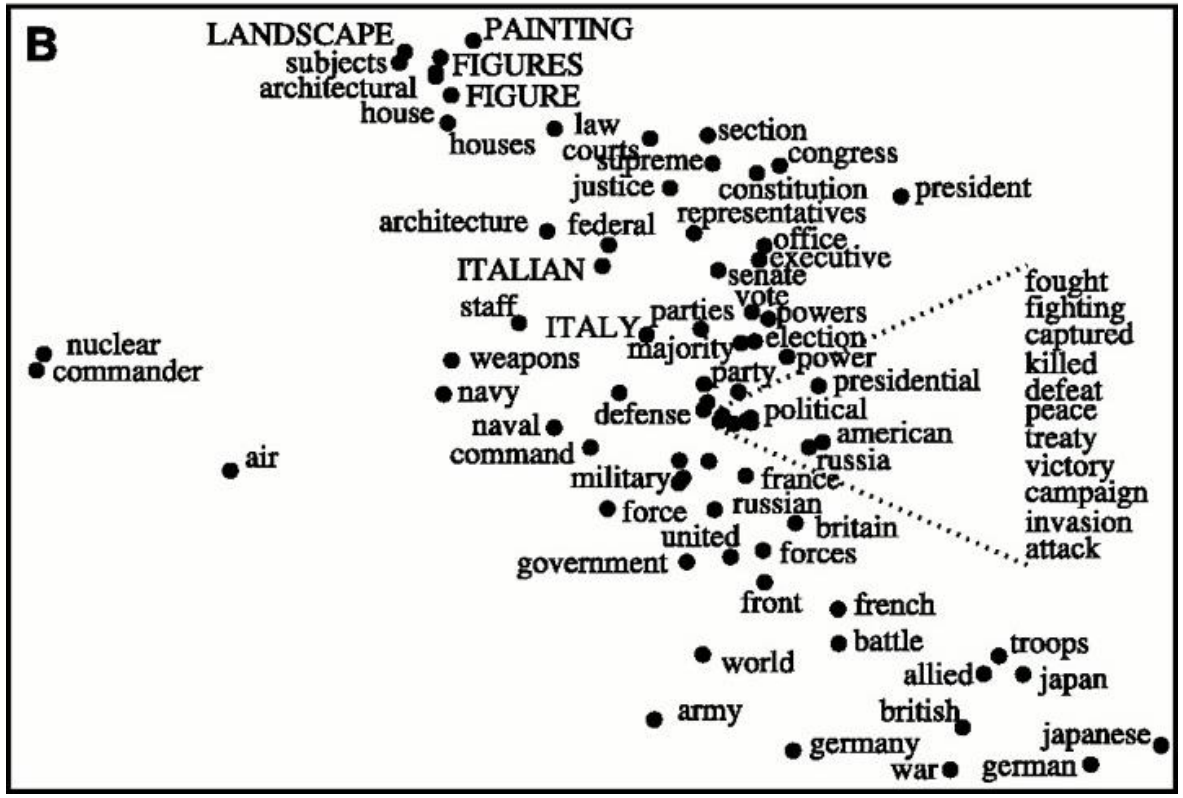
- 비슷한 점들간의 지역적 구조만을 보존해도 "어떤 흐름"이 학습됩니다.
  - 얼굴 이미지 데이터를 LLE로 시각화한 예시



웃는 얼굴 방향

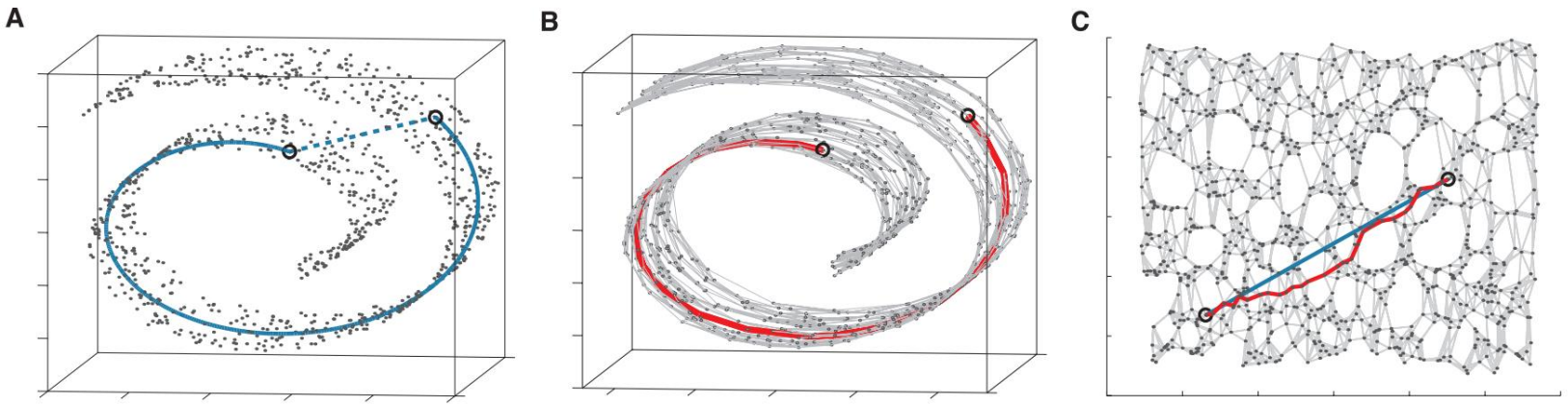
Locally Linear Embedding (LLE)

- Term-document matrix 를 단어 기준으로 임베딩하여 topic modeling 도 하였습니다. 등장한 문서들이 비슷한 단어는 서로 최인접이웃입니다.



ISOMAP

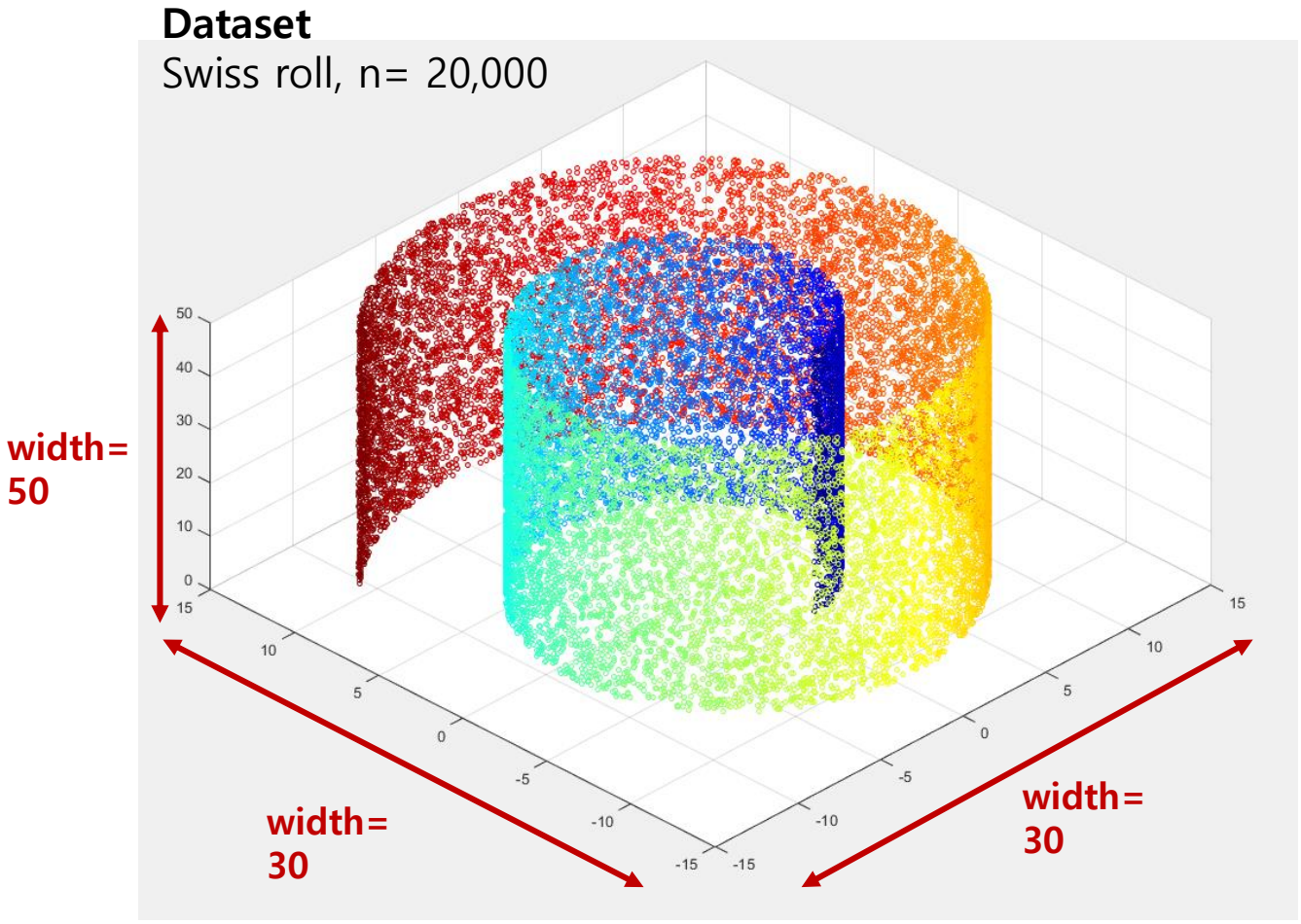
- *k* nearest neighbor graph 에서의 두 점간의 shortest path distance 를 보존합니다.
  - 복잡한 모양의 데이터를 단순한 평면으로 표현할 수 있습니다.



* Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*,

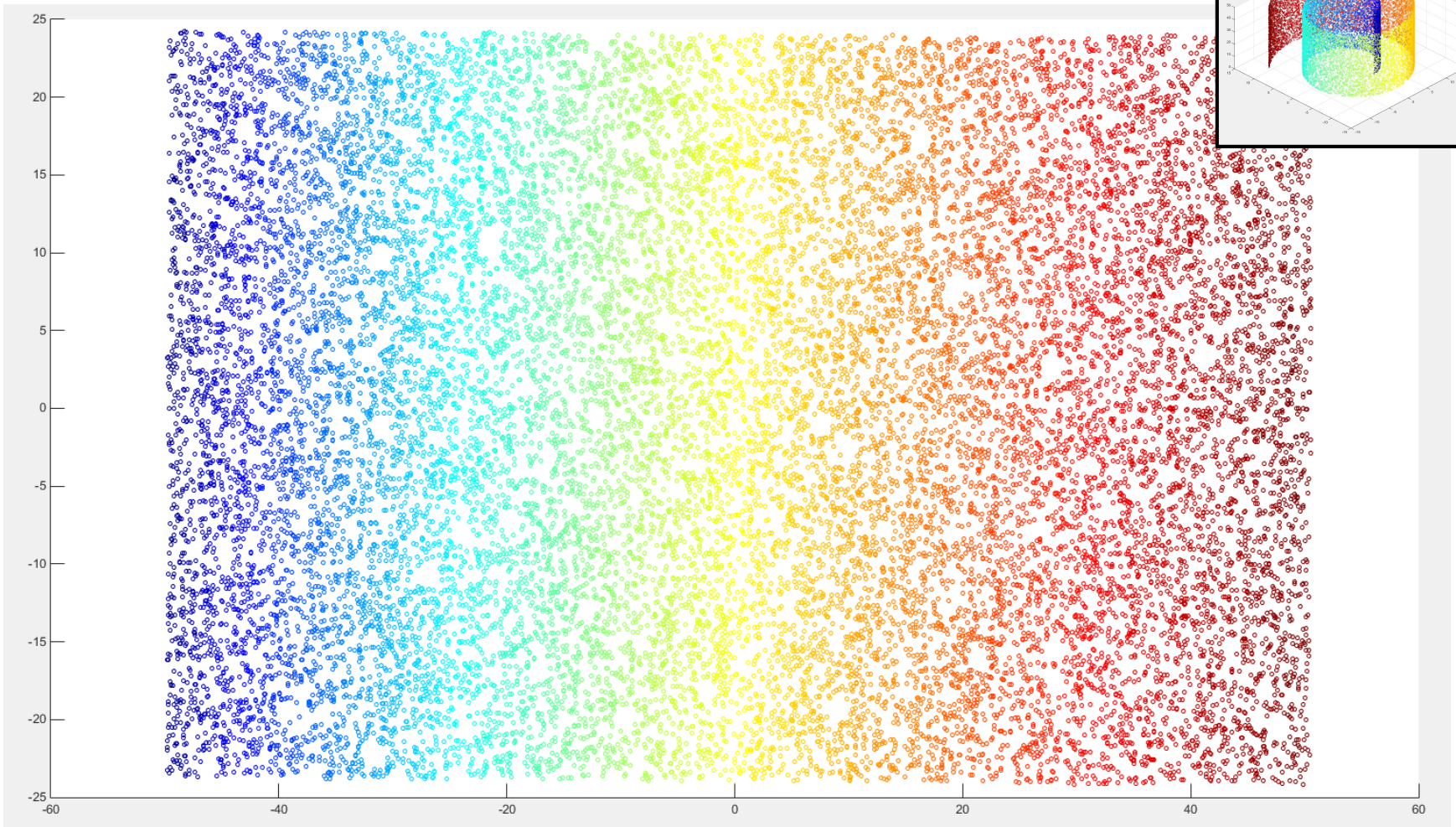
ISOMAP

- ISOMAP은 Swiss roll 과 같은 구조를 잘 표현합니다.



ISOMAP

- ISOMAP은 Swiss roll 과 같은 구조를 잘 표현합니다.



t-Stochastic Neighbor Embedding (t-SNE)

- t-SNE는 최근 시각화 방법으로 가장 널리 쓰입니다. x_i 의 이웃간의 거리를 확률로 표현한 뒤 이를 보존하는 공간을 학습합니다.

$$\text{Find } y_i \text{ that minimizes } \sum p_{ij} * \log \frac{p_{ij}}{q_{ij}}$$

$$p_{j|i} = \frac{\exp\left(-\left|x_i - x_j\right|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-|x_i - x_k|^2 / 2\sigma_i^2\right)}, p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

$$q_{ij} = \frac{\left(1 + \left|y_i - y_j\right|^2\right)^{-1}}{\sum_{k \neq l} (1 + |y_k - y_l|^2)^{-1}}$$

* Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(2579-2605), 85.

t-Stochastic Neighbor Embedding (t-SNE)

- t-SNE 는 X 에서의 $p_{j|i}$ 가 큰 x_i, x_j 가 q_{ij} 도 크도록 y_i 를 학습합니다.
  - X에서 NNG 를 만든 뒤 이를 Y 로 옮기는 의미입니다.

$$p_{j|i} = \frac{\exp\left(-|x_i - x_j|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-|x_i - x_k|^2 / 2\sigma_i^2\right)}$$

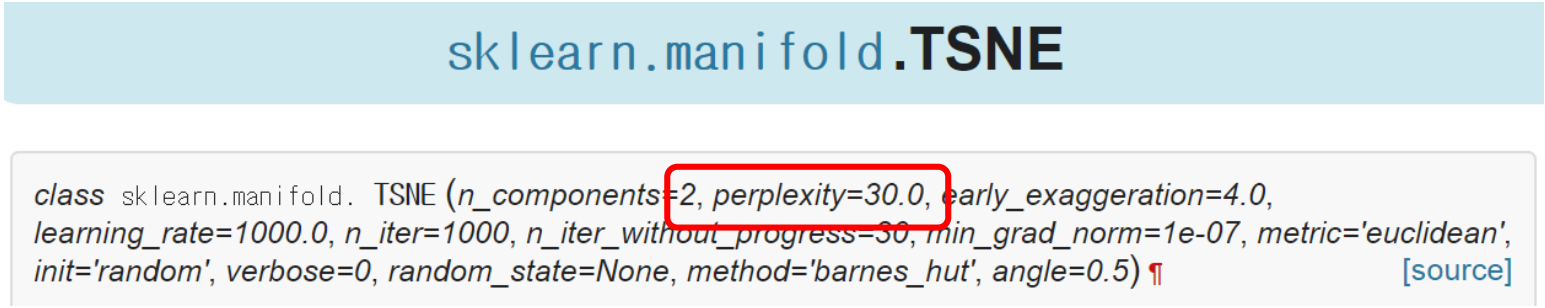
  - LLE와 비슷하지만, 학습방법이 gradient descent (NN 학습방법)을 이용하고, k-NN을 찾는 것이 아니라는 점이 다름
    - nearest neighbor를 표현하는 방법이 k-NNG가 아니라 $p_{j|i}$

t-Stochastic Neighbor Embedding (t-SNE)

- LLE 도 nearest neighbor graph 를 이용합니다.
  - 하지만 t-SNE 는 k 개의 이웃을 이용하지 않고, perplexity 에 의하여 정의되는 특정 영역의 모든 이웃을 이용합니다.
  - nearest neighbor 를 표현하는 방법이 k-NNG 가 아니라 $p_{j|i}$ 로 표현합니다.

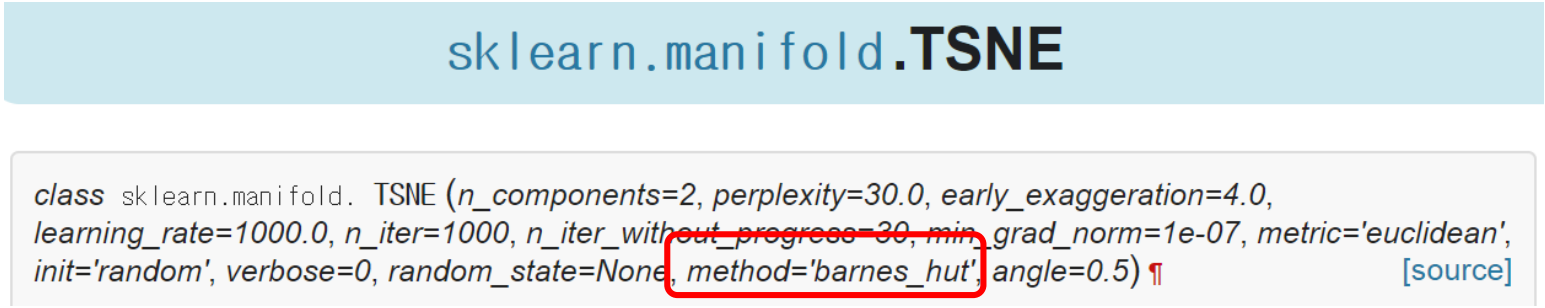
t-Stochastic Neighbor Embedding (t-SNE)

- X 에서 고려하는 최인접이웃의 개수는 perplexity 에 의하여 조절됩니다.
  - Perplexity 가 클수록 더 많은 점을 고려합니다.
  - Perplexity 가 지나치게 크면 모든 점들 간의 거리가 균일하게 나타납니다.
  - 적은 수의 데이터를 학습할 때 결과가 좋지 않으면 perplexity 부터 조절하면 경향이 바뀔 수 있습니다.



t-Stochastic Neighbor Embedding (t-SNE)

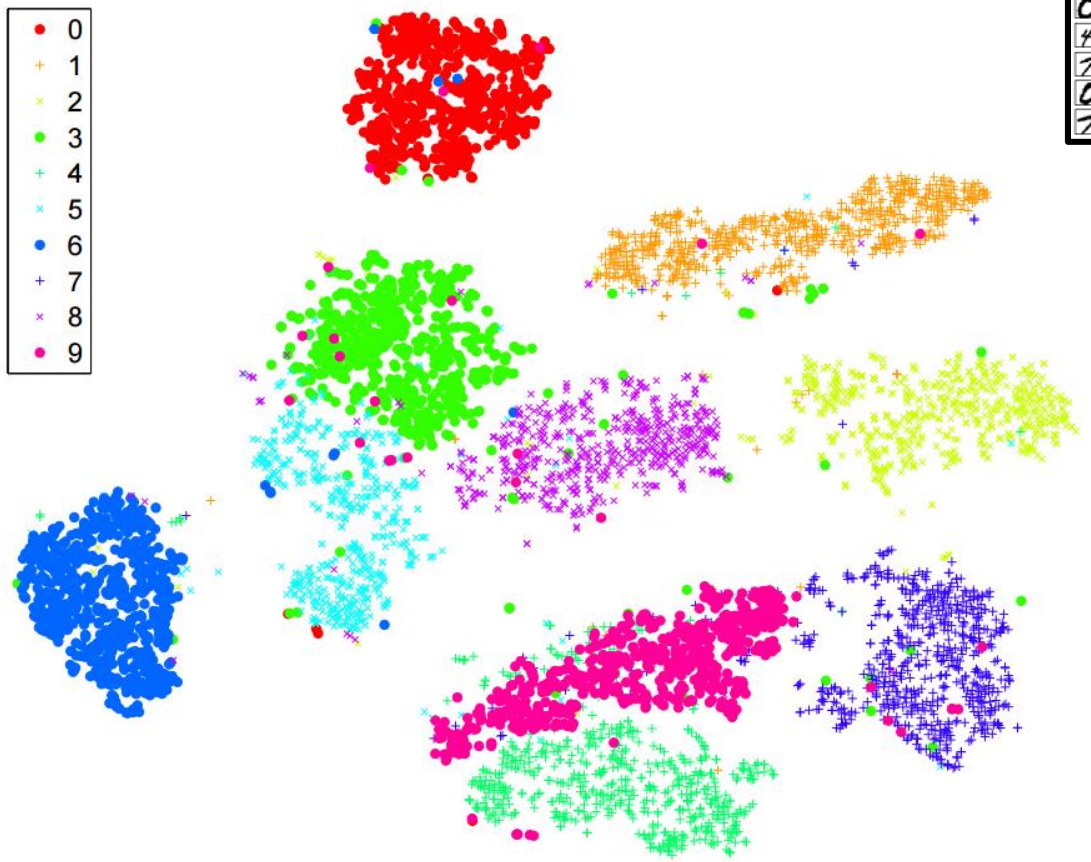
- 처음 제안된 t-SNE (Maaten & Hinton, 2008) 는 계산 복잡도가 높아서 큰 데이터의 시각화에 사용되지 못했습니다.
- 이후 개선된 Barnes hut t-SNE (Maaten, 2014)이 제안되었으며, 대부분의 패키지는 이 알고리즘을 쓰고 있습니다.

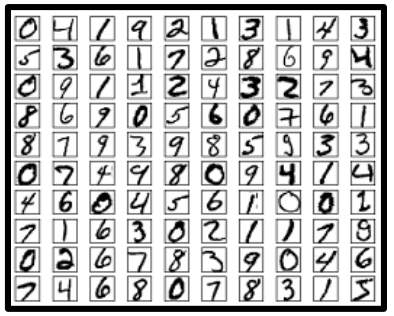


* L.J.P. van der Maaten and G.E. Hinton. V**isualizing High-Dimensional Data Using t-SNE**. *Journal of Machine Learning Research* 9(Nov), 2008
** L.J.P. van der Maaten. **Accelerating t-SNE using Tree-Based Algorithms**. *Journal of Machine Learning Research* 15(Oct):3221-3245, 2014

t-Stochastic Neighbor Embedding (t-SNE)

- 손글씨 숫자 데이터 (MNIST)의 시각화 예시

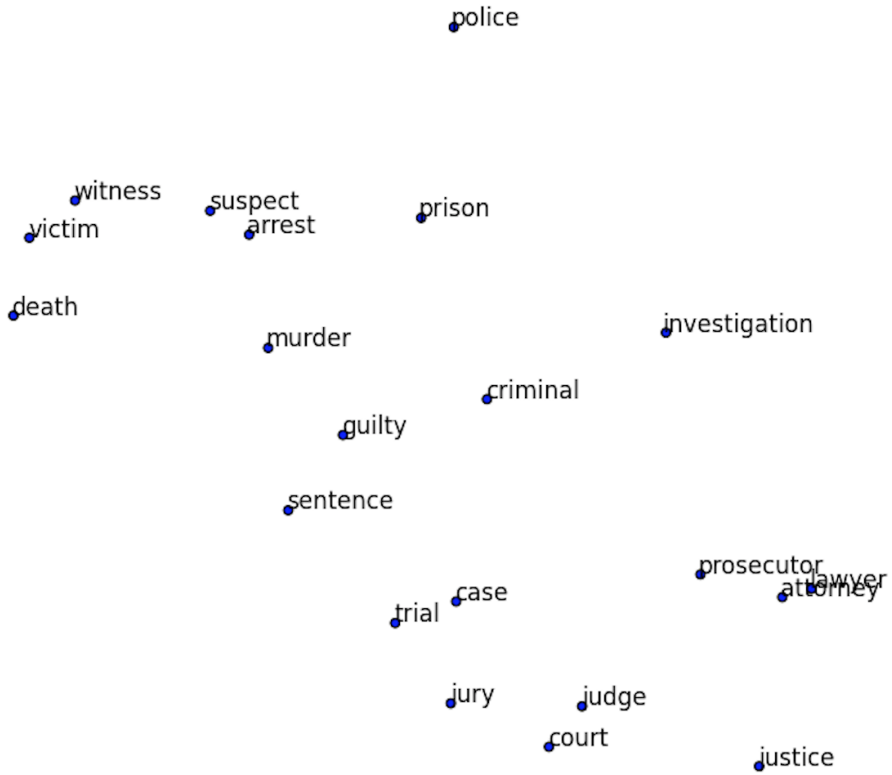




* Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(2579-2605), 85.

t-Stochastic Neighbor Embedding (t-SNE)

- 최근 Word2Vec 과 같은 word embedding (고차원 벡터) 학습 결과 시각화를 위해서 자주 이용됩니다.



* http://nlp.yvespeirsman.be/blog/visualizing-word-embeddings-with-tsne/

어떤 임베딩 방법을 써야 할까?

- MDS는 모든 점들간의 거리 정보를 보존하려 하지만, 시각화 에서는 비슷한 대상들이 비슷한 2차원 벡터를 지니는 것이 중요합니다.
- 비슷한 문서/단어들 (k neighbors) 간의 관계를 잘 보존하는 LLE 나 t-SNE 가 시각화 용도로 가장 적합합니다.
- 더하여 t-SNE 가 안정적인 (경험적) 경향이 있습니다.

어떤 임베딩 방법을 써야 할까?

- Term frequency vector 로 표현된 문서나 word embedding 으로 학습된 단어 벡터 공간은 Swiss roll 처럼 복잡한 공간이 아닙니다.
- 복잡한 공간을 잘 표현하기 위한 방법들을 이용할 필요는 없습니다.

Python

- scikit-learn 에서 다양한 시각화용 임베딩 알고리즘을 사용할 수 있습니다.
  - Slide 작성 당시 버전은 0.19.1 입니다.
  - Parameters 이름이 통일되어 있어서 이용이 간편합니다.
    - n_components : X 의 새로운 공간 Y 에 대한 차원의 개수입니다.
    - fit(X) : X 에서 Y 로 차원을 바꾸는 규칙을 학습합니다.
    - transform(X) : X 를 Y 의 공간으로 변환합니다.
    - fit_transform(X) : fit, transform 두 함수가 함께 이뤄집니다.
  - 그 외의 각 알고리즘 별 parameters 도 있습니다.

Data

- 2016-10-24 뉴스 26,368 건의 단어 691 개에 대한 단어 – 문서 행렬
  - (691, 26368)
- 691 개의 단어들의 등장한 문서가 비슷하면 비슷한 2 차원의 좌표를 얻도록 함으로써, 각 단어의 topical similarity 를 시각화 할 수 있습니다.

t-SNE

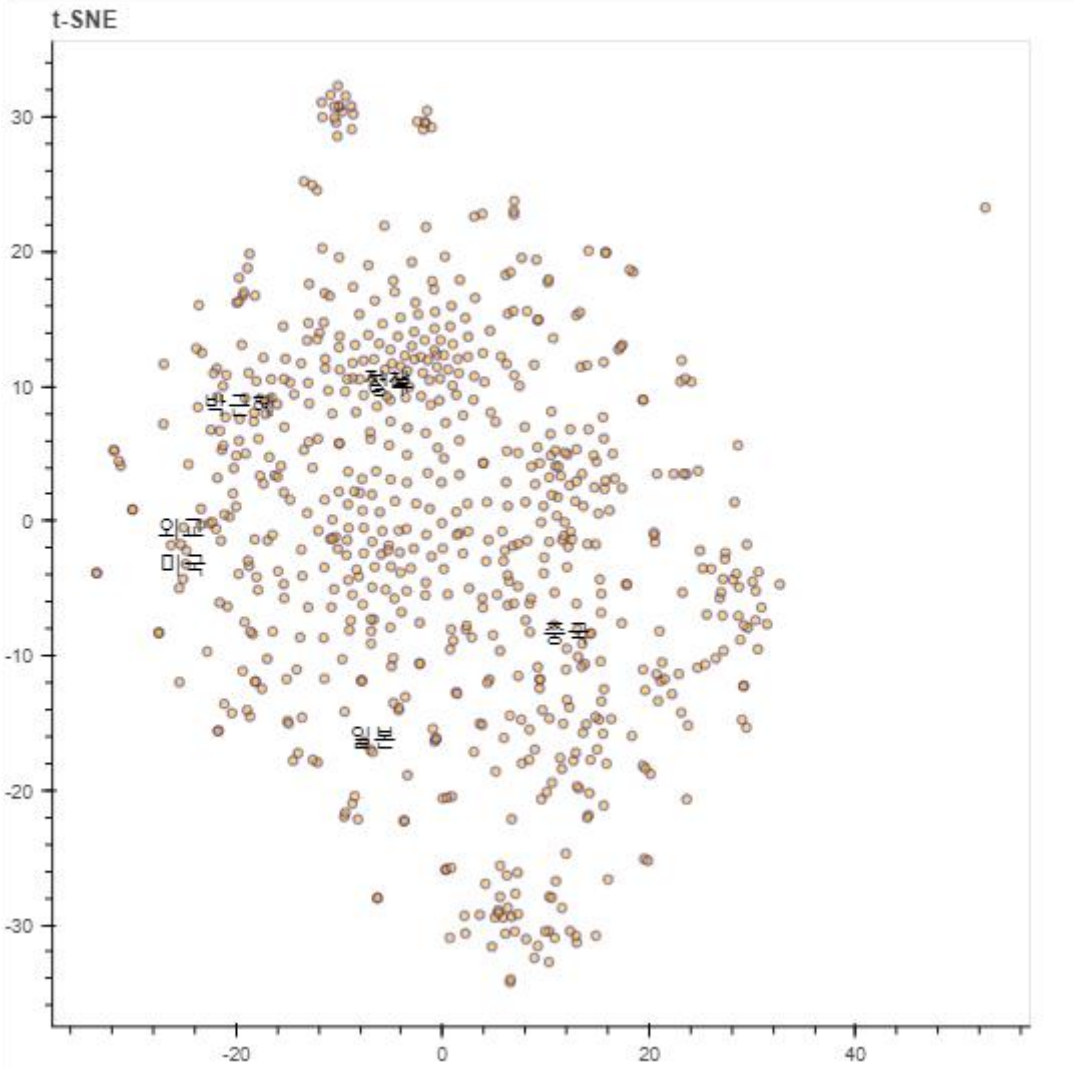
- t-SNE 에서 잘 설정해야 하는 parameter 는 perplexity 입니다.
  - Perplexity 가 크면 아주 가까운 점과 좀 더 떨어진 점과의 거리 차이가 없게 됩니다.
  - 데이터의 개수가 작을 때 perplexity 가 크면 균일한 간격의 grid 모양의 embedding 이 학습될 수 있습니다.

t-SNE

- t-SNE 는 (특히 Barnes-hut 을 이용하는) 원형 모양의 임베딩 공간을 학습하는 경향이 있습니다.
- t-SNE 의 학습 결과는 가까운 점들이 가깝다는 것에만 의미가 있습니다.
  - 학습된 점이 매우 비슷하면 두 점은 원공간 X 에서도 가까운 점일 가능성이 높습니다.
  - 하지만 조금 떨어져있다고하여 원공간에서 떨어진 점은 아닙니다.

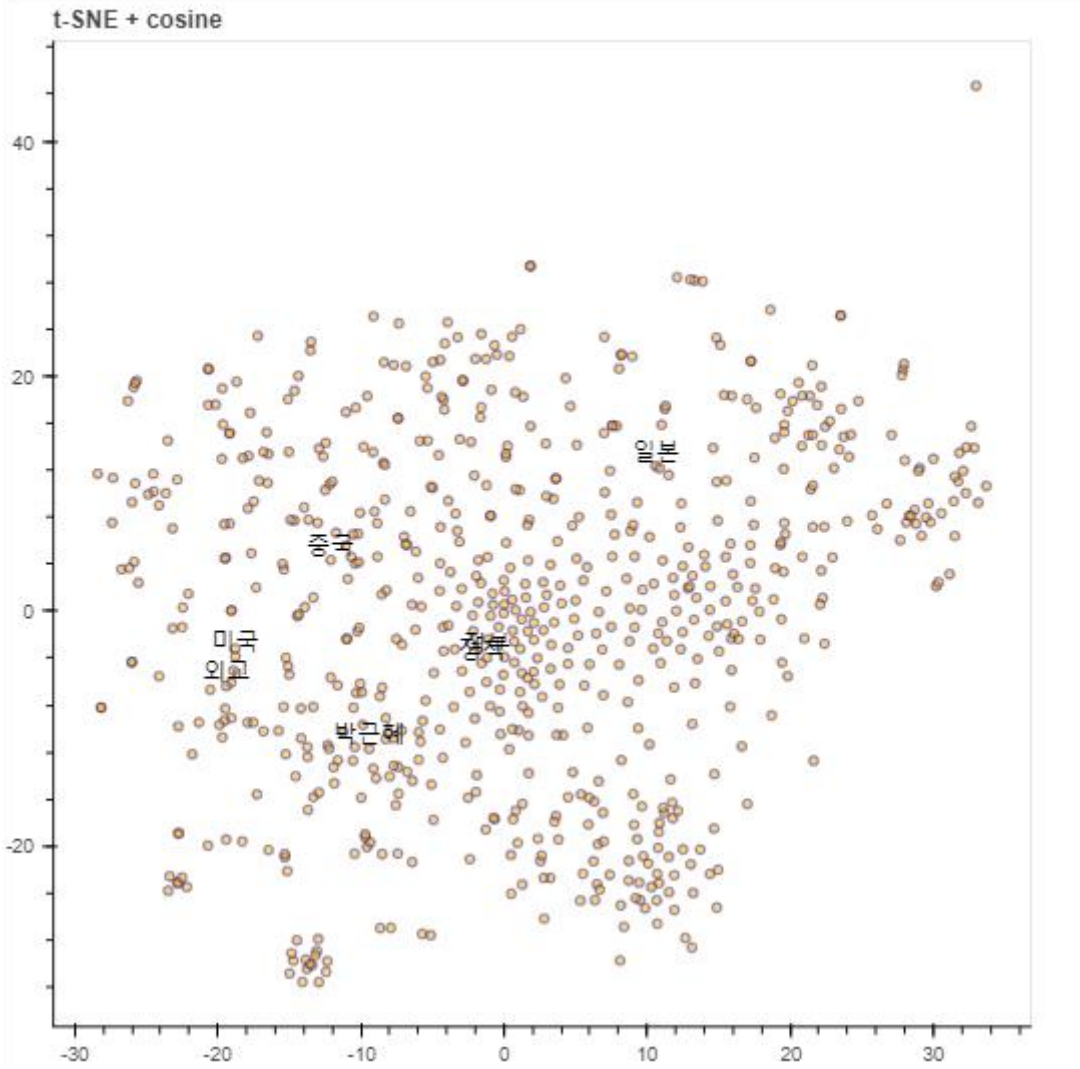
t-SNE

```
from sklearn.manifold import TSNE


tsne = TSNE(
    n_components=2,
    perplexity=30
)
y_tsne = tsne.fit_transform(x)
```



t-SNE + cosine

```
from sklearn.manifold import TSNE


tsne = TSNE(
    n_components=2,
    perplexity=30,
    metric='cosine'
)
y_tsne = tsne.fit_transform(x)
```

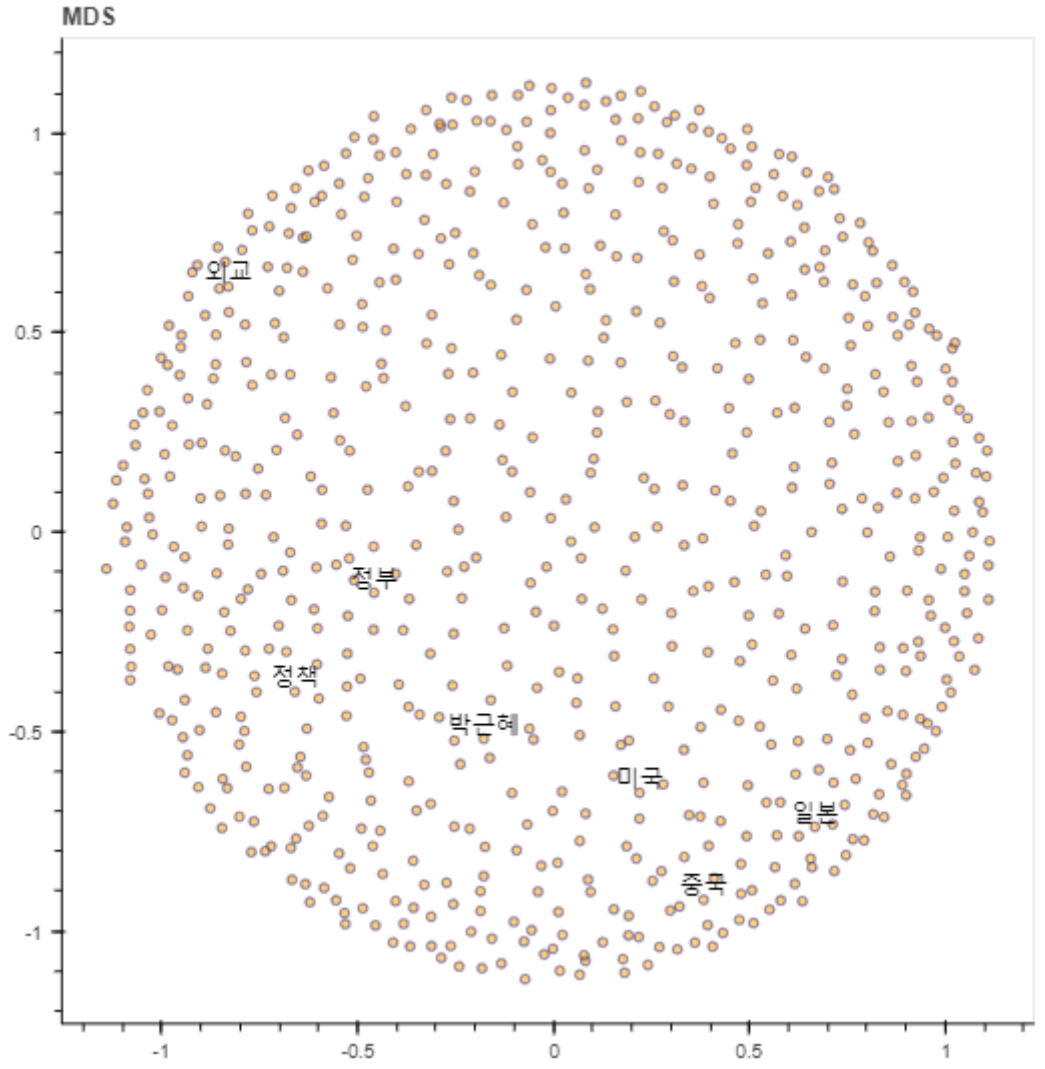


MDS

- 먼 점 간의 거리를 우선적으로 보존하다보니 동그란 모양의 구형이 만들어지는 경우가 많습니다.

MDS

```
from sklearn.manifold import MDS

mds = MDS(n_components=2)
y_mds = mds.fit_transform(x)
```

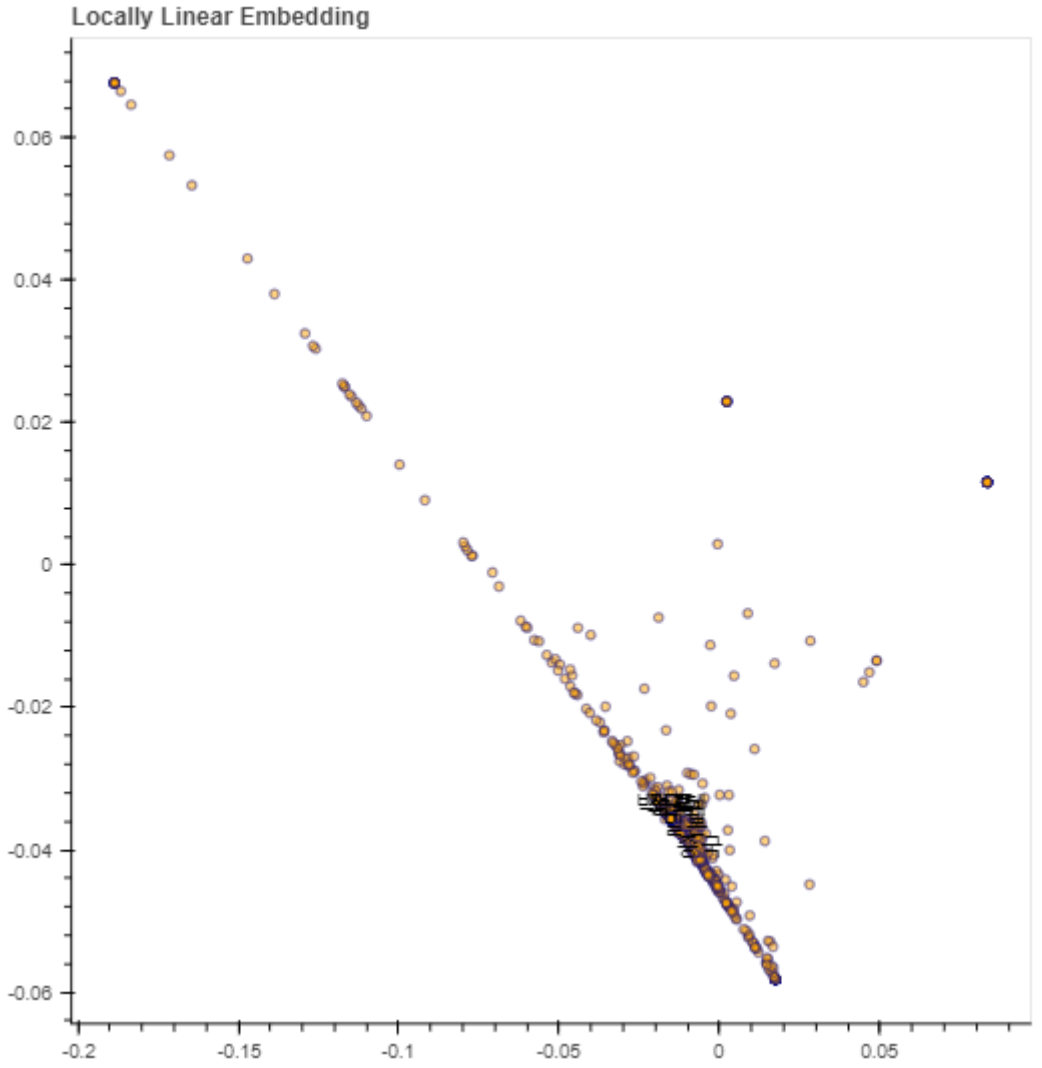


Locally Linear Embedding

- LLE 는 X 의 모든 점에 대하여 nearest neighbors 를 찾은 뒤, 이들과의 관계를 보존하는 새로운 공간 Y 를 학습합니다.
  - Nearest neighbors 의 숫자를 정의할 수 있습니다.
  - 지나치게 작은 숫자만 아니면 경향은 비슷합니다.
- 일직선이나 몇 개의 선들이 그어진 형태로 임베딩이 학습되는 경우가 많습니다.

Locally Linear Embedding

```
from sklearn.manifold import LocallyLinearEmbedding


lle = LocallyLinearEmbedding(
    n_components=2,
    n_neighbors=5
)
y_lle = lle.fit_transform(x)
```

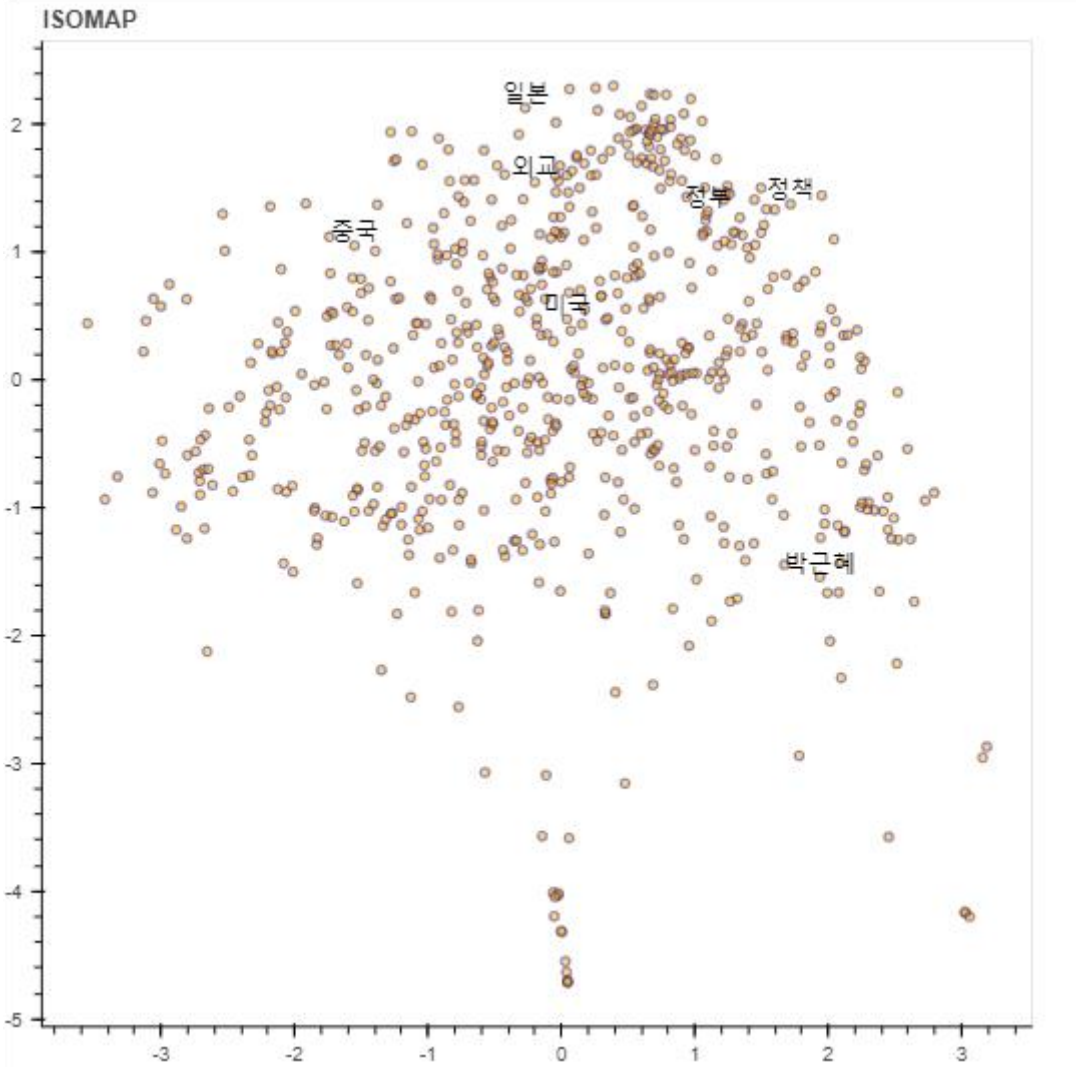


ISOMAP

- ISOMAP 도 nearest neighbors graph 를 만든 뒤, graph 위에서의 shortest path distance 로 두 점 간의 거리를 정의합니다.
  - n_neighbors 로 nearest neighbors graph 의 최인접 이웃의 개수를 조절할 수 있습니다.
- ISOMAP 은 가오리 형태의 모양으로 임베딩이 학습되는 경우가 많습니다.

ISOMAP

```
from sklearn.manifold import Isomap


isomap = Isomap(
    n_components=2,
    n_neighbors=5
)
y_isomap = isomap.fit_transform(x)
```

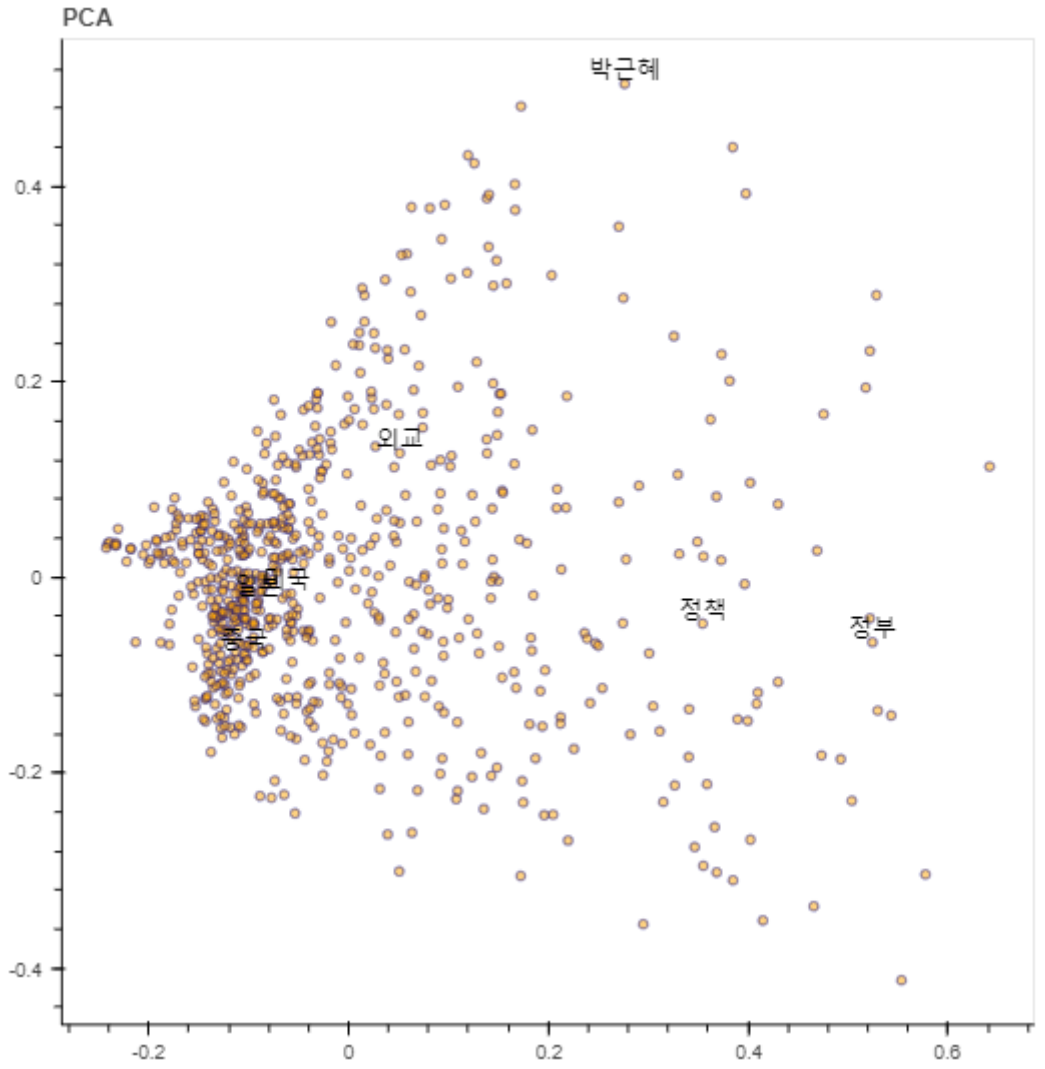


(Kernel) Principal Components Analysis

- PCA 는 sklearn.manifold 가 아닌 sklearn.decompose 에 있습니다.
- Kernel PCA 에서는 다음의 kernel 을 이용할 수 있습니다.
  - "linear" | "poly" | "rbf" | "sigmoid" | "cosine" | "precomputed"
  - default="linear".

Principal Components Analysis

```
from sklearn.decomposition import PCA


pca = PCA(n_components=2)
y_pca = pca.fit_transform(x)
```



Kernel Principal Components Analysis

```
from sklearn.decomposition import KernelPCA

kpca = KernelPCA(
    n_components=2,
    kernel='cosine'
)
y_kpca = kpca.fit_transform(x)
```