



6장 관련 필사

박보영 박지운 오연재

목차

#01 차원축소 기법들

#02 와인 품질 예측 노트북 (classification)

#03 이미지 데이터 차원축소 mnist



1.차원축소 기법들



1.1. 커널 소개

Dataset Decomposition Techniques

Python · [Santander Value Prediction Challenge](#)

Problem Statement: Santander Value Prediction

Santander Group wants to identify the value of transactions for each potential customer. This is a first step that Santander needs to nail in order to personalize their services at scale. The dataset can be downloaded from this [link](#). In this kernel I have explained different approaches for dataset decomposition.

Introduction

The purpose of this kernel is to walkthrough different dataset decomposition techniques and their implementations. Decomposition of dataset into lower dimensions often becomes an important task while dealing with datasets having larger number of features. Dimensionality Reduction refers to the process of converting a dataset having vast dimensions into a dataset with lesser number of dimensions. This process is done by ensuring that the information conveyed by the original dataset is not lost.



“커널의 목적은 다양한 데이터 세트 분해 기술과 그 구현을 살펴보는 것입니다”

☞ PCA, Truncated SVD, ICA, Factor Analysis, NMF, Random Projection, t-SNE with 논문

1.2. 데이터셋

Rows: 4459, Columns: 4991

	48df886f9	0deb4b6a8	34b15f335	a8cb14b00	2f0771a37	30347e683	d08d1fbe3	6ee66e115	20aa07010	dc5a8f1d8	...	3ecc09859	9281abeea	8675bec0b	3
0	0.0	0	0.0	0	0	0	0	0	0.0	0.0	...	0.0	0.0	0.0	
1	0.0	0	0.0	0	0	0	0	0	2200000.0	0.0	...	0.0	0.0	0.0	
2	0.0	0	0.0	0	0	0	0	0	0.0	0.0	...	0.0	0.0	0.0	
3	0.0	0	0.0	0	0	0	0	0	0.0	0.0	...	0.0	0.0	0.0	
4	0.0	0	0.0	0	0	0	0	0	2000000.0	0.0	...	0.0	0.0	0.0	

```
feature_df = train.describe().T
feature_df = feature_df.reset_index().rename(columns = {'index' : 'columns'})
feature_df['distinct_vals'] = feature_df['columns'].apply(lambda x : len(train[x].value_counts()))
feature_df['column_var'] = feature_df['columns'].apply(lambda x : np.var(train[x]))
feature_df['column_std'] = feature_df['columns'].apply(lambda x : np.std(train[x]))
feature_df['column_mean'] = feature_df['columns'].apply(lambda x : np.mean(train[x]))
feature_df['target_corr'] = feature_df['columns'].apply(lambda x : np.corrcoef(target, train[x])[0][1])
feature_df.head()
```

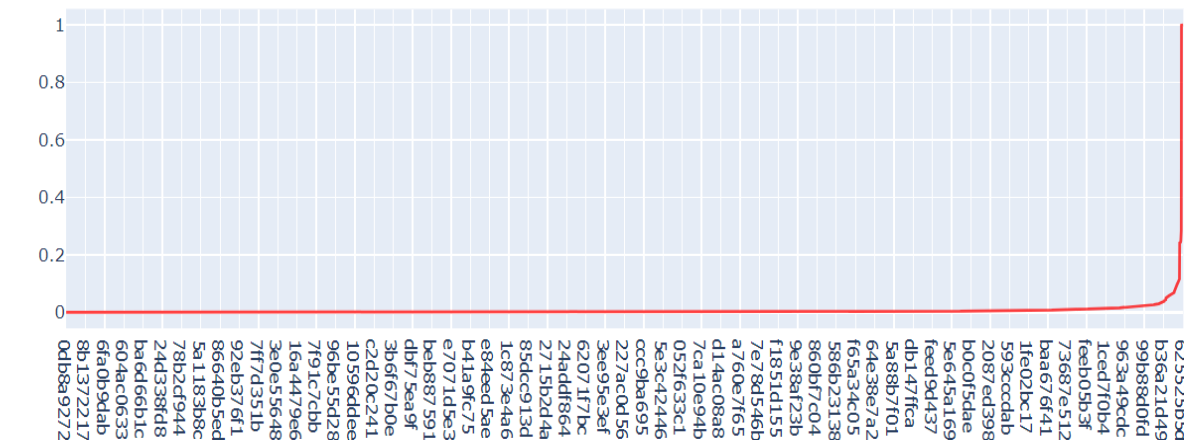
	columns	count	mean	std	min	25%	50%	75%	max	distinct_vals	column_var	column_std	column_mean	target_corr
0	48df886f9	4459.0	14654.930101	3.893298e+05	0.0	0.0	0.0	0.0	20000000.0	32	1.515437e+11	3.892862e+05	14654.930101	0.010188
1	0deb4b6a8	4459.0	1390.894819	6.428302e+04	0.0	0.0	0.0	0.0	4000000.0	5	4.131381e+09	6.427582e+04	1390.894819	0.013805
2	34b15f335	4459.0	26722.450922	5.699652e+05	0.0	0.0	0.0	0.0	20000000.0	29	3.247875e+11	5.699013e+05	26722.450922	0.014694
3	a8cb14b00	4459.0	4530.163714	2.359124e+05	0.0	0.0	0.0	0.0	14800000.0	3	5.564218e+10	2.358860e+05	4530.163714	-0.002917
4	2f0771a37	4459.0	26409.957390	1.514730e+06	0.0	0.0	0.0	0.0	100000000.0	6	2.293893e+12	1.514560e+06	26409.957390	0.016647

```
len(feature_df[feature_df['column_var'].astype(float) == 0.0])
```

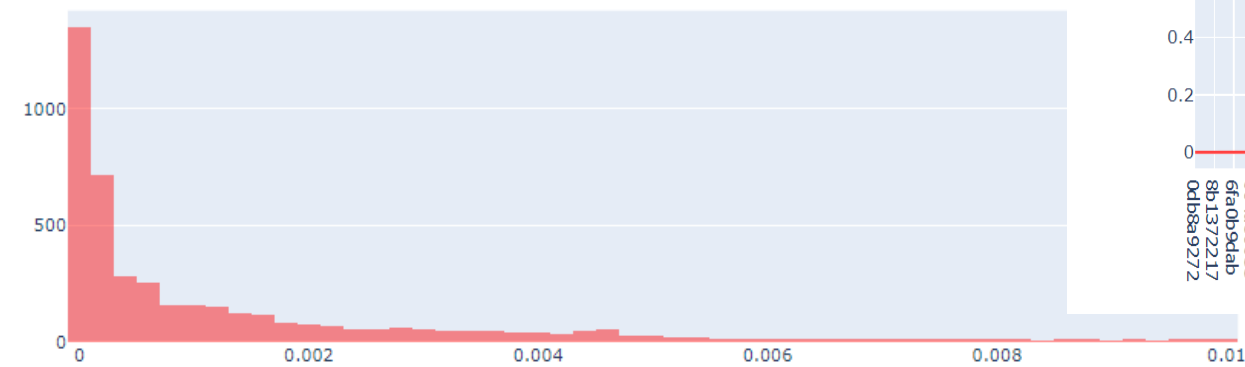
1.3.분포도

```
feature_df = feature_df.sort_values('column_var', ascending = True)
feature_df['column_var'] = (feature_df['column_var'] - feature_df['column_var'].min()) / (feature_df['column_var'].max() - feature_df['column_var'].min())
trace1 = go.Scatter(x=feature_df['columns'], y=feature_df['column_var'], opacity=0.75, marker=dict(color="red"))
layout = dict(height=400, title='Feature Variance', legend=dict(orientation="h"));
fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);
```

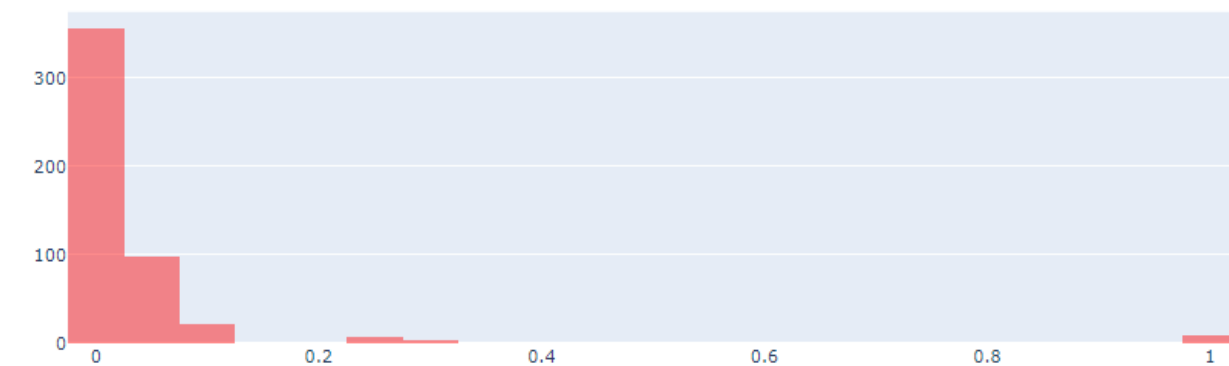
Feature Variance



Distribution of Variable Variance <= 0.01

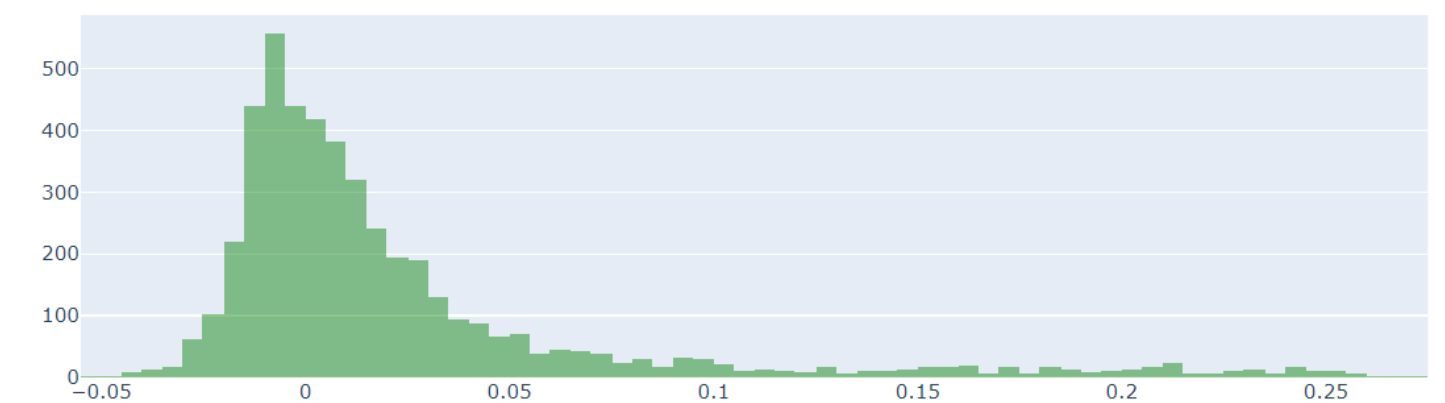


Distribution of Variable Variance > 0.01



```
trace1 = go.Scatter(x=feature_df['columns'], y=feature_df['_corr'], opacity=0.45, marker=dict(color="green"))
layout = dict(height=400, title='Distribution of correlation with target', legend=dict(orientation="h"));
fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);
```

Distribution of correlation with target



Pearson의 상관 계수는 두 개의 연속 변수 간의 통계적 관계 또는 연관성을 측정하는 검정 통계.

1.4. $T(V) = \lambda V$ Decomposition into EigenVectors and EigenValues

```
# Calculating Eigenvectors and eigenvalues of Cov matrix
mean_vec = np.mean(standardized_train, axis=0)
cov_matrix = np.cov(standardized_train.T)
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)

# Create a list of (eigenvalue, eigenvector) tuples
eig_pairs = [ (np.abs(eig_vals[i]),eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the eigenvalue, eigenvector pair from high to low
eig_pairs.sort(key = lambda x: x[0], reverse=True)

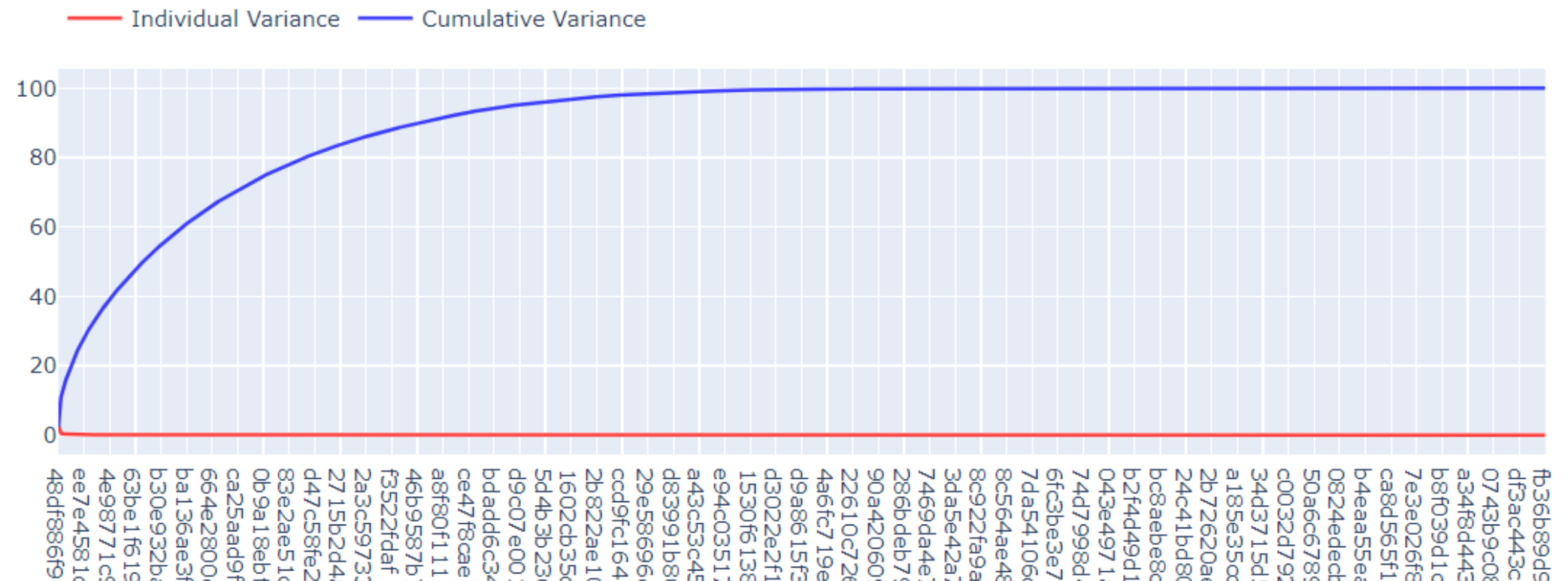
# Calculation of Explained Variance from the eigenvalues
tot = sum(eig_vals)

# Individual explained variance
var_exp = [(i/tot)*100 for i in sorted(eig_vals, reverse=True)]
var_exp_real = [v.real for v in var_exp]

# Cumulative explained variance
cum_var_exp = np.cumsum(var_exp)
cum_exp_real = [v.real for v in cum_var_exp]

## plot the variance and cumulative variance
trace1 = go.Scatter(x=train.columns, y=var_exp_real, name="Individual Variance")
trace2 = go.Scatter(x=train.columns, y=cum_exp_real, name="Cumulative Variance")
layout = dict(height=400, title='Variance Explained by Variables', legend=dict(x=100, y=100))
fig = go.Figure(data=[trace1, trace2], layout=layout);
iplot(fig);
```

Variance Explained by Variables



1.5. PCA

Kernel PCA-비선형적인 차원을 축소

Incremental PCA-데이터의 Singular Value Decomposition을 사용한 선형 차원 축소, 가장 중요한 특이 벡터만 유지하여 데이터를 더 낮은 차원 공간에 투영

Sparse PCA-데이터를 최적으로 재구성할 수 있는 희소 구성 요소 집합을 찾는다

Mini Batch Sparse PCA- 데이터를 최적으로 재구성할 수 있는 희소 구성 요소 집합을 찾는다

Research on Face Recognition Technology Based on PCA and SVM

2022 7th International Conference on Big Data Analytics (ICBDA) Big Data Analytics (ICBDA), 2022
7th International Conference on. :248-252 Mar, 2022

A Mechanical Impact Fault Detection Method Based on PCA for Marine Current Turbine

2021 CAA Symposium on Fault Detection, Supervision, and Safety for Technical Processes (SAFEPROCESS) Fault Detection, Supervision, and Safety for Technical Processes (SAFEPROCESS),
2021 CAA Symposium on. :1-5 Dec, 2021



1.5. PCA

```
def _get_number_components(model, threshold):
    component_variance = model.explained_variance_ratio_
    explained_variance = 0.0
    components = 0

    for var in component_variance:
        explained_variance += var
        components += 1
        if(explained_variance >= threshold):
            break
    return components

### Get the optimal number of components
pca = PCA()
train_pca = pca.fit_transform(standardized_train)
components = _get_number_components(pca, threshold=0.85)
components
```

```
def plot_3_components(x_trans, title):
    trace = go.Scatter3d(x=x_trans[:,0], y=x_trans[:,1], z = x_trans[:,2],
                        name = target, mode = 'markers', text = target, showlegend = False,
                        marker = dict(size = 8, color=x_trans[:,1],
                        line = dict(width = 1, color = '#f7f4f4'), opacity = 0.5))

    layout = go.Layout(title = title, showlegend= True)
    fig = dict(data=[trace], layout=layout)
    iplot(fig)

def plot_2_components(x_trans, title):
    trace = go.Scatter(x=x_trans[:,0], y=x_trans[:,1], name=target, mode='markers',
                      text = target, showlegend = False,
                      marker = dict(size = 8, color=x_trans[:,1], line = dict(width = 1, color = '#fefefe'), opacity = 0.7))
    layout = go.Layout(title = title, hovermode= 'closest',
                      axis=dict(title= 'First Component',
                                ticklen = 5, zeroline= False, gridwidth= 2),|
                      yaxis=dict(title= 'Second Component',
                                ticklen = 5, gridwidth = 2), showlegend= True)
    fig = dict(data=[trace], layout=layout)
    iplot(fig)
```

993

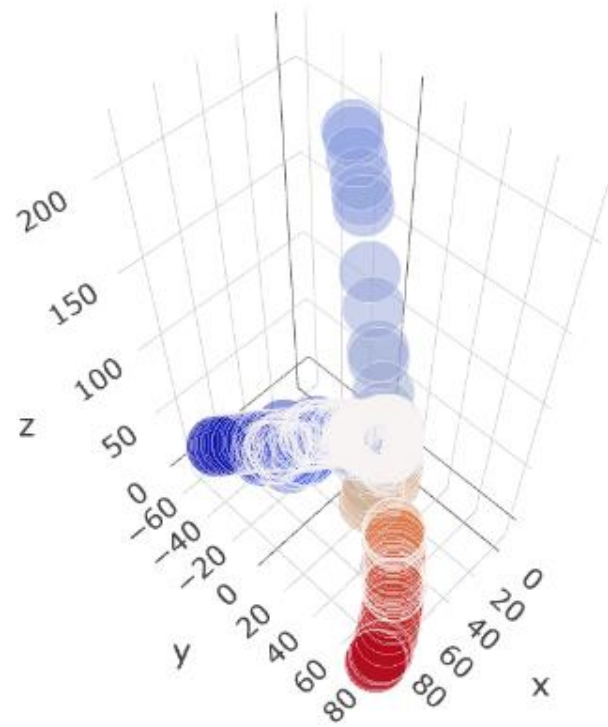
N개의 구성 요소가 있는 PCA를 사용하고 설명된 분산의 임계값과 일치하는 올바른 수를 얻는다
993개의 구성 요소는 데이터 세트 분산의 약 85%를 설명한다(임계값 0.85)

1.5. PCA

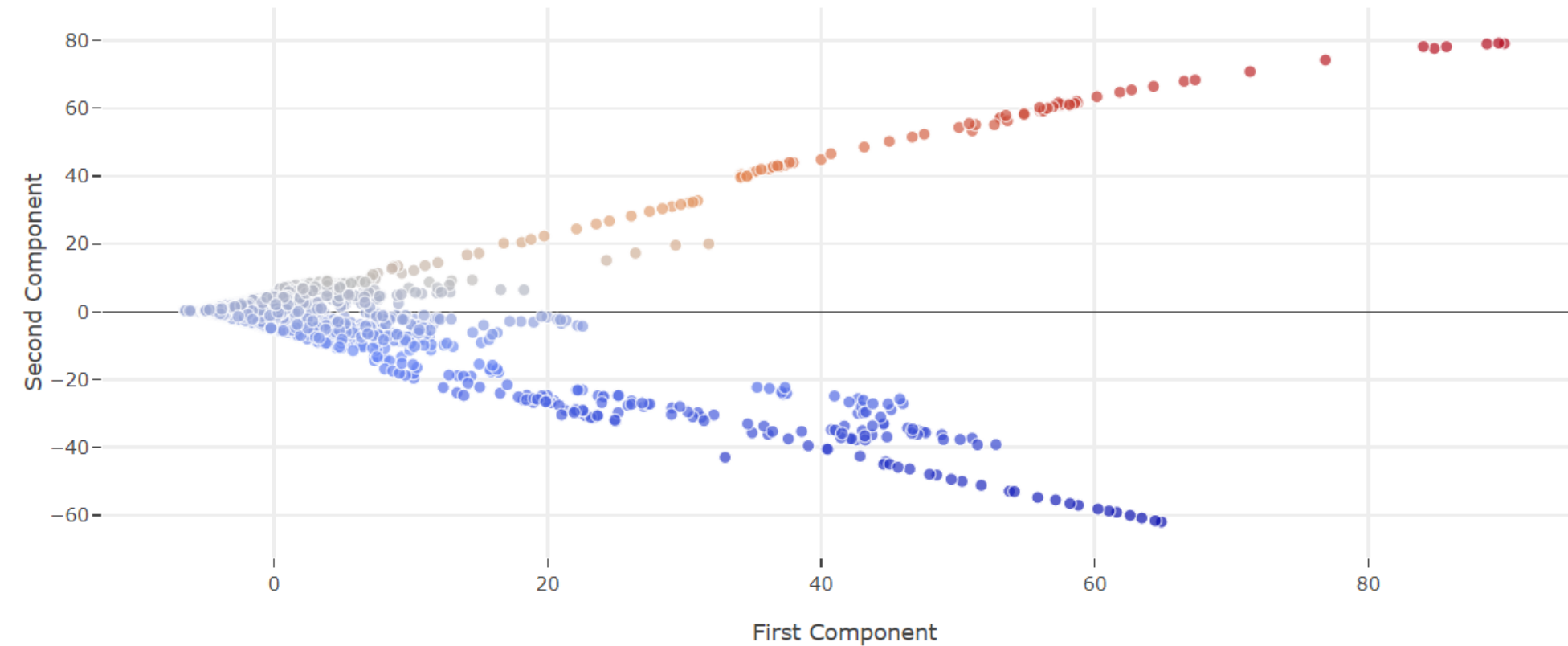
```
### Implement PCA
obj_pca = model = PCA(n_components = components)
X_pca = obj_pca.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_pca, 'PCA - First Three Component')
plot_2_components(X_pca, 'PCA - First Two Components')
```

PCA - First Three Component



PCA - First Two Components



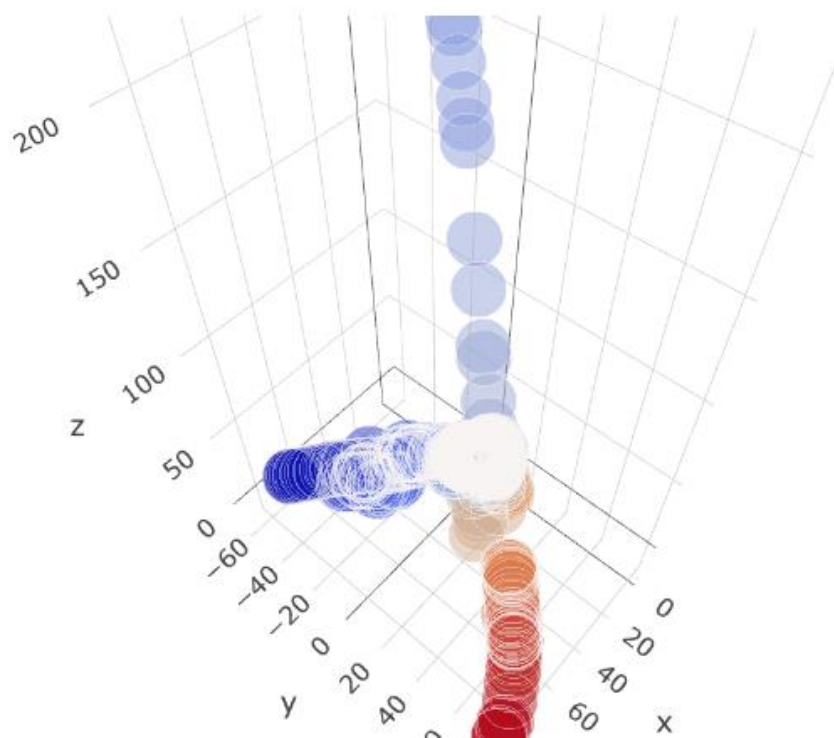
구성 요소의 주요 속성이 서로 직교한다는 것을 관찰할 수 있다. 즉, 상관 관계가 없음을 의미한다

1.6. Truncated SVD

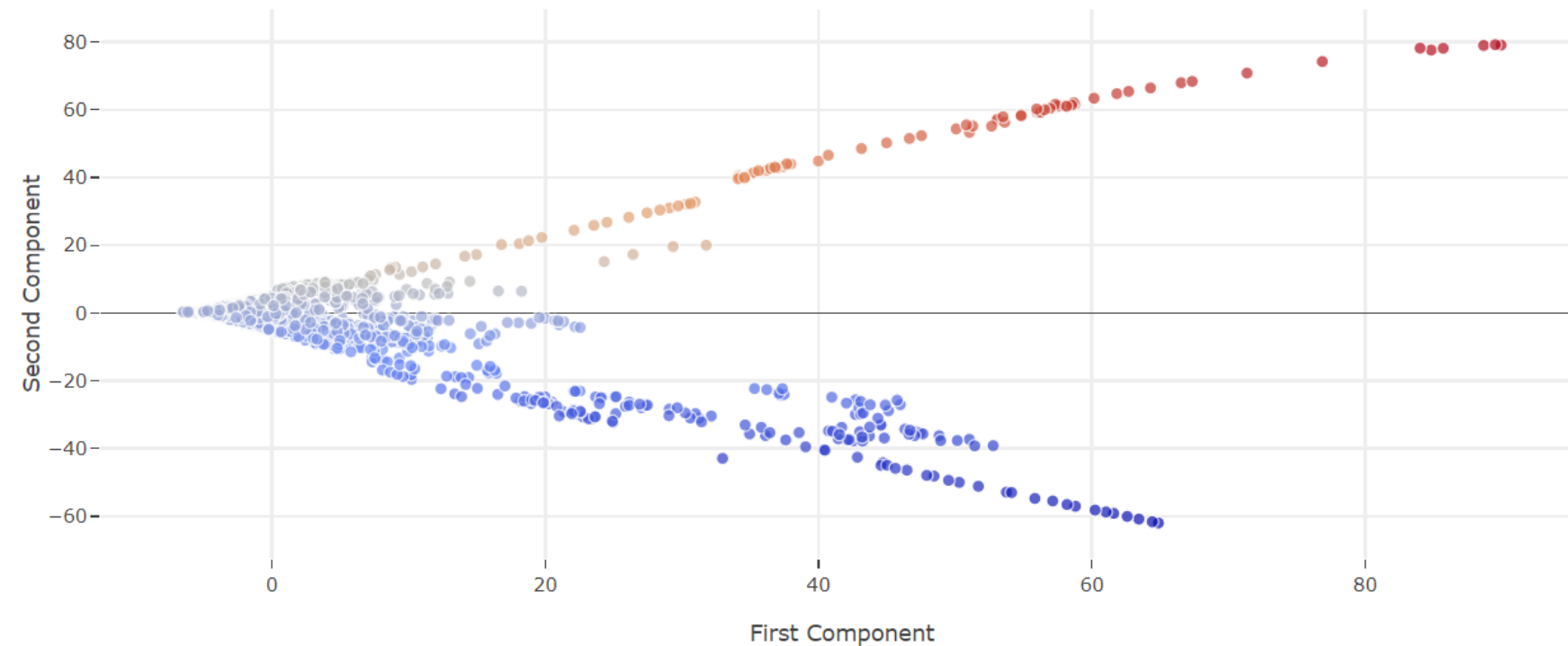
```
### Implement Truncated SVD
obj_svd = TruncatedSVD(n_components = components)
X_svd = obj_svd.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_svd, 'Truncated SVD - First three components')
plot_2_components(X_svd, 'Truncated SVD - First two components')
```

Truncated SVD - First three components



Truncated SVD - First two components



PCA와 Truncated SVD의 결과가 유사==데이터가 중앙에 있는 경우 두 클래스가 동일한 작업을 수행한다
TruncatedSVD는 메모리를 너무 많이 사용하지 않고는 중앙에 배치할 수 없는 대규모 희소 데이터 세트에 유용하다

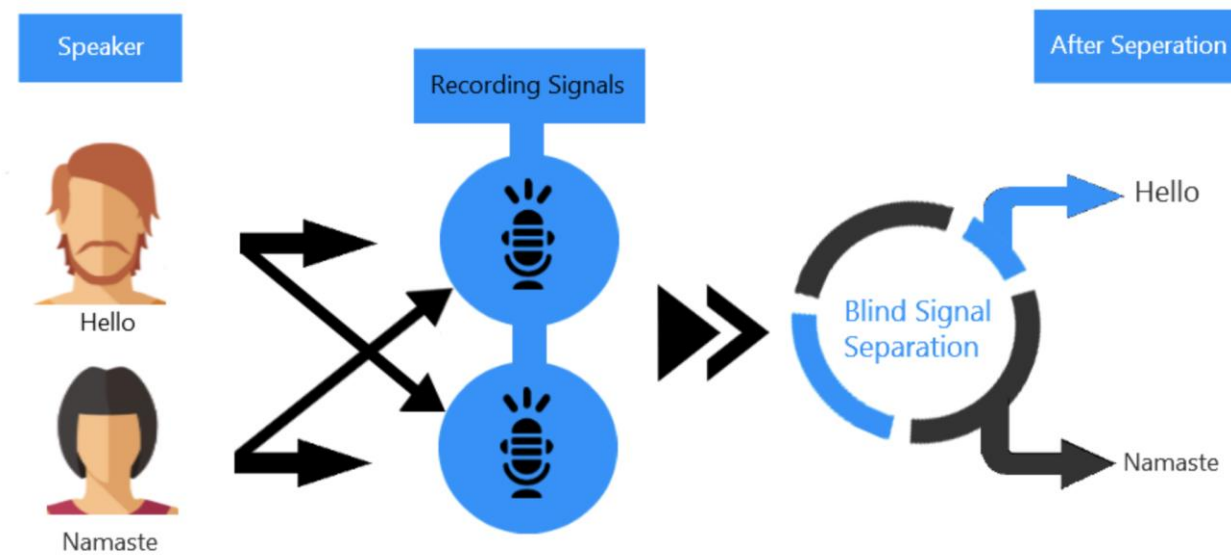
A News Recommendation Algorithm Based on SVD and Improved K-means

2021 International Conference on Networking, Communications and Information Technology (NetCIT) NETCIT Networking, Communications and Information Technology (NetCIT), 2021 International Conference on. :130-134 Dec, 2021

Human Point Cloud Data Segmentation based on Normal Vector Estimation using PCA-SVD Approaches for Elderly Activity Daily Living Detection

TENCON 2021 - 2021 IEEE Region 10 Conference (TENCON) Region 10 Conference (TENCON), TENCON 2021 - 2021 IEEE. :632-636 Dec, 2021

1.7. Independent Component Analysis - ICA



$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t)$$

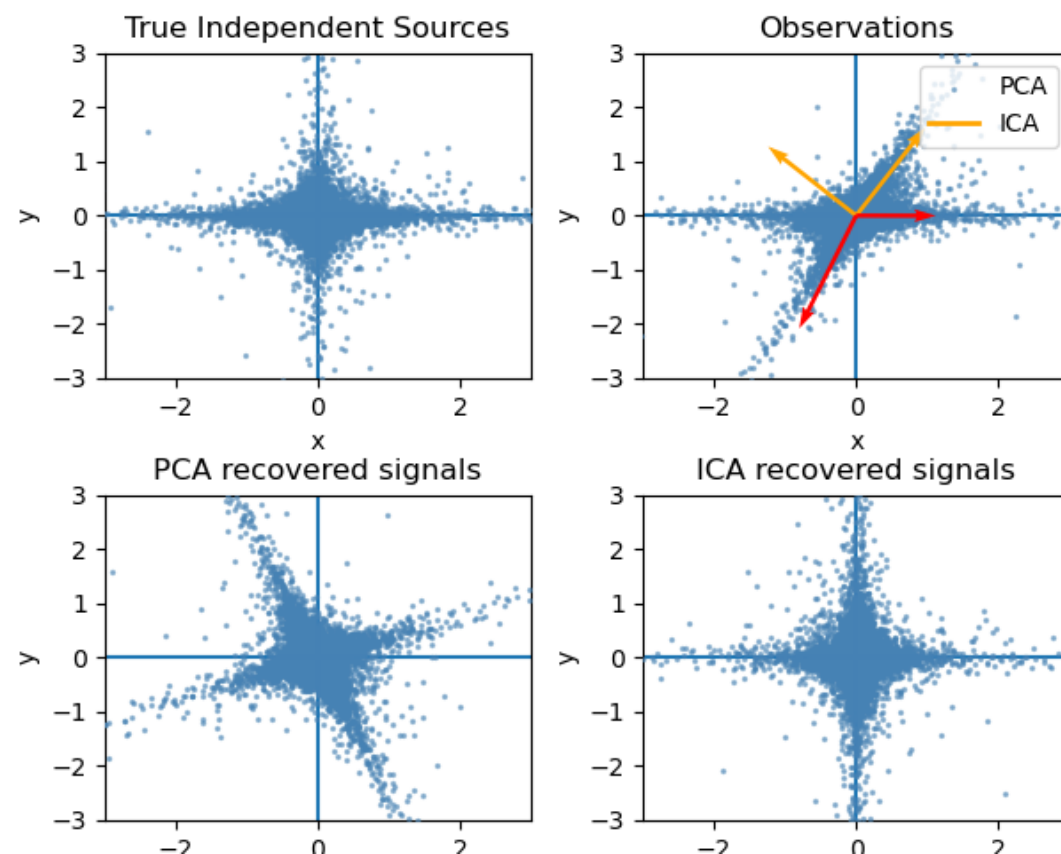
$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t)$$

$$x = As$$

$$s = A^{-1}x = Wx$$

독립 성분 분석(ICA)은 다변량의 신호를 통계적으로 독립적인 하부 성분으로 분리하는 계산 방법

가정: source들이 서로 독립적이다



공통점은 데이터를 대표하는 basis vector를 찾는다는 것

PCA는 데이터를 정사영했을 때 최대분산을 얻을 수 있는 벡터와 직교하는 벡터를 basis로 삼는다

ICA는 데이터를 정사영했을 때 최대독립을 얻을 수 있는 벡터를 basis로 삼고, 서로 직교하지 않을 수도 있다

1.7. ICA

Use of ICA to Separate Micro-Doppler Signatures in ISAR Images of Aircraft That Has Fast-Rotating Parts

IEEE Transactions on Aerospace and Electronic Systems IEEE Trans. Aerosp. Electron. Syst. Aerospace and Electronic Systems, IEEE Transactions on. 58(1):234-246 Feb, 2022

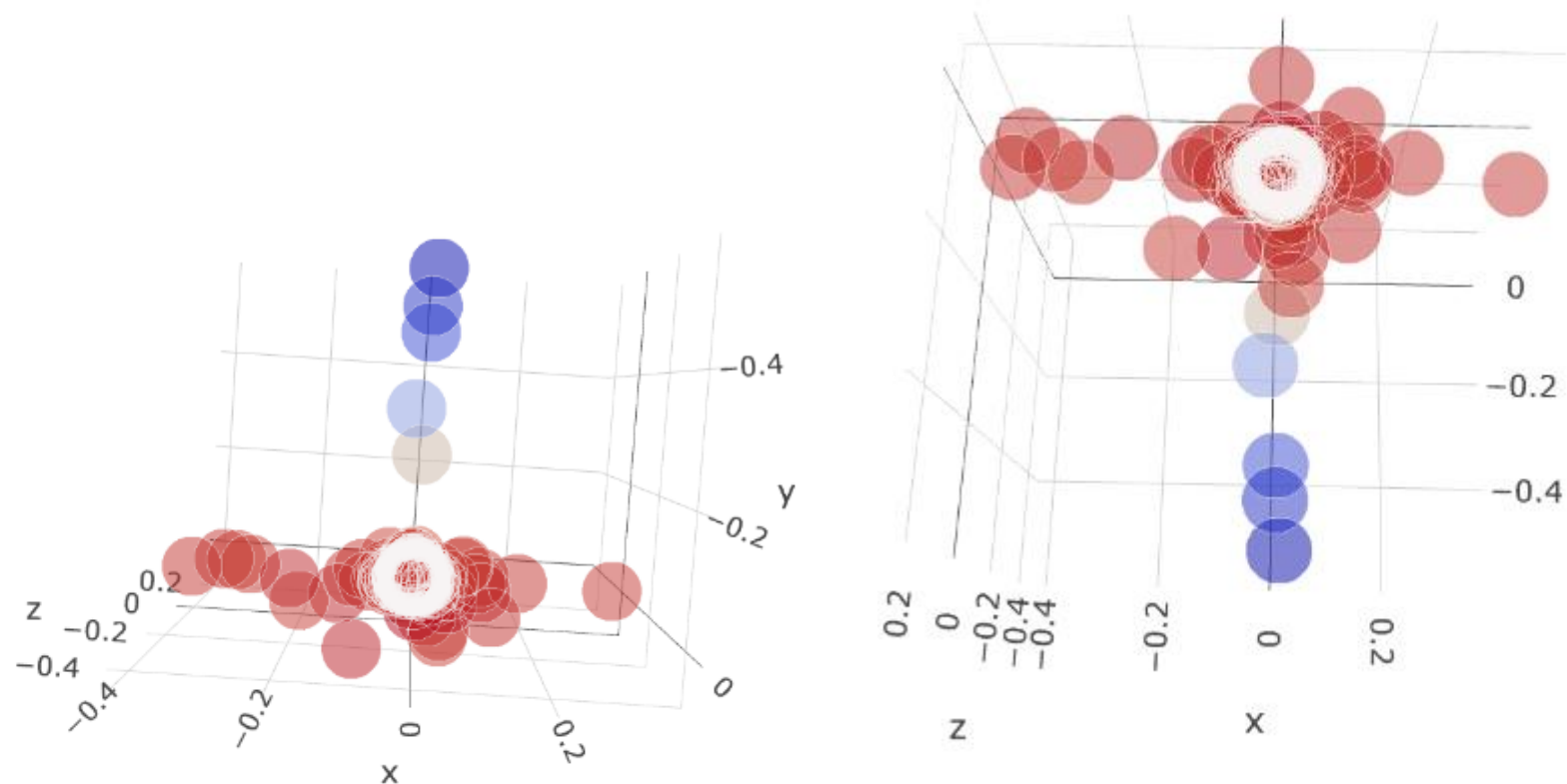
```
### Implement ICA
obj_ica = FastICA(n_components = 30)
X_ica = obj_ica.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_ica, 'ICA - First three components')
plot_2_components(X_ica, 'ICA - First two components')
```

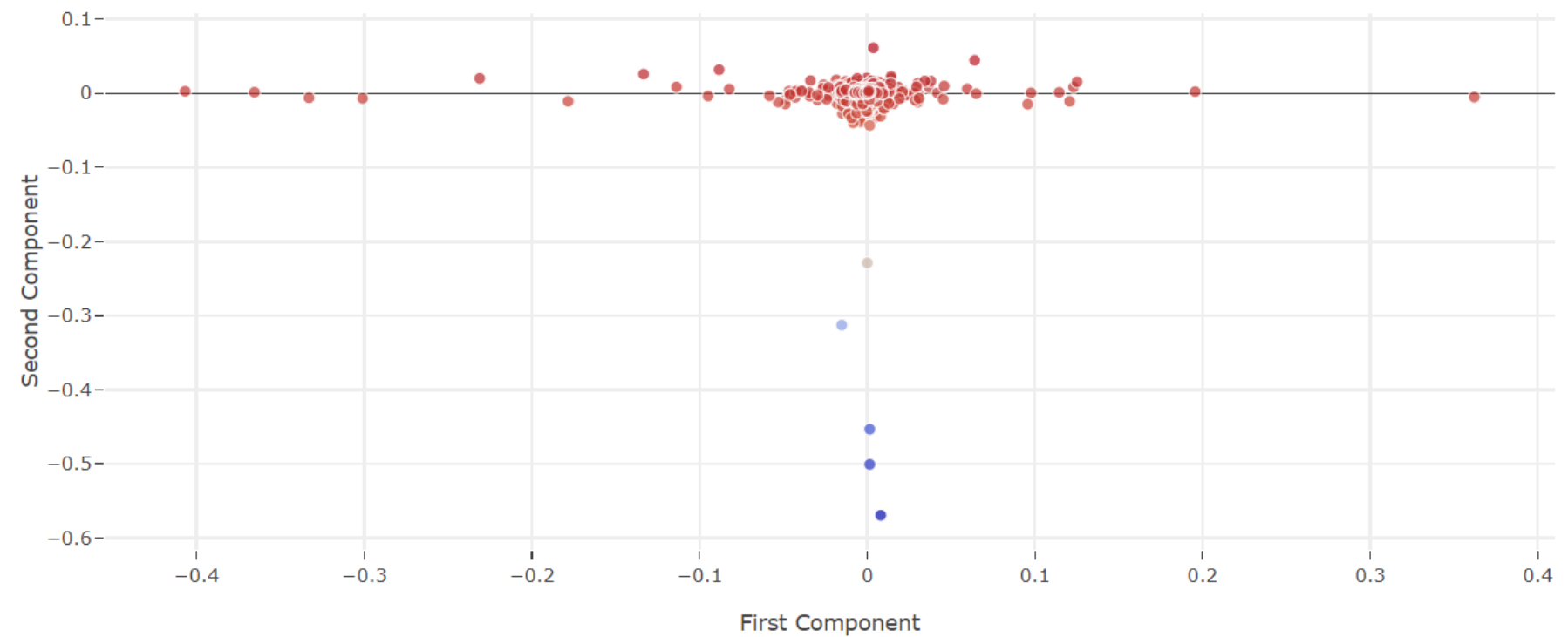
Using ICA as a Pre-Processor for CNNs to Efficiently Process High-Dynamic Videos

2021 17th International Conference on Computational Intelligence and Security (CIS) CIS Computational Intelligence and Security (CIS), 2021 17th International Conference on. :585-589 Nov, 2021

ICA - First three components



ICA - First two components



정사영 했을 때 수직이 된다. 최대독립을 만드는 basis는 직교하지 않을 수 있다

1.8. Factor Analysis

요인분석은 수많은 변수들 중에서 잠재된 몇 개의 변수(요인)을 찾아내는 것이다

가정:

1. 데이터에는 특이치가 없다.
2. 표본 크기는 요인보다 커야 한다.
3. 완벽한 다중 공선성은 없어야 한다.
(변수들 간 상관관계가 있지만, 다른 변수의 선형결합으로 표현되면 안됨)
4. 변수들 사이에 동질성이 있어서는 안 된다.

이 데이터가 수학, 과학, 영어, 중국어, 독어, 작곡, 연주 의 점수의 7개 변수로 구성되어 있다고 하면,
수학, 과학(수리계산능력) 영어, 중국어, 독어 (외국어능력) 작곡, 연주 (음악적능력) 상관관계가 있다

내부적으로는 3개의 잠재변수(latent variable)으로 구성된다

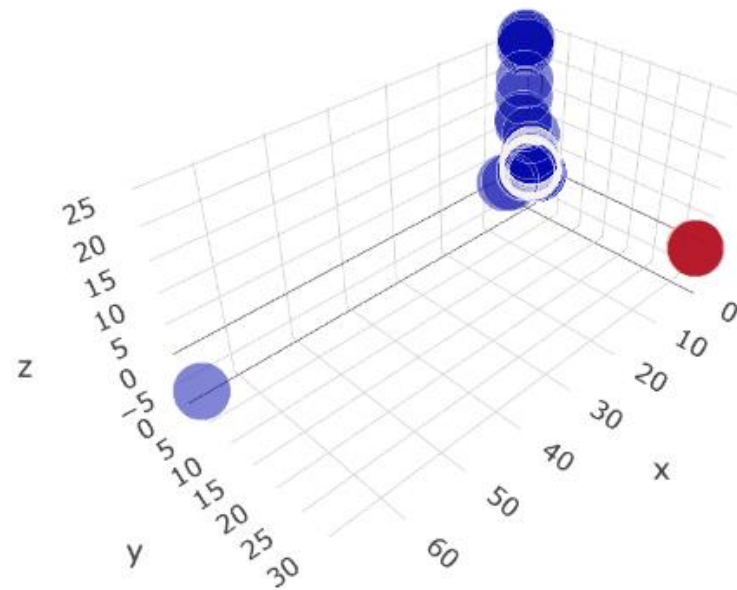
데이터에 상관성이 높은 변수들을 묶어 잠재된 몇 개의 변수를 찾는 것이 목적
요인분석은 데이터 축소(Data Reduction)과 관계가 있다.

1.8. Factor Analysis

```
### Implement Factor Analysis
obj_fa = FactorAnalysis(n_components = 30)
X_fa = obj_fa.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_fa, 'Factor Analysis - First three components')
```

Factor Analysis - First three components



The Knowledge Map and Exploratory Factor Analysis for University Staff's Information and Communication Technology Competence: A Case Study in Chiang Mai University

2022 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON) Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON), 2022 Joint International Conference on. :169-174 Jan, 2022

잠재 공간의 차원, 변환 후 얻은 X의 구성 요소 수가 30개이고, 그 중 3개의 주요 잠재요소의 그래프이다

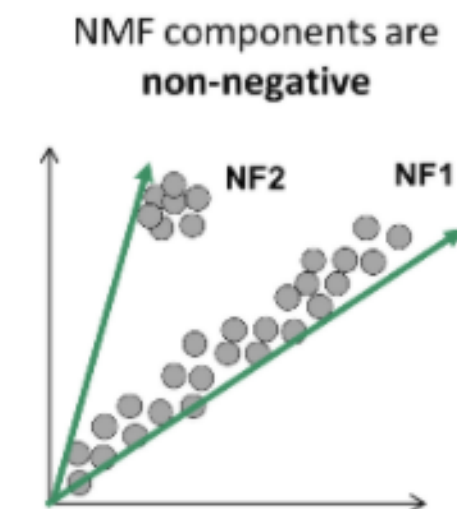
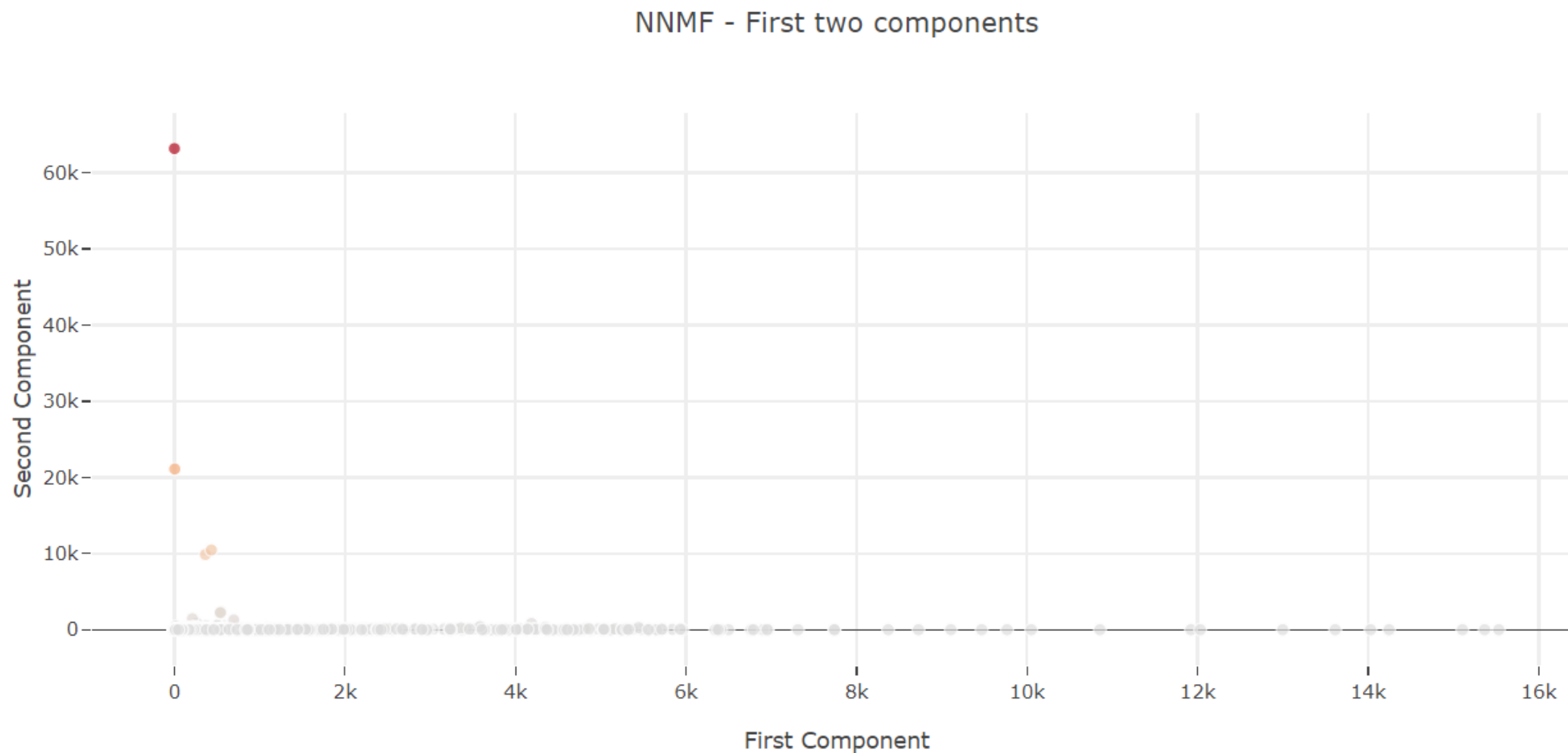
1.9. NMF (Non Negative Matrix Factorization)

```
### Implement NonNegative Matrix Factorization
obj = NMF(n_components = 2)
X_nmf = obj.fit_transform(train)

## Visualize the Components
plot_2_components(X_nmf, 'NNMF - First two components')
```

A Novel NMF Guided for Hyperspectral Unmixing From Incomplete and Noisy Data

IEEE Transactions on Geoscience and Remote Sensing IEEE Trans. Geosci. Remote Sensing
Geoscience and Remote Sensing, IEEE Transactions on. 60:1-15 2022



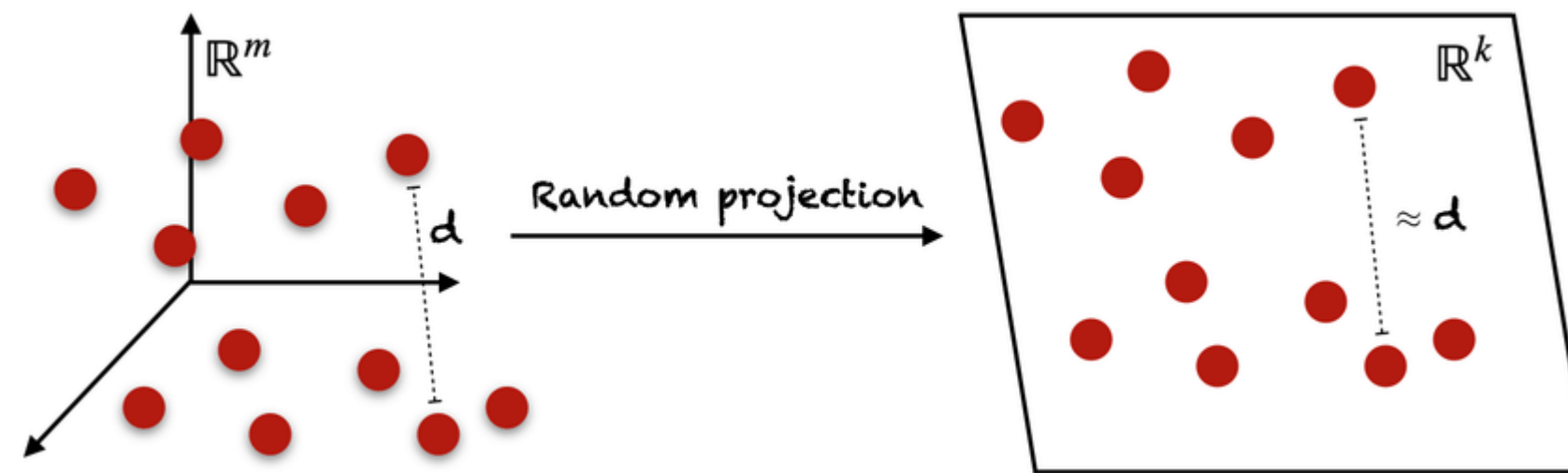
양수가 되게 하는 basis를 선택했을 때 대부분 first component 축 위에 놓여 있다

1.10. Random Projection

더 빠른 처리 시간과 더 작은 모델 크기를 위해 제어된 정확도(추가 분산으로)를 교환하여 데이터의 차원을 줄이는 간단하고 계산 효율적인 방법을 구현합니다

가우스 랜덤 행렬과 희소 랜덤 행렬의 두 가지 유형

임의의 투영 행렬의 차원과 분포는 데이터 세트의 **두 샘플 사이의 쌍별 거리를 유지가 목적** 따라서 랜덤 투영은 거리 기반 방법에 적합한 근사 기술이다



Random Projection Through the Lens of Data Complexity Indicators

2021 International Conference on Data Mining Workshops (ICDMW) ICDMW Data Mining Workshops (ICDMW), 2021 International Conference on. :482–489 Dec, 2021

1.10.1. Gaussian Random Projection

구성 요소가 $N()$ 분포에서 추출되는 무작위로 생성된 Gaussian Random Projection 행렬에 원래 입력 공간을 투영하여 차원을 줄인다

$$N\left(0, \frac{1}{n_{\text{components}}}\right)$$

Gaussian Random Projection Based Non-invertible Cancelable Biometric Templates

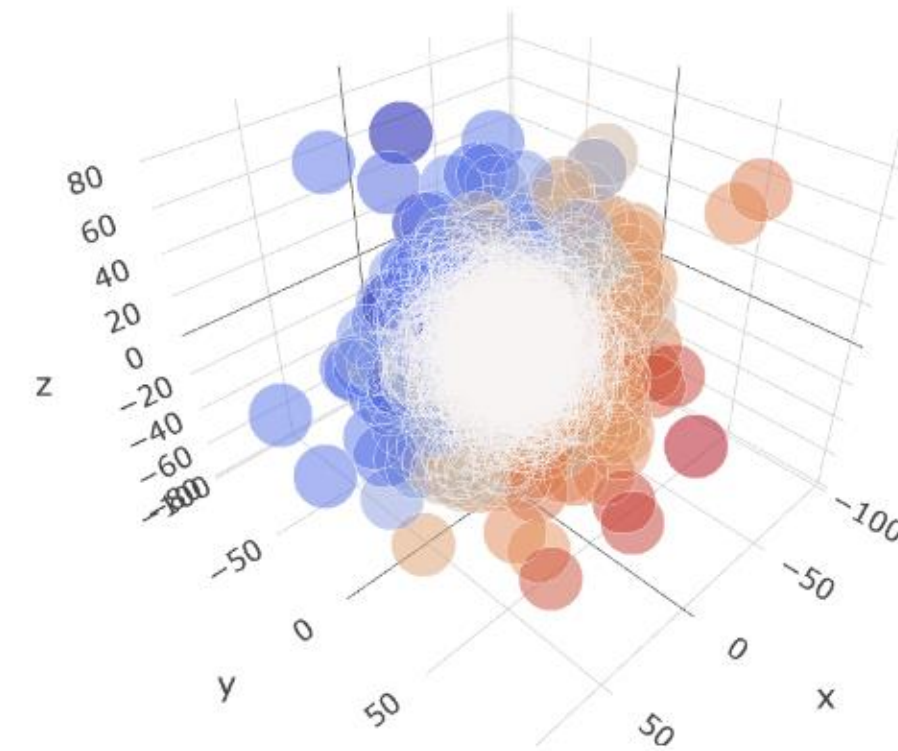
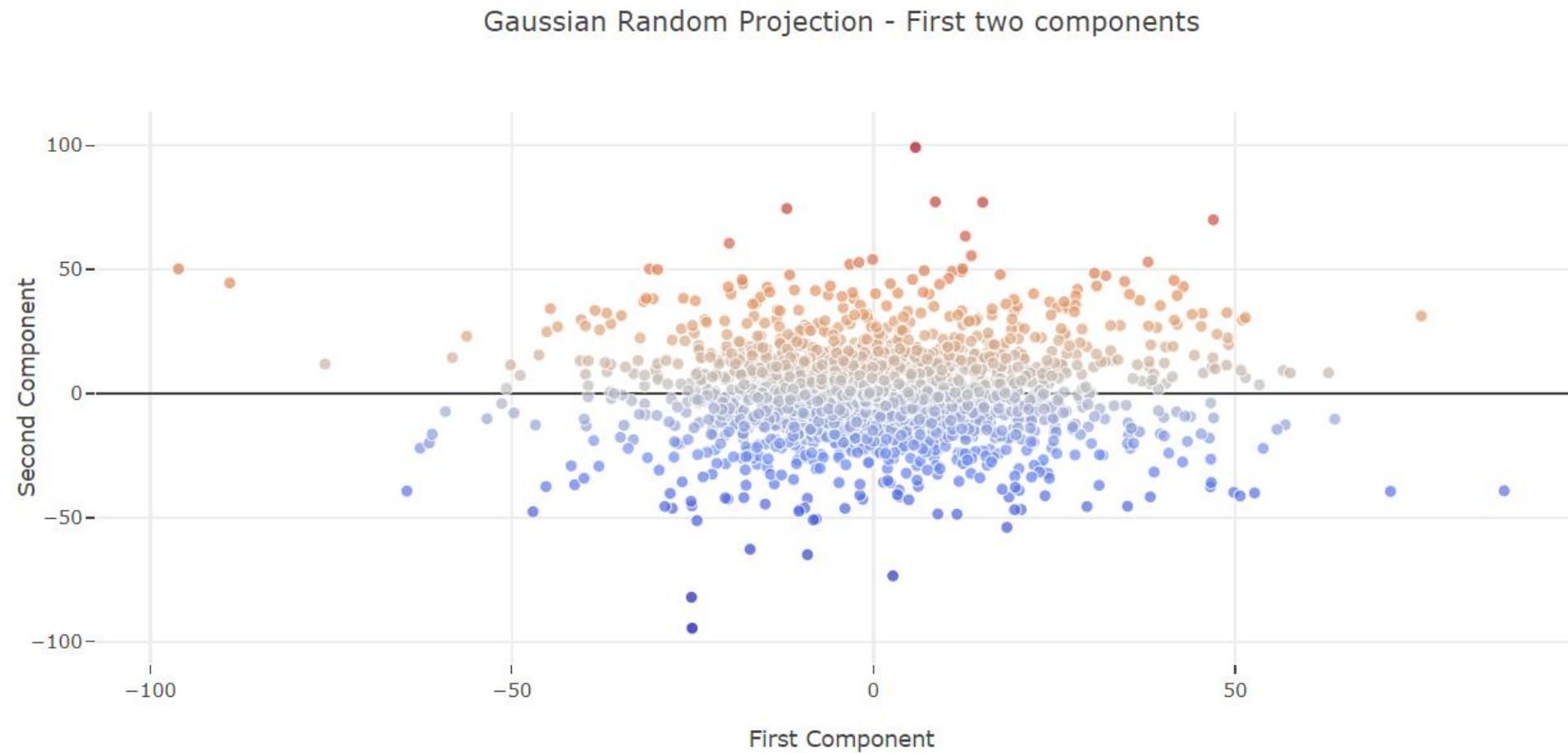
In Procedia Computer Science 2015 54:661-670

1.10.1. Gaussian Random Projection

```
### Implement Gaussian Random Projection
obj_grp = GaussianRandomProjection(n_components = 30, eps=0.1)
X_grp = obj_grp.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_grp, 'Gaussian Random Projection - First three components')
plot_2_components(X_grp, 'Gaussian Random Projection - First two components')
```

Gaussian Random Projection - First three components



1.10.2. Sparse Random Projection

희소 랜덤 행렬을 사용하여 원래 Sparse Random Projection 입력 공간을 투영하여 차원을 줄인다 유사한 임베딩embedding 품질을 보장하는 동시에 훨씬 더 메모리 효율적이고 투영된 데이터의 더 빠른 계산을 허용하는 조밀한dense 가우스 랜덤 투영 행렬의 대안이다.

$$\begin{cases} -\sqrt{\frac{s}{n_{\text{components}}}} & 1/2s \\ 0 & \text{with probability } 1 - 1/s \\ +\sqrt{\frac{s}{n_{\text{components}}}} & 1/2s \end{cases} \quad s = 1 / \text{density}$$

Binary Models for Motor-Imagery Brain-Computer Interfaces: Sparse Random Projection and Binarized SVM

2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)
Artificial Intelligence Circuits and Systems (AICAS), 2020 2nd IEEE International Conference on. :163-167 Aug, 2020

3D face recognition using facial curves, sparse random projection and fuzzy similarity measure

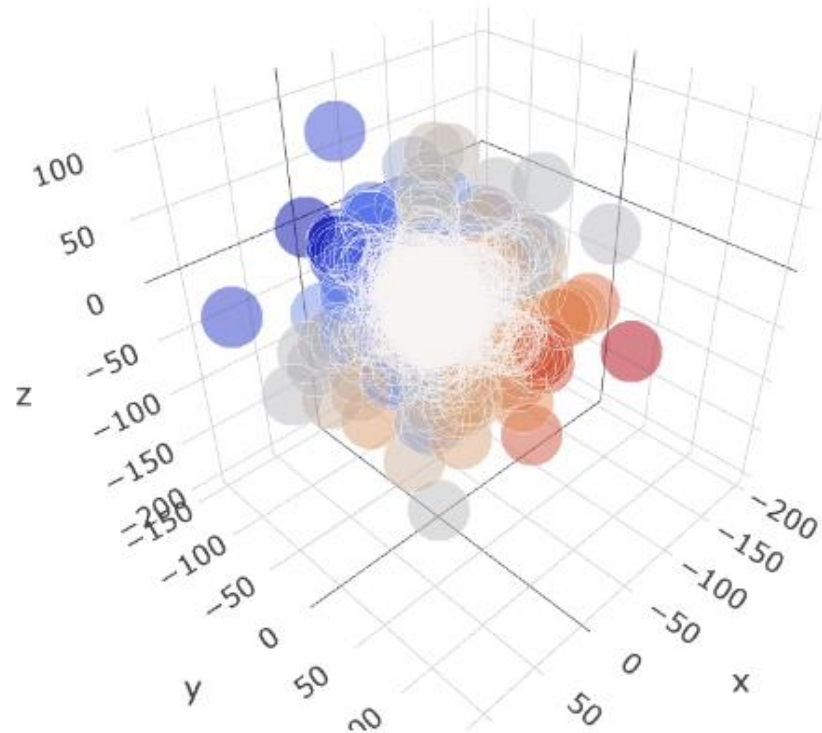
2014 Third IEEE International Colloquium in Information Science and Technology (CIST) Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in. :317-322 Oct, 2014

1.10.2. Sparse Random Projection

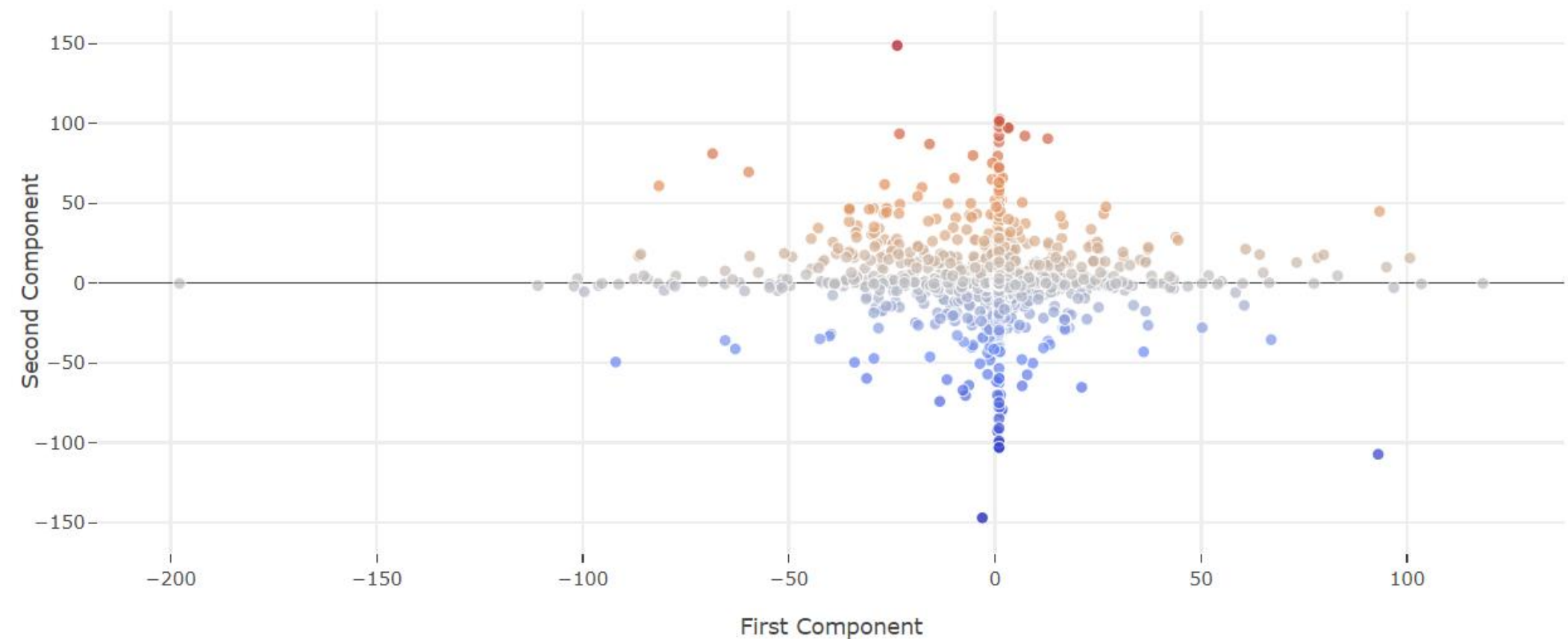
```
### Implement Sparse Random Projection
obj_srp = SparseRandomProjection(n_components = 30, eps=0.1)
X_srp = obj_srp.fit_transform(standardized_train)

## Visualize the Components
plot_3_components(X_srp, 'Sparse Random Projection - First three components')
plot_2_components(X_srp, 'Sparse Random Projection - First two components')
```

Sparse Random Projection - First three component:



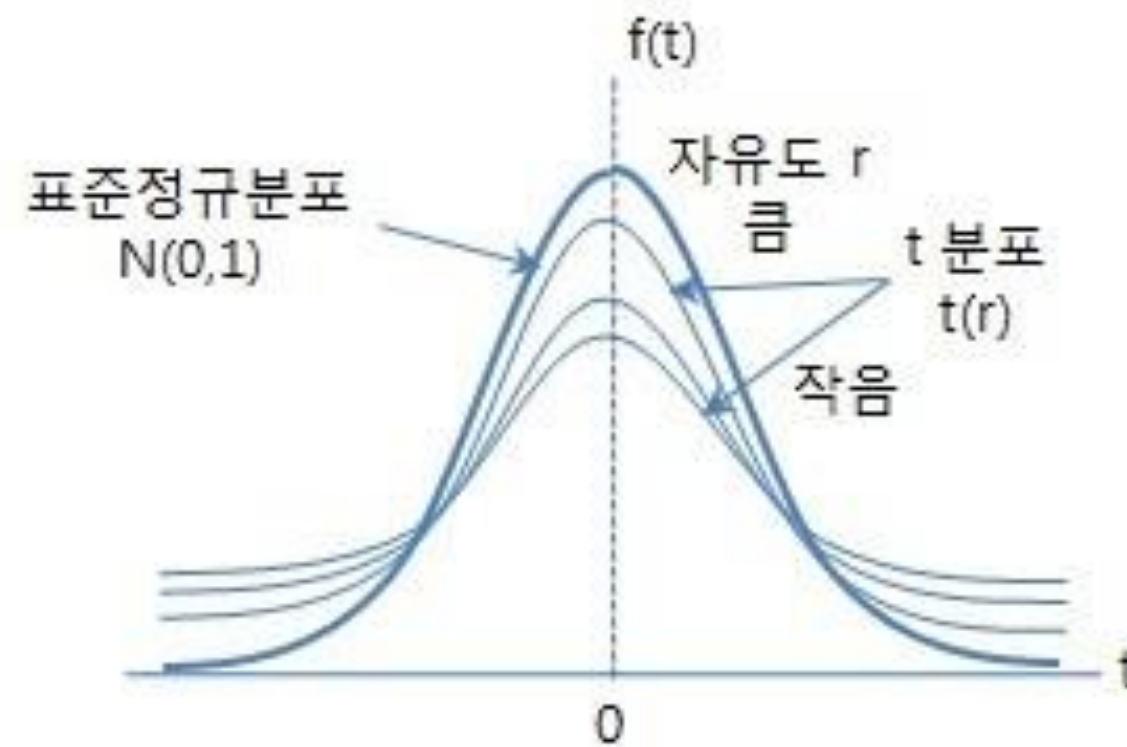
Sparse Random Projection - First two components



밀도를 고려한 무작위한 투영

1.11. SNE(Stochastic Neighbor Embedding)

SNE는 n 차원에 분포된 이산 데이터를 k (n 이하의 정수) 차원으로 축소하며 거리 정보를 보존하되, **거리가 가까운 데이터의 정보를 우선하여 보존하기 위해** 고안되었다 (큰 가중치를 주는 방식)



SNE 학습과정에 사용되는 정규분포는 t 분포에 비해, 거리에 따른 확률 값 변화의 **경사가 가파른** 특징을 갖는다. 따라서 **특정 거리 이상부터는 학습과정에 거의 반영이 되지 않는 문제점**을 가지고 있으며, 이를 **Crowding Problem** 이라고 한다

이러한 문제점을 해결, 보완하기 위해 고안된 방법이 t-SNE이다.

1.12. T-distributed Stochastic Neighbor Embedding

t-SNE은 비선형 관계를 이용한 데이터셋 분해 방법이다

목표는 고차원 공간에서 점point 세트를 가져와 저차원 공간에서 해당 점의 표현을 찾는 것이다

정규분포 대신 t분포를 이용한다

탁월한 성능을 가졌지만 많은 시간을 소요한다

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 4459 samples in 0.016s...
[t-SNE] Computed neighbors for 4459 samples in 0.696s...
[t-SNE] Computed conditional probabilities for sample 1000 / 4459
[t-SNE] Computed conditional probabilities for sample 2000 / 4459
[t-SNE] Computed conditional probabilities for sample 3000 / 4459
[t-SNE] Computed conditional probabilities for sample 4000 / 4459
[t-SNE] Computed conditional probabilities for sample 4459 / 4459
[t-SNE] Mean sigma: 0.184712
[t-SNE] KL divergence after 250 iterations with early exaggeration: 82.259232
[t-SNE] KL divergence after 500 iterations: 2.736388
```

Semi-supervised t-SNE for Millimeter-wave Wireless Localization

2021 7th International Conference on Computer and Communications (ICCC) Computer and Communications (ICCC), 2021 7th International Conference on. :1015-1019 Dec, 2021

A Novel Supervised t-SNE Based Approach of Viseme Classification for Automated Lip Reading

2021 International Conference on Electrical, Computer and Energy Technologies (ICECET) Electrical, Computer and Energy Technologies (ICECET), 2021 International Conference on. :1-7 Dec, 2021

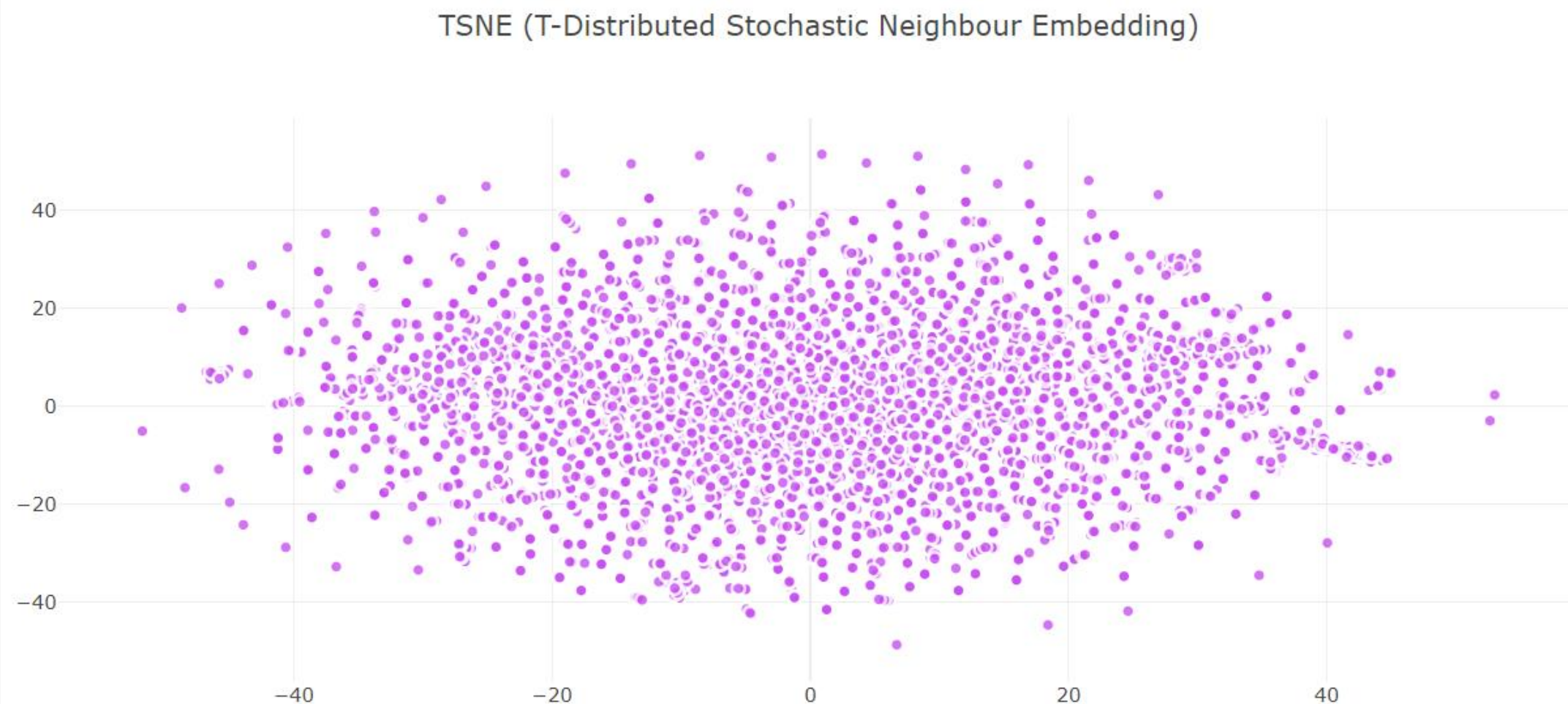
1.12. t-SNE

```
tsne_model = TSNE(n_components=2, verbose=1, random_state=42, n_iter=500)
tsne_results = tsne_model.fit_transform(X_svd)

traceTSNE = go.Scatter(
    x = tsne_results[:,0],
    y = tsne_results[:,1],
    name = target,
    hoveron = target,
    mode = 'markers',
    text = target,
    showlegend = True,
    marker = dict(
        size = 8,
        color = '#c94ff2',
        showscale = False,
        line = dict(
            width = 2,
            color = 'rgb(255, 255, 255)'
        ),
        opacity = 0.8
    )
)
data = [traceTSNE]

layout = dict(title = 'TSNE (T-Distributed Stochastic Neighbour Embedding)',
    hovermode = 'closest',
    yaxis = dict(zero = False),
    xaxis = dict(zero = False),
    showlegend = False,
)

fig = dict(data=data, layout=layout)
iplot(fig)
```



거리를 보존했으므로 원본에서 데이터간 거리를 추측할 수 있다

2. 와인 품질 예측 노트북 필사



#2.1 대회 소개 및 Data Description

와인의 여러가지 화학 성분을 기반으로 와인의 품질을 분류해내는 것이 목표.

<https://www.kaggle.com/code/sonalisingh1411/analysis-pca-red-wine-quality-prediction-87>

- fixed acidity 산도
- volatile acidity 휘발성산
- citric acid 시트르산
- residual sugar 잔당 : 발효 후 와인 속에 남아있는 당분
- chlorides 염화물
- free sulfur dioxide 독립 이산화황
- total sulfur dioxide 총 이산화황
- density 밀도
- pH 수소이온농도
- sulphates 황산염
- alcohol 도수
- quality 품질

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   fixed acidity          1599 non-null   float64
1   volatile acidity       1599 non-null   float64
2   citric acid            1599 non-null   float64
3   residual sugar         1599 non-null   float64
4   chlorides              1599 non-null   float64
5   free sulfur dioxide    1599 non-null   float64
6   total sulfur dioxide   1599 non-null   float64
7   density                1599 non-null   float64
8   pH                    1599 non-null   float64
9   sulphates              1599 non-null   float64
10  alcohol                1599 non-null   float64
11  quality                1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

-> winequality-red.csv 데이터 사용, 12개의 변수와 1599개의 관측치 존재

#2.2 EDA

1. Dupliicated values 찾고 없애기

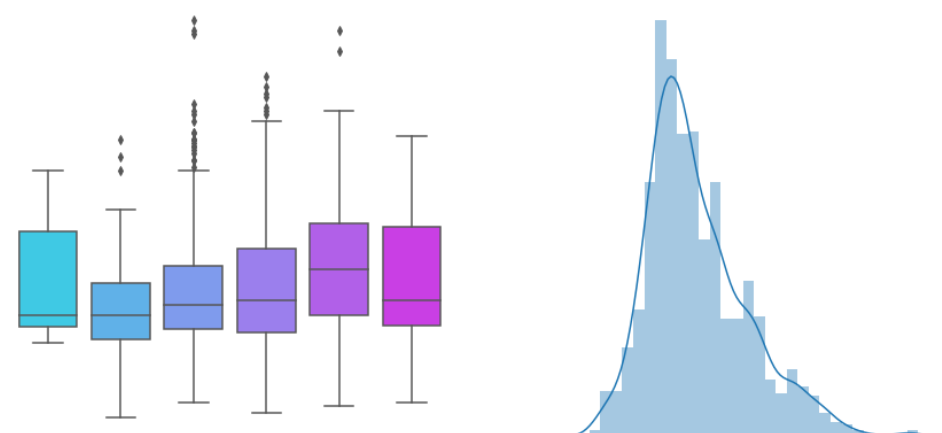
```
df.duplicated().sum() #결과는 240
df.drop_duplicates(inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1359 entries, 0 to 1598
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   fixed acidity          1359 non-null   float64
1   volatile acidity       1359 non-null   float64
2   citric acid            1359 non-null   float64
3   residual sugar         1359 non-null   float64
4   chlorides              1359 non-null   float64
5   free sulfur dioxide    1359 non-null   float64
6   total sulfur dioxide   1359 non-null   float64
7   density                1359 non-null   float64
8   pH                    1359 non-null   float64
9   sulphates              1359 non-null   float64
10  alcohol                1359 non-null   float64
11  quality                1359 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 178.0 KB
```

240개의 observation이 사라진 것을 확인

2. Feature 분석 & Outliers 찾기

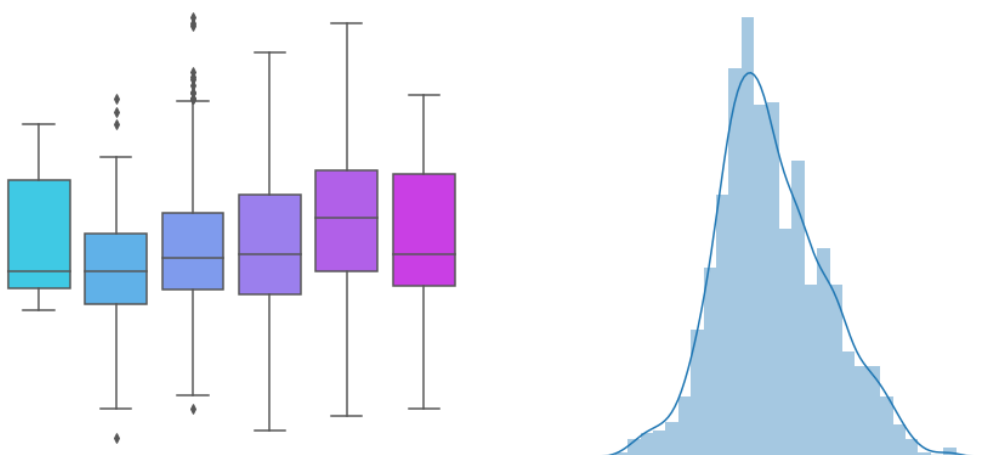
Distribution Of The Fixed Acidity Before Treating Outlier



→ outliers 때문에 분포가 치우친 경향 有
⇒ 변수에 대해서 log transformation 실행!

```
df['Log_fixed acidity'] = np.log(df['fixed acidity'])
```

Distribution Of The Fixed Acidity After Treating Outlier



→ outlier와 분포의 치우침 정도가 조금 완화된 것을 확인할 수 있음

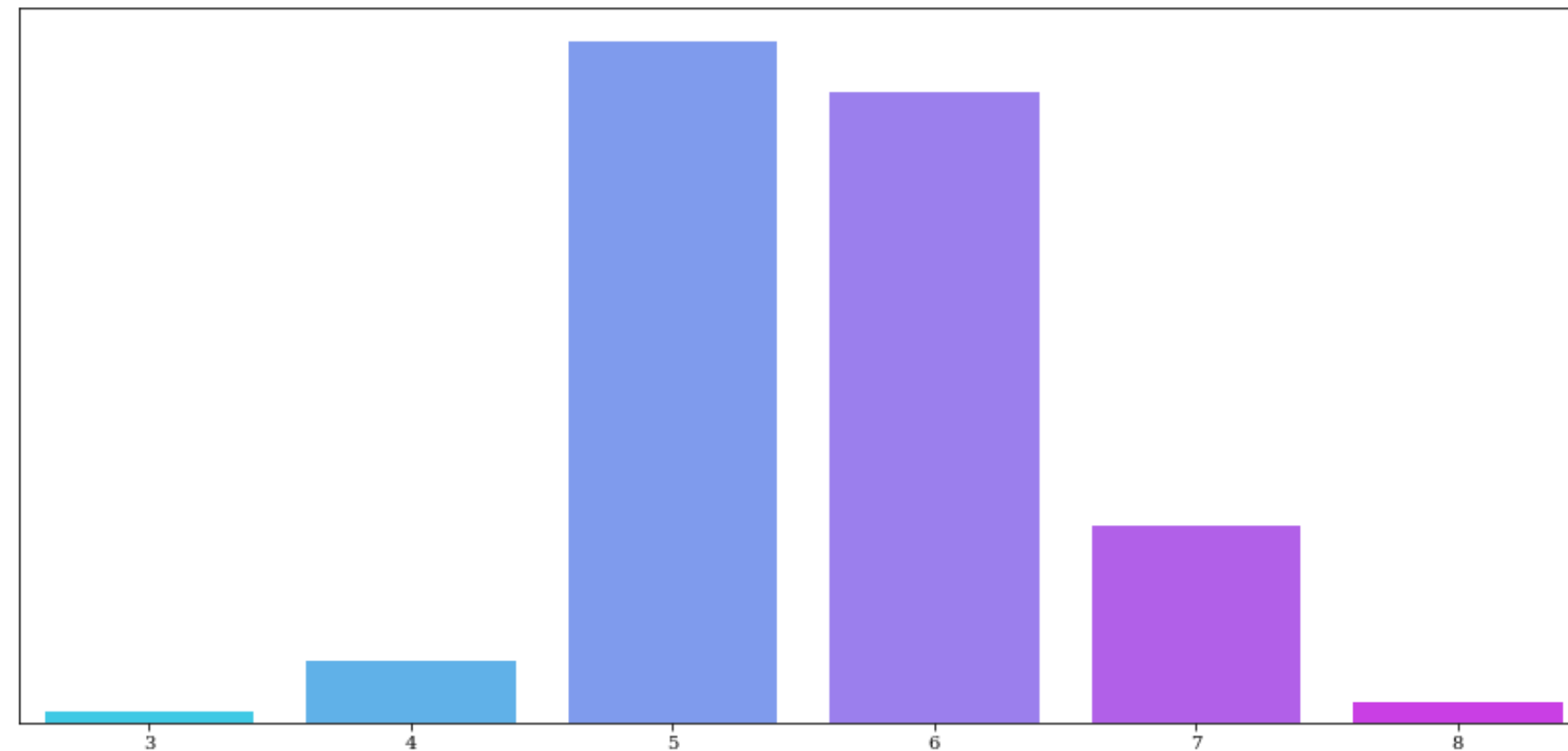
나머지 피쳐들에 대해서도 분포와 outlier를 확인하고 log transformation (노트북에서는 Volatile Acidity, Residual Sugar, Chlorides, Free Sulfur dioxide, Total Sulfur dioxide, Sulphates, Alcohol에 대해서 log transformation을 실행함)

```
list1 = ['volatile acidity', 'residual sugar', 'chlorides', 'free sulfur dioxide',
        'total sulfur dioxide', 'sulphates', 'alcohol']
for i in list1:
    df[i] = np.log(df[i])
```

#2.3 Feature Engineering & Feature Transformation

타겟변수 Quality의 분포도

Before-Distribution Of The Quality



→ 5, 6에 타겟값이 너무 많이 몰려 있음(불균형한 레이블값)

⇒ 이진 분류로 변환하자! (pd.cut을 이용해 bad, good으로만 라벨링하여 feature engineering 수행)

#2.3 Feature Engineering & Feature Transformation

```
#Feature Engineering...
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
df['quality'] = pd.cut(df['quality'], bins = bins, labels = group_names)

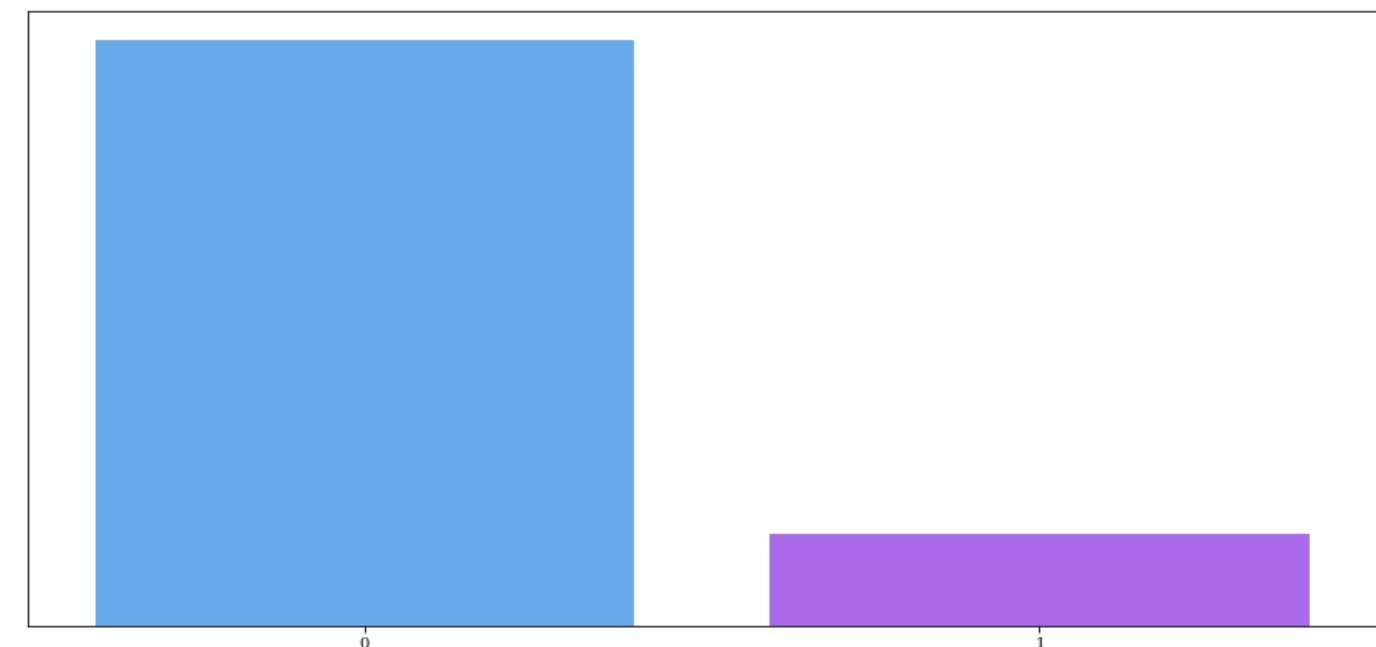
#Feature Transformation...
df['quality'].replace({'bad':0, 'good':1}, inplace=True)

plt.rcParams['font.family'] = ['serif']
background_color = "#ffffff"
fig = plt.figure(figsize=(15,7), facecolor=background_color)
fig.suptitle('After-Distribution Of The Quality', fontsize=30,
            fontweight='bold')

ax = sns.barplot(y = df['quality'].value_counts().values.tolist(),
                x = df['quality'].value_counts().index, palette="cool")
ax.get_yaxis().set_visible(False)
plt.show()
```

Feature Engineering 후 분포도 변화

After-Distribution Of The Quality



→ quality가 2~6.5 사이에 값이 있으면 bad(0)으로 분류,
6.5~8 사이에 값이 있으면 good(1)로 분류

#2.4 PCA를 이용하여 차원 축소

PCA란?

→ large 변수 세트의 대부분의 정보를 포함하는 smaller 변수 세트로 변환(large data set의 차원을 줄이는데 사용)

⇒ 피처가 많으므로 PCA를 사용하여 차원을 축소하자!

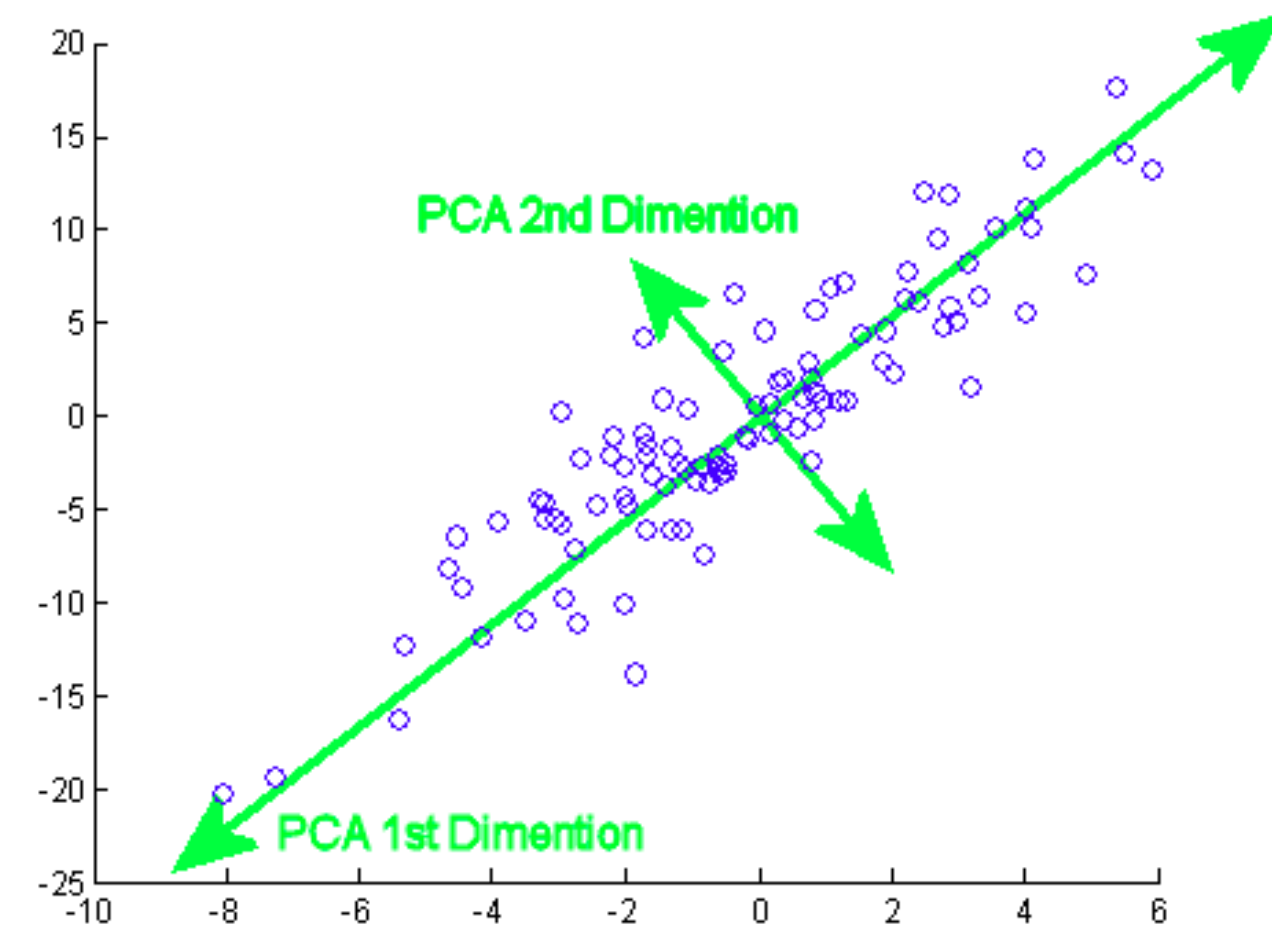
Step 1: Standardization.

Step 2: Covariance Matrix computation.

Step 3: Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components.

Step 4: Feature vector.

Step 5: Recast the data along the principal components axes.



2.4 PCA를 이용하여 차원 축소

1. PCA 후 explained variance 살펴보기 (n_components=None 으로 지정)

```
#PCA 적용하기
from sklearn.decomposition import PCA
pca = PCA(n_components=None)
X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)
explained_variance = pca.explained_variance_ratio_

print("Sorted List returned :")
print(sorted(explained_variance,reverse = True))
```

```
Sorted List returned :
[0.5806173780958788, 0.11278589033694, 0.10741172195074977, 0.06981449350
045157, 0.059741097869300736, 0.028786349665512363, 0.022723443272606354,
0.008874680016794197, 0.005225084817623676, 0.0040195205349985675, 3.3993
914415372063e-07]
```

⇒ 여기서 상위 5개 피처만 선택

2.4 PCA를 이용하여 차원 축소

2. 5개의 컴포넌트를 가진 PCA 변환 수행

```
#상위 5개 피처만 선택
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)
explained_variance = pca.explained_variance_ratio_

print(explained_variance)
```

```
[0.58061738 0.11278589 0.10741172 0.06981449 0.0597411 ]
```


#2.5 Model Creation

+ 불균형한 레이블을 가진 데이터셋이라 tree-based algorithms을 사용했다고 함(CART 알고리즘의 장점)

AdaBoost, GradientBoosting, RandomForest, DecisionTree 이용해 cv_score 구하기

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score

kfold = StratifiedKFold(n_splits=8, shuffle=True, random_state=42)

rs = 15
clrs = []

clrs.append(AdaBoostClassifier(random_state=rs))
clrs.append(GradientBoostingClassifier(random_state=rs))
clrs.append(RandomForestClassifier(random_state=rs))
clrs.append(DecisionTreeClassifier(random_state = rs))

cv_results = []
for clr in clrs :
    cv_results.append(cross_val_score(clr, X_train, y_train , scoring = 'accuracy'))

cv_means = []
cv_std = []
for cv_result in cv_results:
    cv_means.append(cv_result.mean())
    cv_std.append(cv_result.std())

cv_df = pd.DataFrame({"CrossVal_Score_Means":cv_means,"CrossValerrors": cv_std})
g = sns.barplot("CrossVal_Score_Means", "Algo", data = cv_df, orient = "h", **kwargs)
g.set_xlabel("Mean Accuracy", fontsize = 18)
g = g.set_title("Cross validation scores", fontsize = 24)
plt.figure(figsize = (15,7))
print(cv_df)
```

	CrossVal_Score_Means	CrossValerrors	Algo
0	0.838067	0.017989	RandomForestClassifier
1	0.849832	0.012887	AdaBoostClassifier
2	0.853746	0.016421	Gradient Boosting
3	0.822366	0.009710	DecisionTreeClassifier

⇒ cross_val_score가 높고 cross_val_error가 낮은 AdaBoost가 성능이 가장 좋아보임
(또는 Gradient Boosting)

+ AdaBoost의 confusion matrix

```
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import AdaBoostClassifier

ada_model = AdaBoostClassifier(random_state=44)
ada_model.fit(X_train, y_train)
y_preds = ada_model.predict(X_test)
confusion_matrix(y_test, y_preds)
```

```
array([[270, 23],
       [ 43,  4]], dtype=int64)
```

#2.6 Hyper parameter Tuning

(Adaboost의 base estimator로 DecisionTreeClassifier를 사용할 것이기 때문에 DecisionTreeClassifier 먼저 튜닝)

1. DecisionTreeClassifier의 하이퍼 파라미터 튜닝

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(random_state=44)
grid_params = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'min_samples_split' : range(2, 10, 1),
    'min_samples_leaf' : range(2, 10, 1)
}

grid_search = GridSearchCV(dtc, grid_params, cv=5, n_jobs=-10, verbose=True)
grid_search.fit(X_train, y_train)
grid_search.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf
=7,
                        random_state=44)
```

→ best estimator로 파라미터 변경

```
from sklearn.metrics import accuracy_score
dtc = grid_search.best_estimator_
y_pred = dtc.predict(X_test)
print(accuracy_score(y_test, y_pred))
```

0.8205882352941176

#2.6 Hyper parameter Tuning

2. AdaBoost의 하이퍼 파라미터 튜닝

```
ada_model = AdaBoostClassifier(base_estimator=dtc)
parameters = {
    'n_estimators' : [50, 70, 90, 120, 180, 200],
    'learning_rate' : [0.001, 0.01, 0.1, 1, 10],
    'algorithm' : ['SAMME', 'SAMME.R']
}

grid_search = GridSearchCV(ada_model, parameters, n_jobs = -1, cv = 10,
                           verbose = 1)
grid_search.fit(X_train, y_train)

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

```
{'algorithm': 'SAMME', 'learning_rate': 0.001, 'n_estimators': 180}
0.8665501844302078
```

#2.7 Final Model

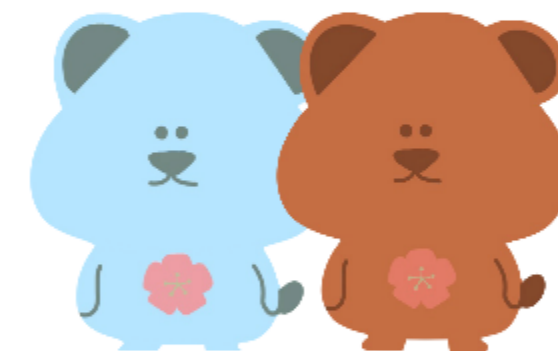
```
ada_model = AdaBoostClassifier(base_estimator=dtc,  
                               algorithm='SAMME', learning_rate=0.001,  
                               n_estimators=180)  
  
ada_model.fit(X_train, y_train)  
y_pred = ada_model.predict(X_test)  
confusion_matrix(y_test, y_pred)
```

```
array([[275, 18],  
       [ 44,  3]], dtype=int64)
```

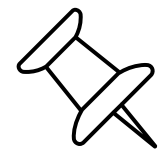
```
from sklearn.metrics import classification_report  
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.86	0.94	0.90	293
1	0.14	0.06	0.09	47
accuracy			0.82	340
macro avg	0.50	0.50	0.49	340
weighted avg	0.76	0.82	0.79	340

03. 이미지 차원 축소 MNIST



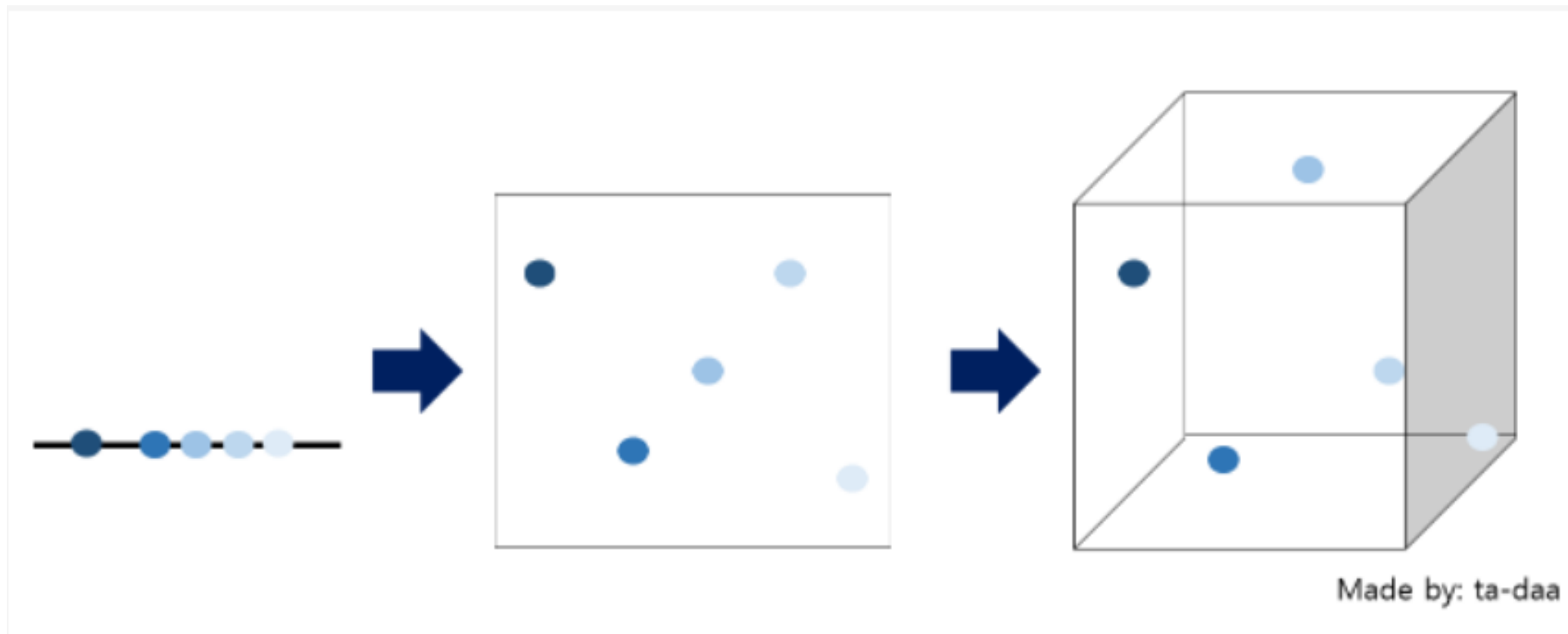
3.1 The Curse of Dimensionality (복습)



차원의 저주: 데이터 학습을 위해 차원이 증가, 학습데이터 수가 차원의 수보다 적어져 성능이 저하되는 현상, 차원이 증가할수록 개별 차원 내 학습할 데이터 수가 적어지는(sparse)발생

→차원이 증가(변수 증가)에 따라 모델의 성능 저하

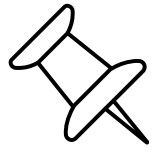
→(무조건 변수가 증가한다고 차원의 저주가 생기게 아니라 관측치 수보다 변수의 수가 많아지면 발생)



변수가 1개인 상태에서 차원만 증가한다. (차원이 늘어날수록 점들 사이에 공간이 많이 빈다.)

→ 빈 공간은 컴퓨터 상으로 0으로 채워졌다는 의미, 자연스레 모델 성능 저하

3.2 이미지 데이터 차원축소 mnist 데이터



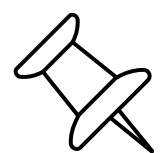
노트북 설명: 데이터를 feature의 수를 줄이면서 더 직관적인 해석이 가능하게 만드는 것이 목표
- PCA, Truncated SVD, NMF, LDA, t-SNE, UMAP 을 이용한다.

MNIST 데이터

- 60,000개의 트레이닝 셋, 10,000개의 테스트 셋
- 손으로 쓰여진 이미지들로 구성
- 숫자는 0에서 1까지의 값을 갖는 고정 크기 이미지 (28x28 픽셀)로 크기 표준화되고 중심에 배치



3.2 이미지 데이터 차원축소 mnist 데이터



테스트 데이터 셋 확인

아나콘다 프롬프트 창에서.. pip install --upgrade tensorflow 을 먼저 설치하기

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
from keras.datasets import mnist
(train_x, train_y), (test_x, test_y) = mnist.load_data()
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11493376/11490434 [=====] - 0s 0us/step

11501568/11490434 [=====] - 0s 0us/step

```
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(10):
    ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
    ax.imshow(train_x[idx], cmap='Blues_r')
    ax.set_title(str(train_y[idx]), fontsize=25)
```



```
img = train_x[0]

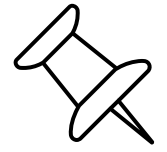
fig = plt.figure(figsize = (12,12))
ax = fig.add_subplot(111)
ax.imshow(img, cmap='Blues_r')
width, height = img.shape
thresh = img.max()/2.5
for x in range(width):
    for y in range(height):
        val = round(img[x][y],2) if img[x][y] !=0 else 0
        ax.annotate(str(val), xy=(y,x),
                    horizontalalignment='center',
                    verticalalignment='center',
                    color='white' if img[x][y]<thresh else 'black')
```

```
# Reshape to 2D data
train_x = train_x.reshape(train_x.shape[0], -1)
print(train_x.shape)

sample_size = 5000
# Use only the top 1000 data for training
train_x = pd.DataFrame(train_x[:sample_size, :])
train_y = train_y[:sample_size]
```

(60000, 784)

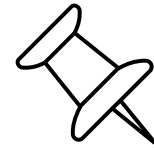
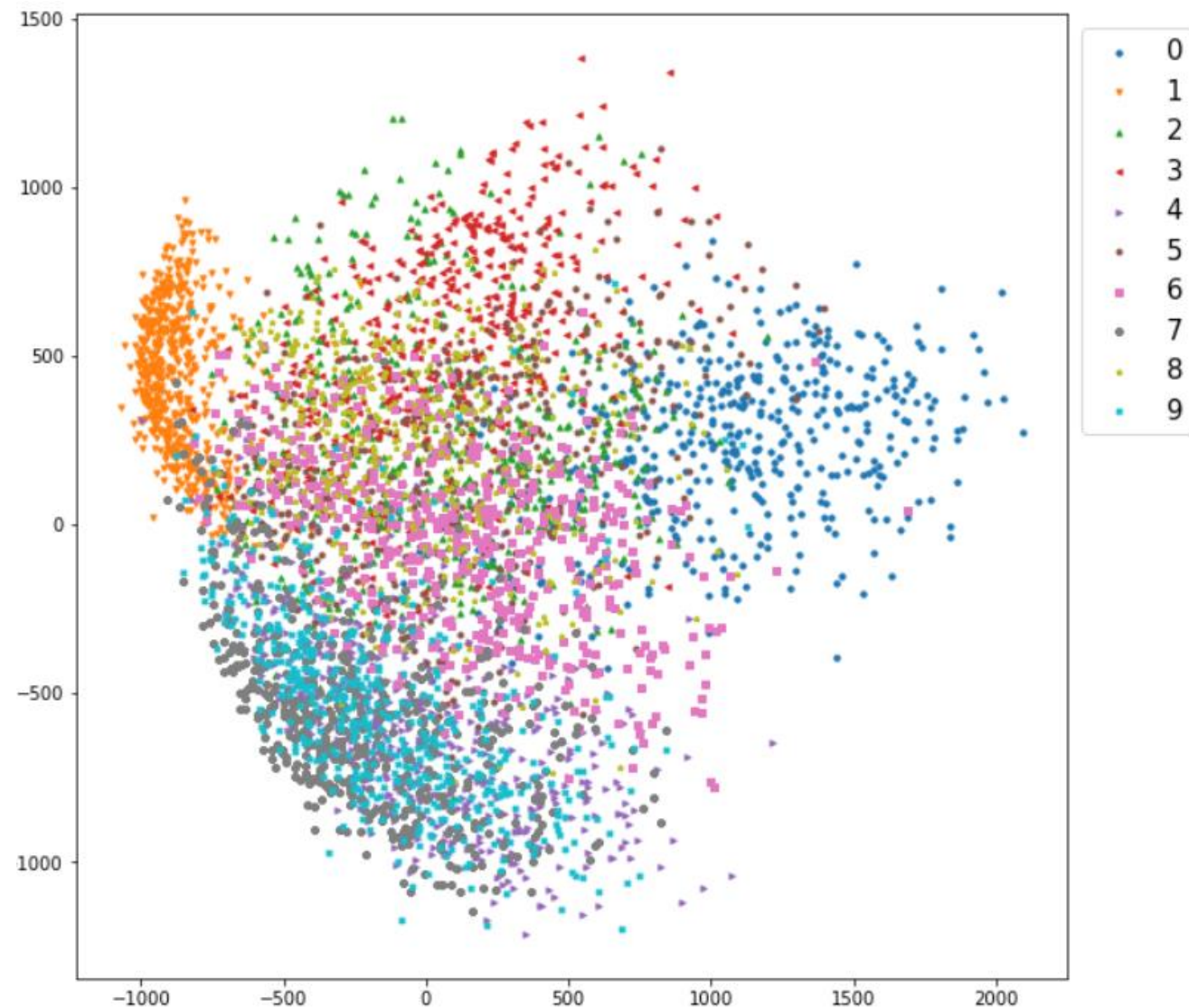
3.3 mnist 데이터 – PCA (복습)



PCA

```
from sklearn.decomposition import PCA

pca = PCA()
x_pca = pca.fit_transform(train_x)
markers=['o','v','^','<','>','8','s','P','*','X']
# plot in 2D by class
plt.figure(figsize=(10,10))
for i,marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_pca[mask, 0], x_pca[mask, 1], label=i, s=10, alpha=1,marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left',fontsize=15)
```

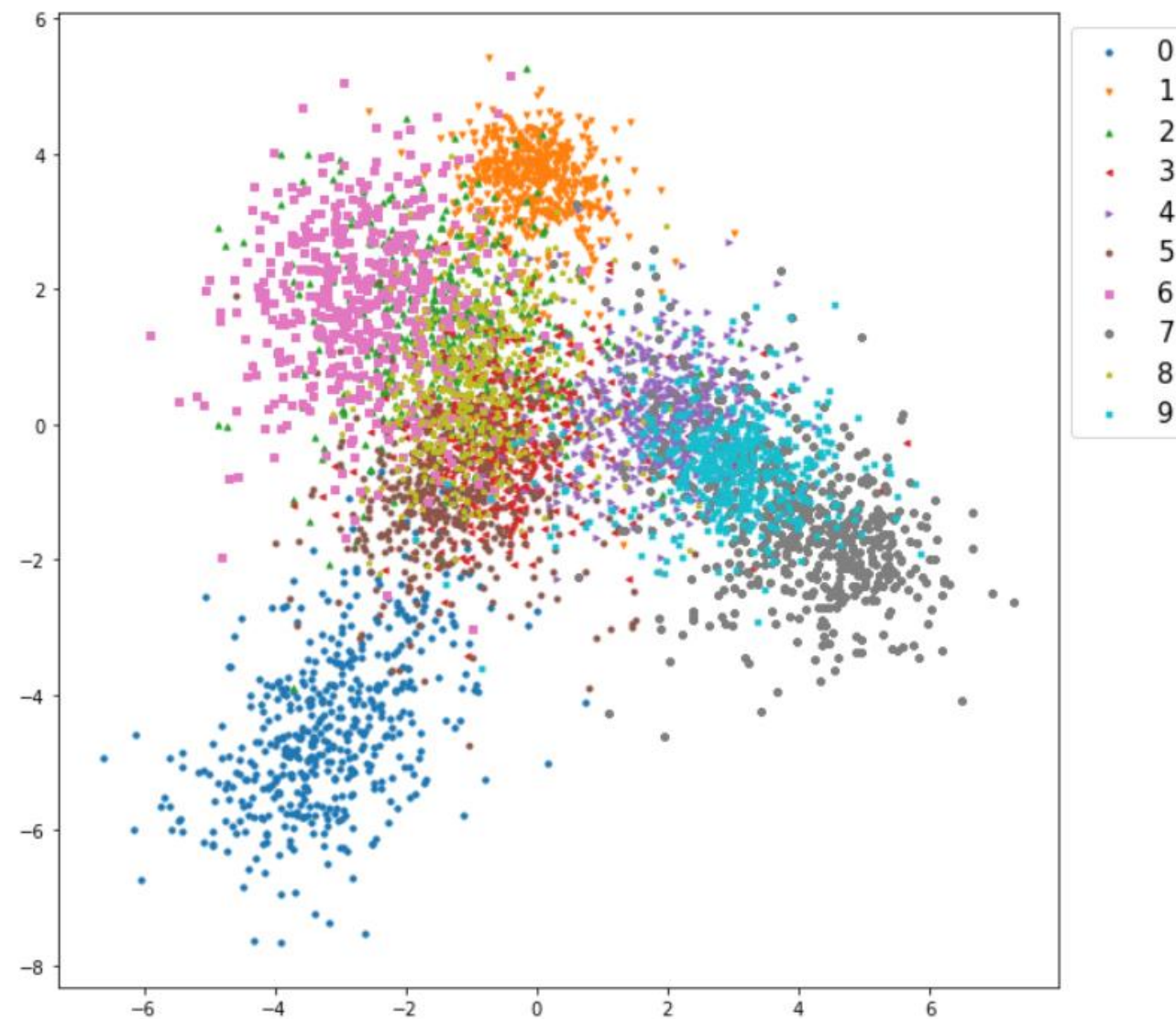


LDA

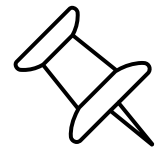
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=2)
x_lda = lda.fit_transform(train_x, train_y)

plt.figure(figsize=(10,10))
for i,marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_lda[mask, 0], x_lda[mask, 1], label=i, s=10, alpha=1,marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left',fontsize=15)
```



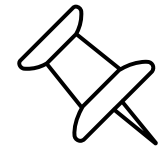
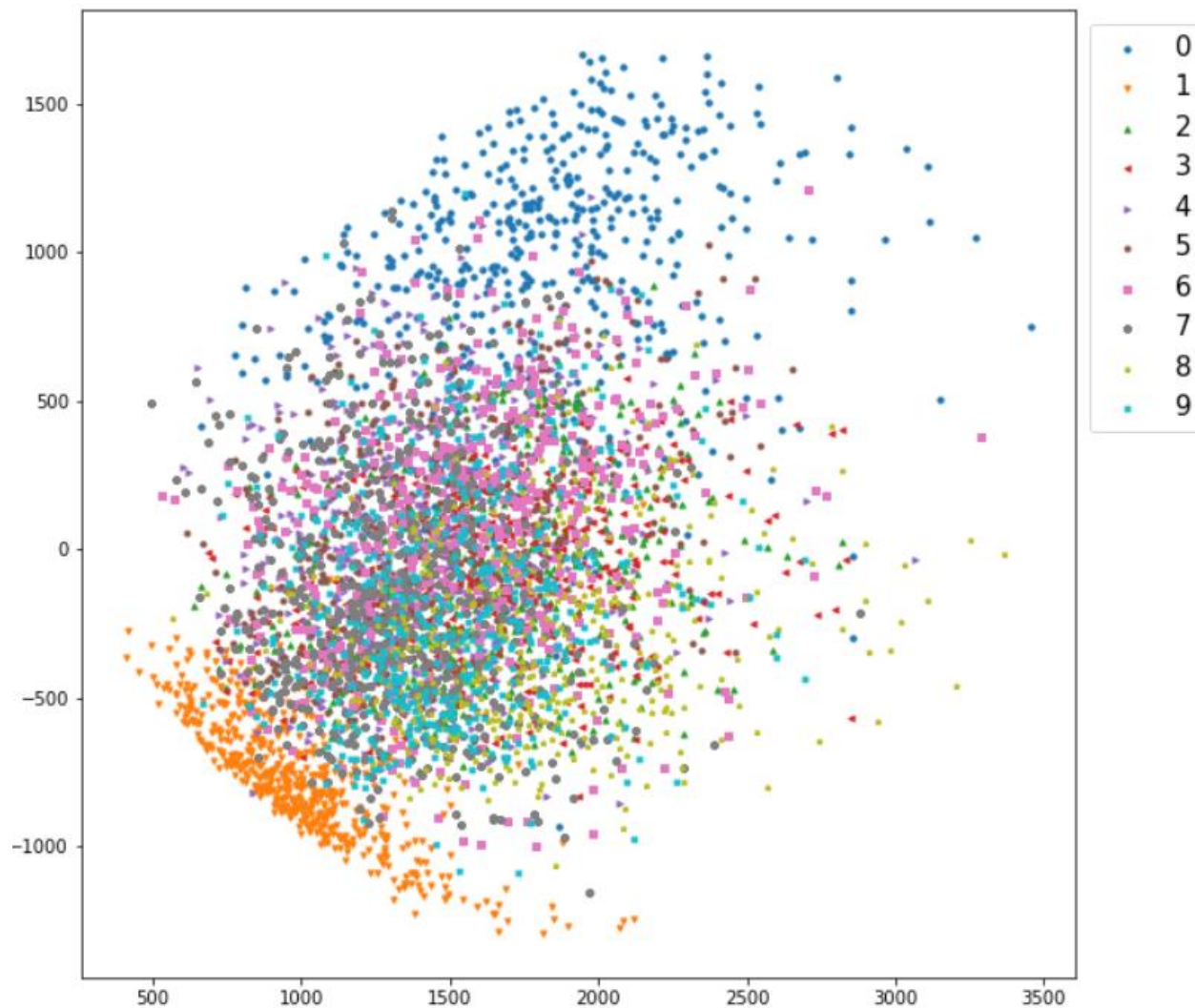
3.4 mnist 데이터 – Truncated SVD (복습)



Truncated SVD

```
from sklearn.decomposition import TruncatedSVD

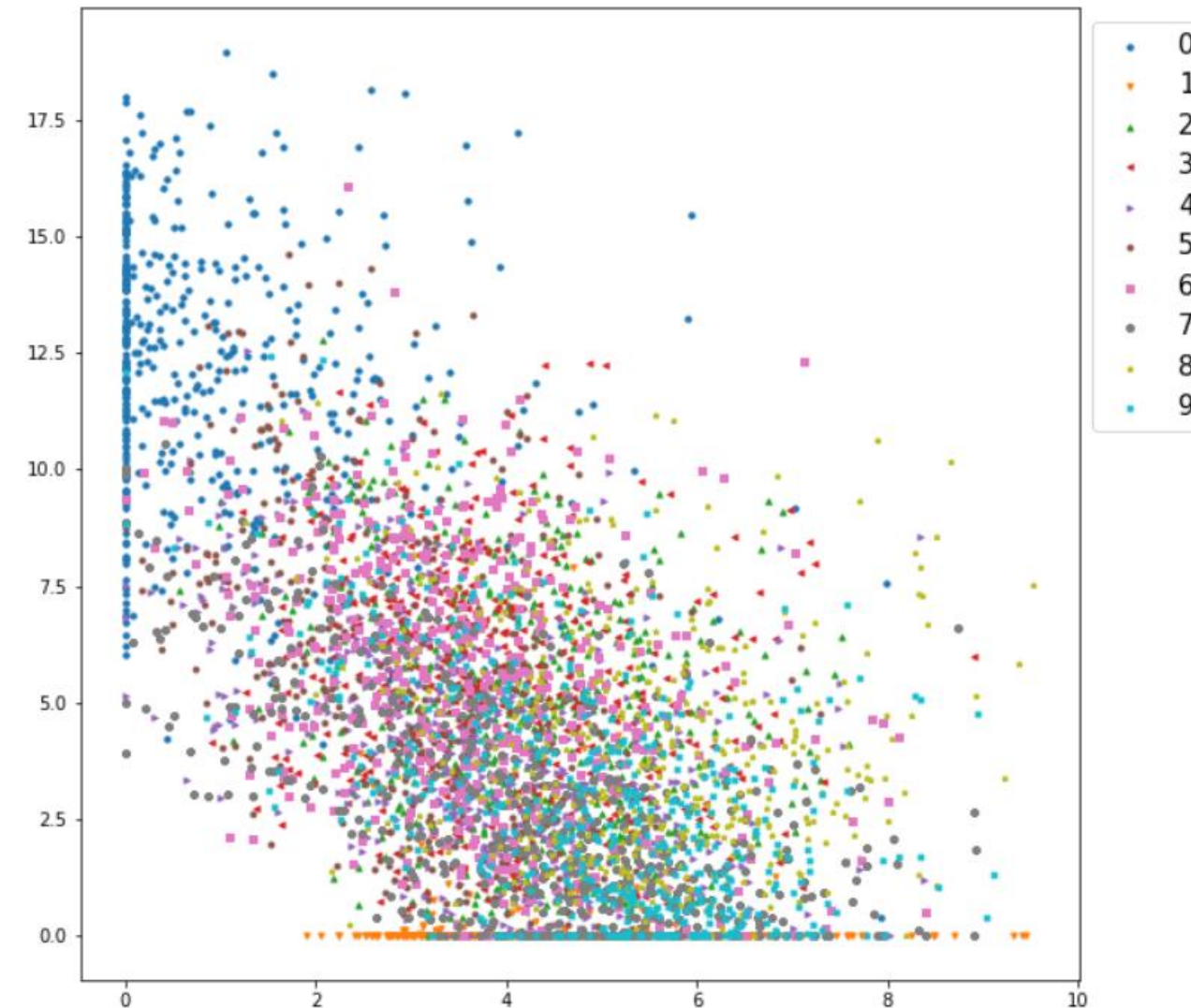
tsvd = TruncatedSVD()
x_tsvd = tsvd.fit_transform(train_x)
markers=['o','v','^','<','>','8','s','P','*','X']
# plot in 2D by class
plt.figure(figsize=(10,10))
for i,marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_tsvd[mask, 0], x_tsvd[mask, 1], label=i, s=10, alpha=1, marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left', fontsize=15)
```



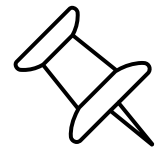
NMF

```
from sklearn.decomposition import NMF

nmf = NMF(n_components=2, init='random', random_state=0)
x_nmf = nmf.fit_transform(train_x)
markers=['o','v','^','<','>','8','s','P','*','X']
# plot in 2D by class
plt.figure(figsize=(10,10))
for i,marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_nmf[mask, 0], x_nmf[mask, 1], label=i, s=10, alpha=1, marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left', fontsize=15)
```

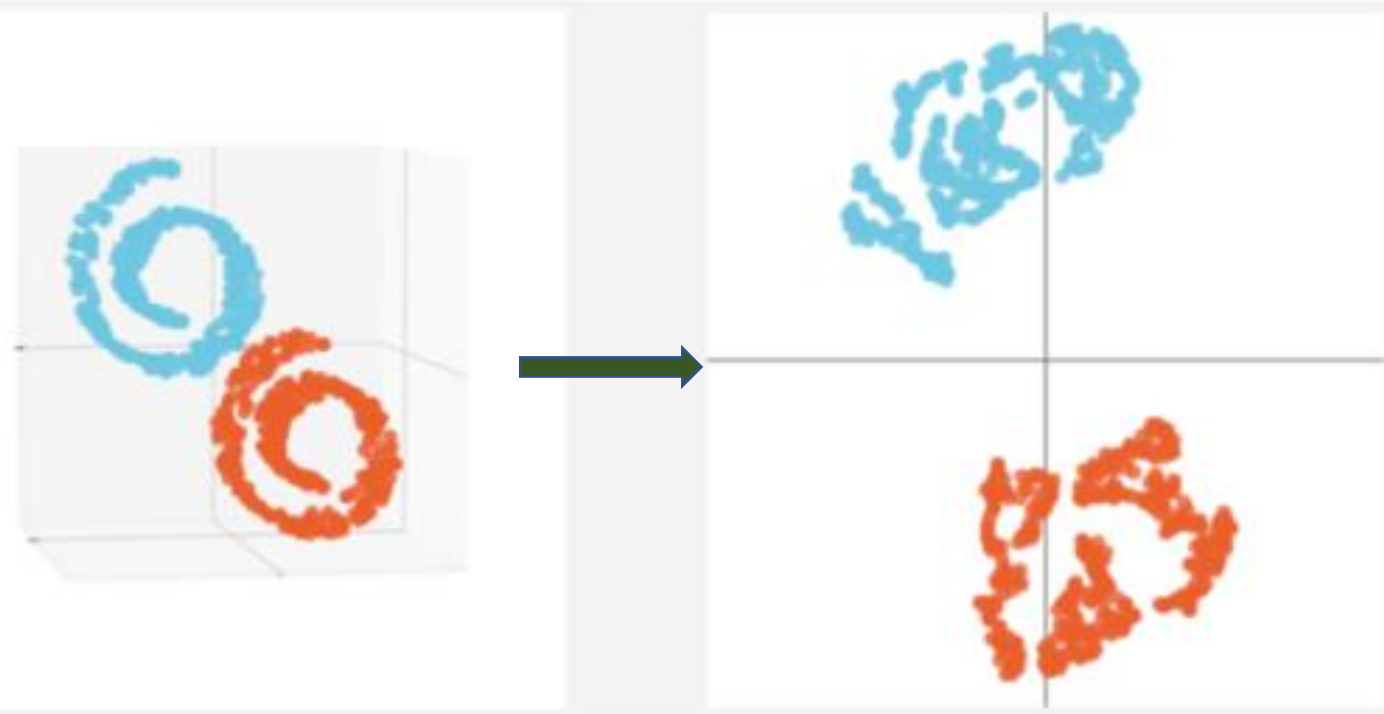


3.7 mnist 데이터 – t-SNE



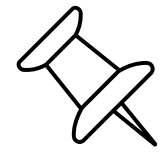
T-SNE

- 낮은 차원 공간의 시각화에 주로 사용
- 차원 축소할 때는 비슷한 구조끼리 데이터를 정리한 상태이므로 데이터 구조를 이해하는 데 도움을 줌
- t-SNE는 매니폴드 학습의 하나로 복잡한 데이터의 시각화가 목적
- 높은 차원의 데이터를 2차원 또는 3차원으로 축소시켜 시각화 함



- t-SNE를 사용하면 높은 차원 공간에서 비슷한 데이터 구조는 낮은 차원 공간에서 가깝게 대응하며, 비슷하지 않은 데이터 구조는 멀리 떨어져 대응됨
- 왼쪽 사진: 3차원을 2차원으로 나타낸 것(비슷한 데이터 구조는 2차원에서 더 가까이, 비슷하지 않은 데이터는 더 멀리 위치한다.)

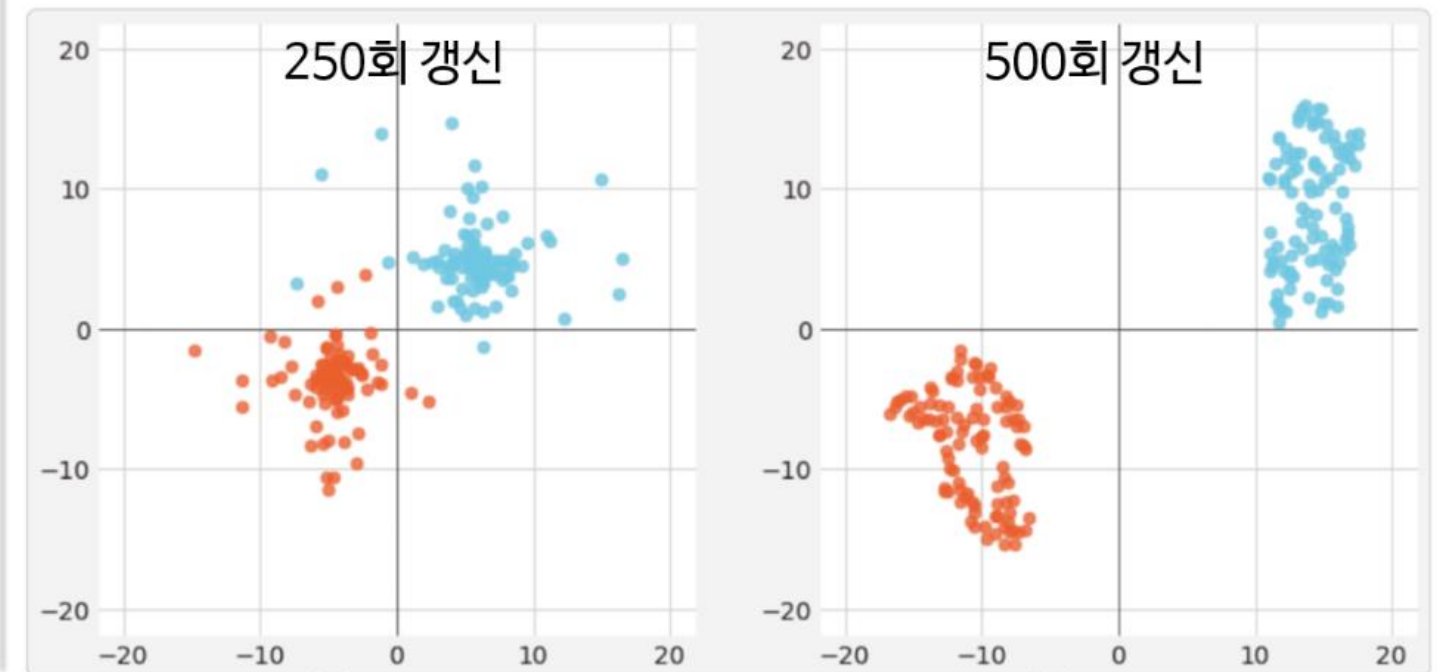
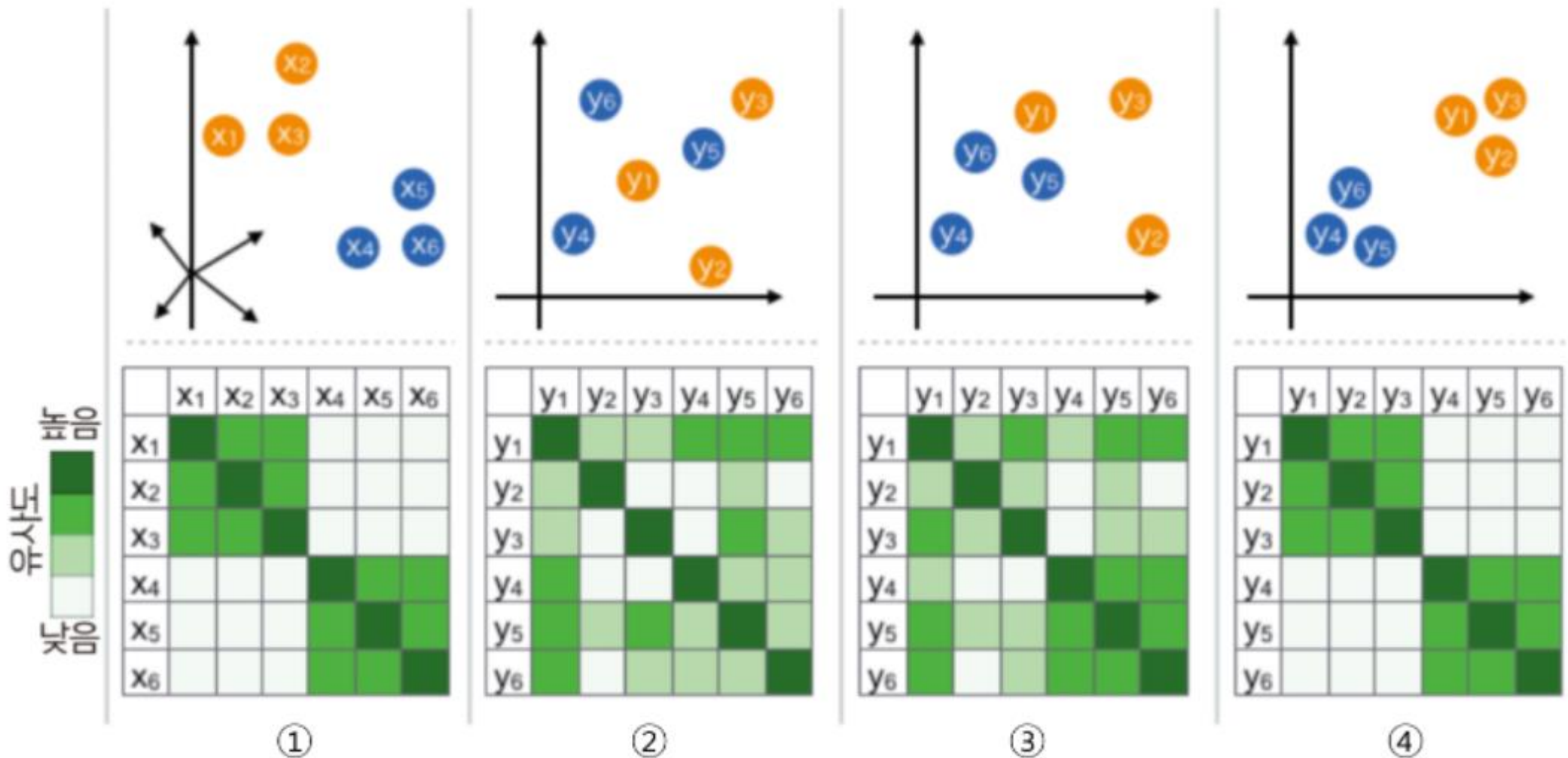
3.7 mnist 데이터 – t-SNE



T-SNE 의 작동 방식 ①

X는 기존 데이터에 해당, y는 t-SNE를 통해 저차원으로 매핑된 데이터

- 1) 모든 x에 대한 가우시안 분포
- 2) X와 같은 개수의 점만큼 y를 무작위하게 저차원에 배열, 모든 y의 쌍에 대한 가우시안 분포를 표현
- 3) 1), 2)에서 정의한 유사도 분포가 같아지도록 y값을 갱신
- 4) 수렴할 때까지 3)을 반복



3.7 mnist 데이터 – t-SNE

T-SNE 의 작동 방식 ②

t-SNE에서 낮은 차원에 임베딩을 진행할 때: t-분포 이용

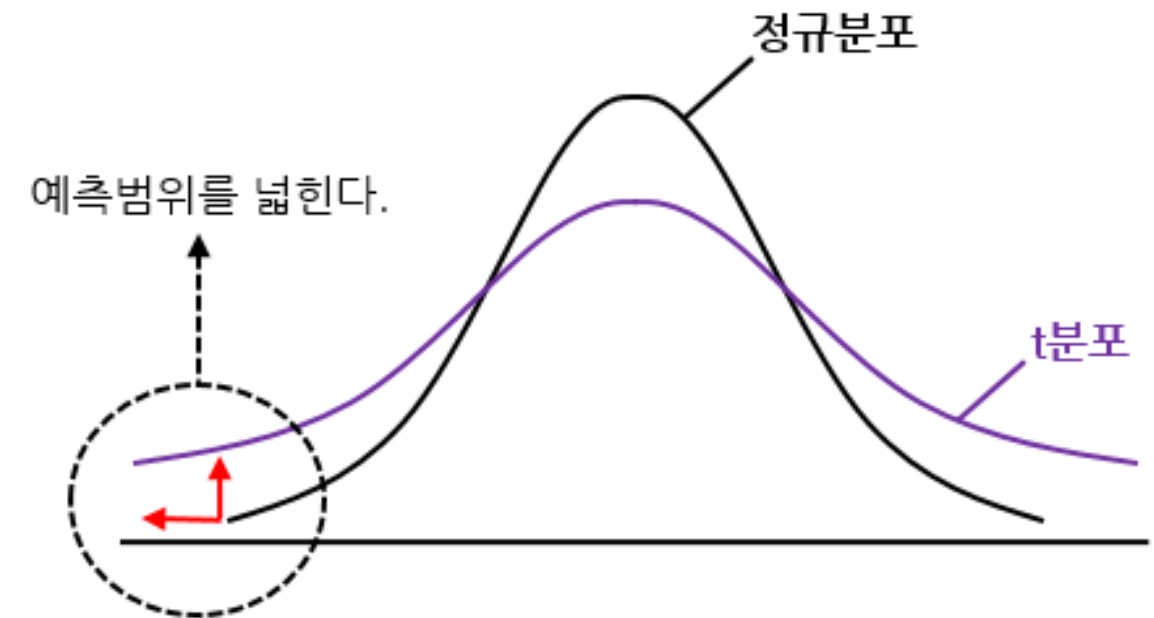
→정규분포보다 꼬리 두꺼운 분포

이유: t-SNE는 정규분포를 전제하지만 정규분포는 꼬리가 두껍지 않아

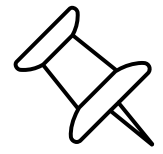
적당히 멀리 떨어진 개체와 엄청 멀리 떨어진 개체가 선택될 확률의 차이가 크지 않다.

→구분을 더 잘하기 위해 꼬리가 두꺼운 t-분포를 이용

→자연스럽게 표본의 수가 많아질 수록 정규분포에 수렴하게 된다. (by 중심 극한 정리)



3.7 mnist 데이터 – t-SNE



T-SNE 파라미터

```
from sklearn.manifold import TSNE

tsne= TSNE(n_components=2,verbose=1,learning_rate=1000,perplexity=100,n_iter=1000)
tsne_result = tsne.fit_transform(train_feature_reduced)
```

n_components: 몇 차원으로 줄일 것인지

verbose: TSNE가 진행중인 장면이 나오게 할 것인지 결정하는 파라미터. (0으로 두면 안 나타나고 1로 두면 나옴)

learning_rate: (Default = 200 이지만 주로 10~1000까지 사용됨) 높으면 빠르지만 최솟값을 찾기가 힘들고 낮으면 너무 느리다.

perplexity: 복잡도에 관련된 파라미터로 이웃 간에 거리에 대한 파라미터
너무 높이면 거의 한점 처럼보이고 너무 낮으면 cluster를 잘 시키기 힘들다.
Default = 30으로 5~50를 쓴다.

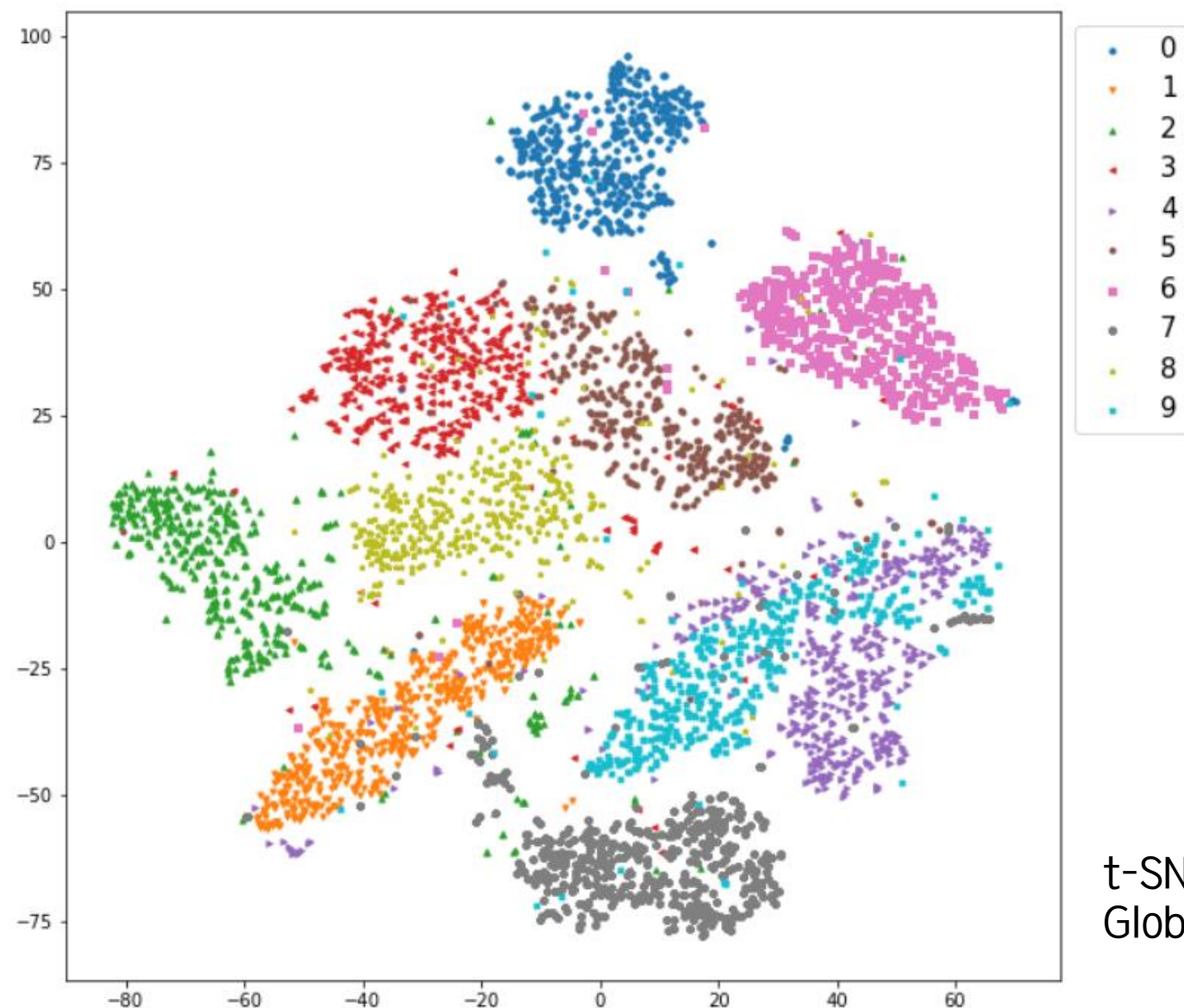
n_iter: 몇 번 돌릴 것인지에 대한 파라미터. Default = 10000이다.

3.7 mnist 데이터 – t-SNE

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2)  # 기본값: 2, 차원 축소 시에 임베딩되는 차원
x_tsne = tsne.fit_transform(train_x)

plt.figure(figsize=(10,10))
for i, marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_tsne[mask, 0], x_tsne[mask, 1], label=i, s=10, alpha=1, marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left', fontsize=15)
```



t-SNE사용결과, 변수 간 clustering된 모습을 확인할 수 있으나
Global structure는 잘 보이지 않는다.

3.8 mnist 데이터 – UMAP

UMAP 의 작동 방식

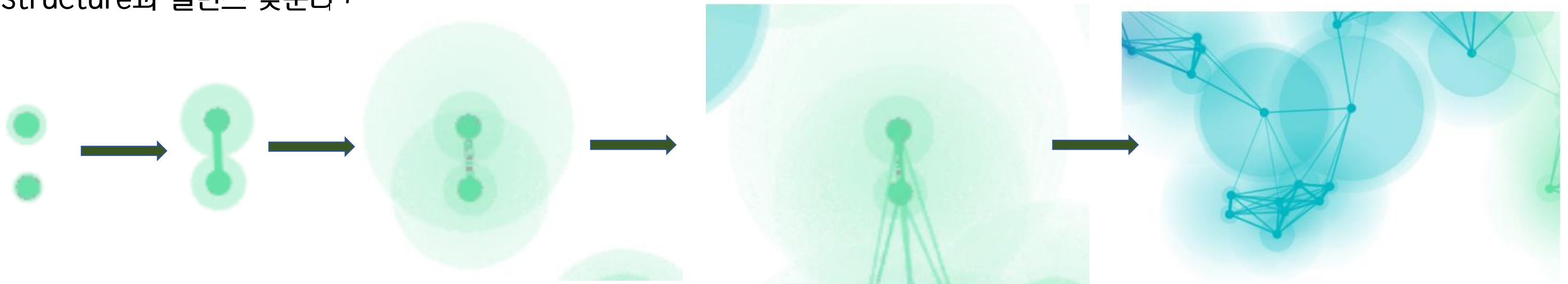
데이터의 고차원 그래프 표현을 만든 후 → 저차원의 것을 고차원 그래프와 최대한 유사하게 최적화

1) 고차원 그래프 생성: fuzzy simplicial complex(경계가 모호하고 단순한 복합체)를 구축
Edge weight(두 포인트가 연결 된 정도 -likelihood)를 이용해서 나타낸 weighted graph.

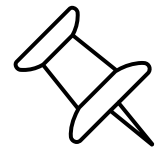
2) 각 point를 중심으로 원을 그리고 그 원끼리 포개지는 점들에 대해 두 점이 연결되어 있다고 생각.

- 원의 반지름의 크기가 중요(너무 작으면 고립된 클러스터 생성하도록 유도, 너무 크면 모두 다 연결)
(각 포인트마다 규칙에 맞게 반지름을 구해 적용, 모든 포인트에 똑같이 적용이 아님, locally)
- UMAP은 원의 반지름이 커짐에 따라 그래프를 fuzzy하게 만든다. (점 사이의 연결정도를 줄인다.)

3) 각 포인트들은 최소한 가장 가까운 점들과는 연결되어야 한다고 규정(local structure는 보존, global structure과 밸런스 맞춘다)



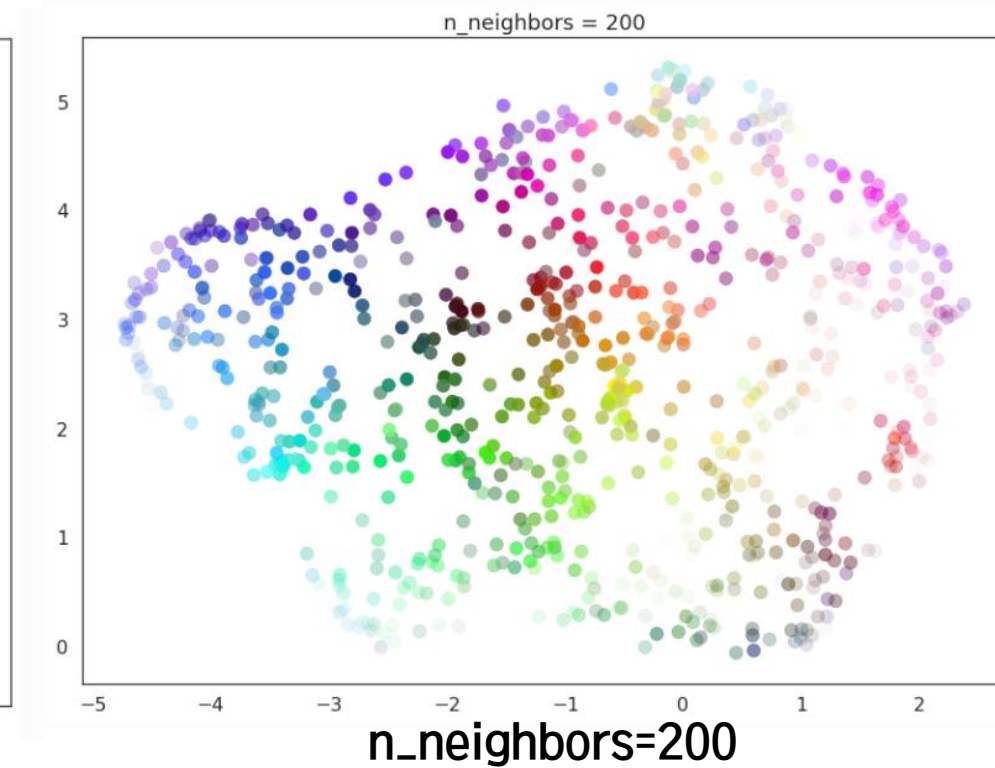
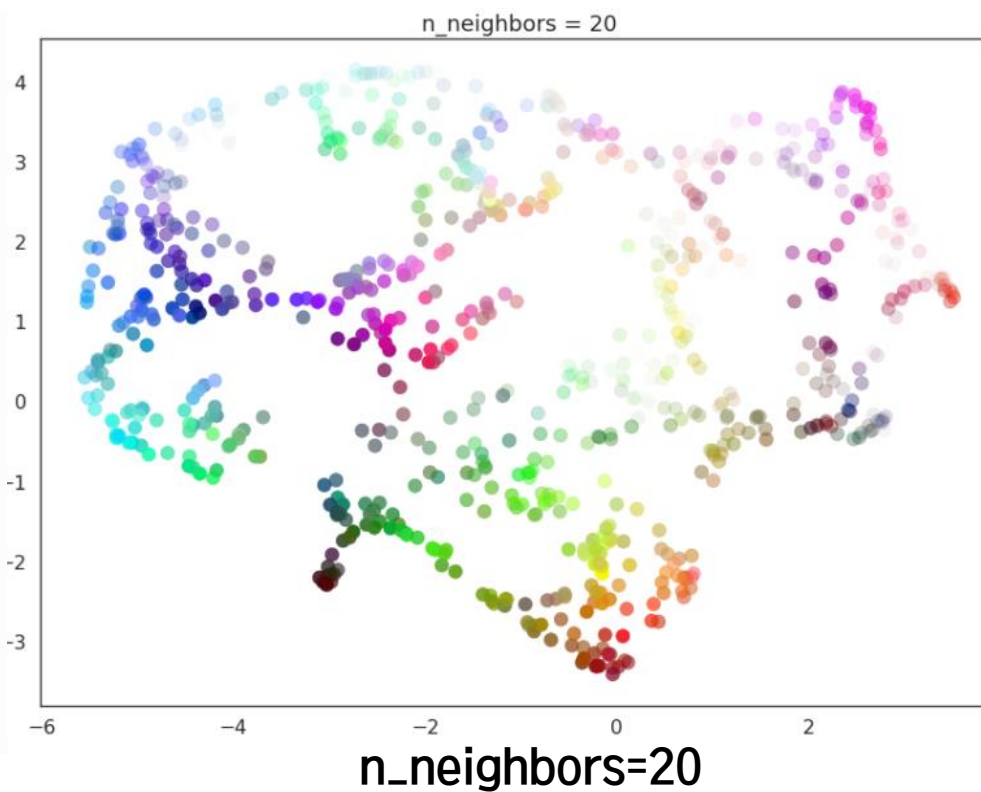
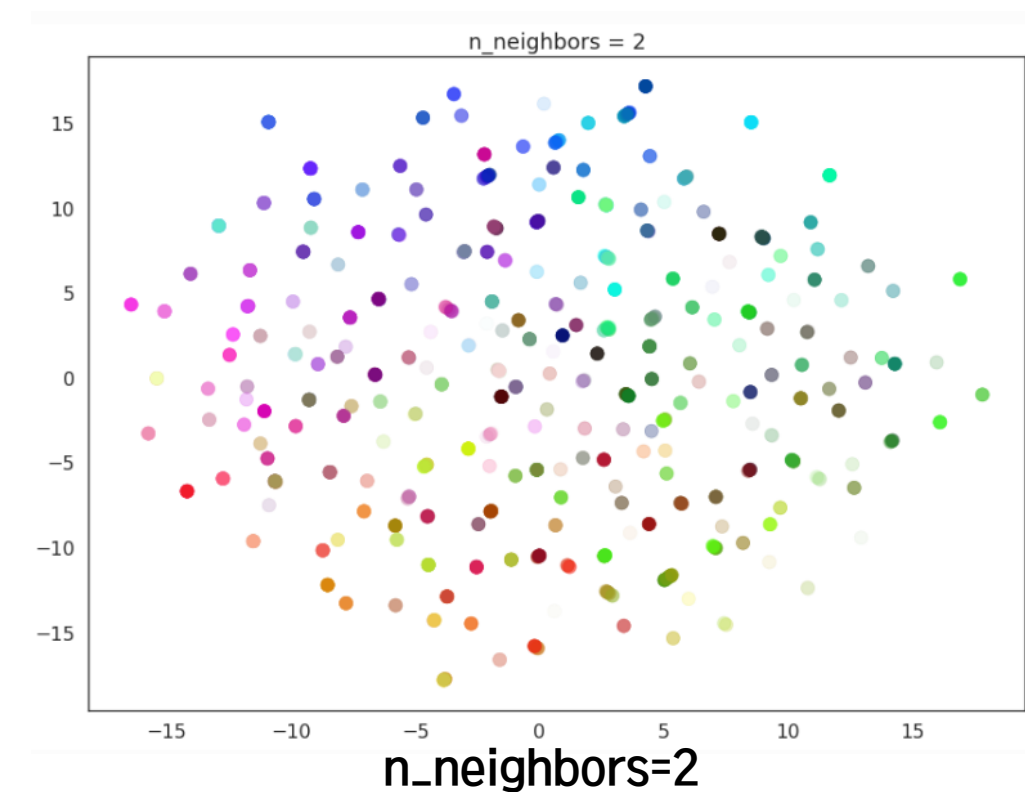
3.8 mnist 데이터 – UMAP



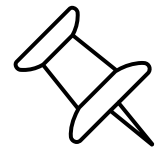
UMAP 파라미터

```
for n in (2, 5, 10, 20, 50, 100, 200):  
    draw_umap(n_neighbors=n, title='n_neighbors = {}'.format(n))
```

n_neighbors: 초기 고차원 그래프 생성 시에 사용되는 nearest neighbor의 숫자
이 값이 작으면: local structure(inner organization)에 집중
이 값이 크면 global structure(outer shape)에 집중



3.8 mnist 데이터 – UMAP



UMAP 파라미터

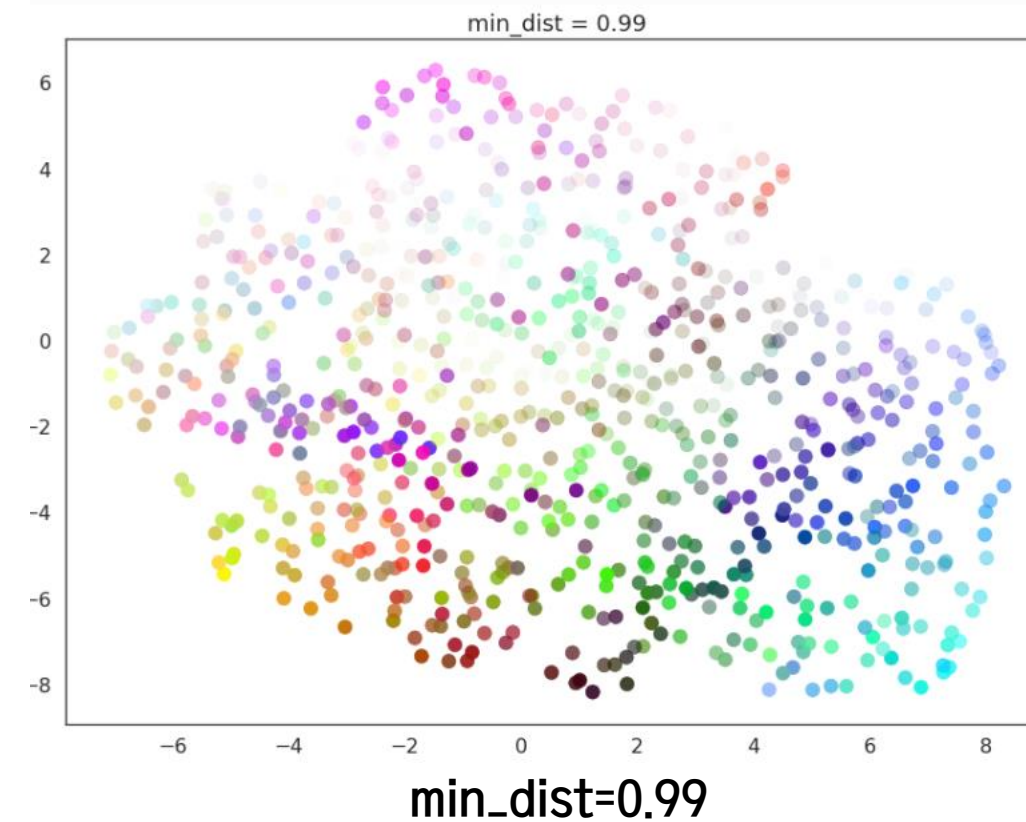
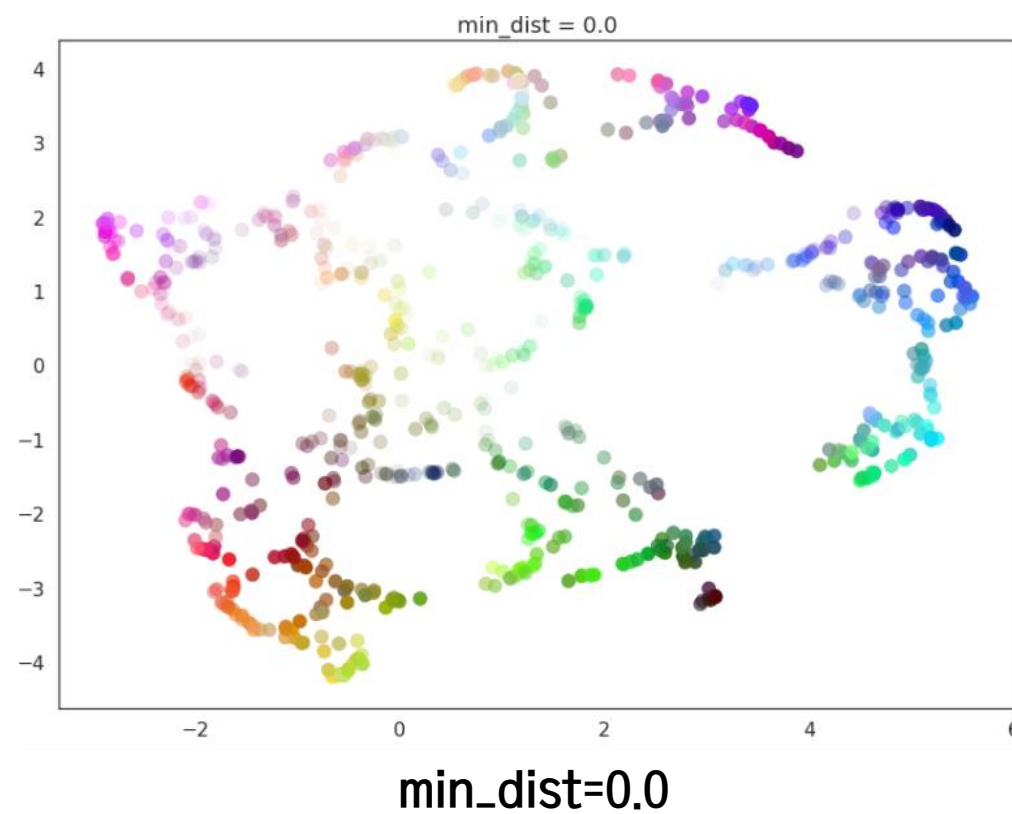
```
for d in (0.0, 0.1, 0.25, 0.5, 0.8, 0.99):  
    draw_umap(min_dist=d, title='min_dist = {}'.format(d))
```

min_dist: 저차원 공간에서 포인트 간의 최소 거리

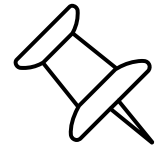
- UMAP에서 포인트를 얼마나 촘촘히 묶을지를 결정

이 값이 작으면: 포인트들이 촘촘히 무리지어 진다.

이 값이 크면: 포인트들이 상대적으로 느슨하게 퍼진다.



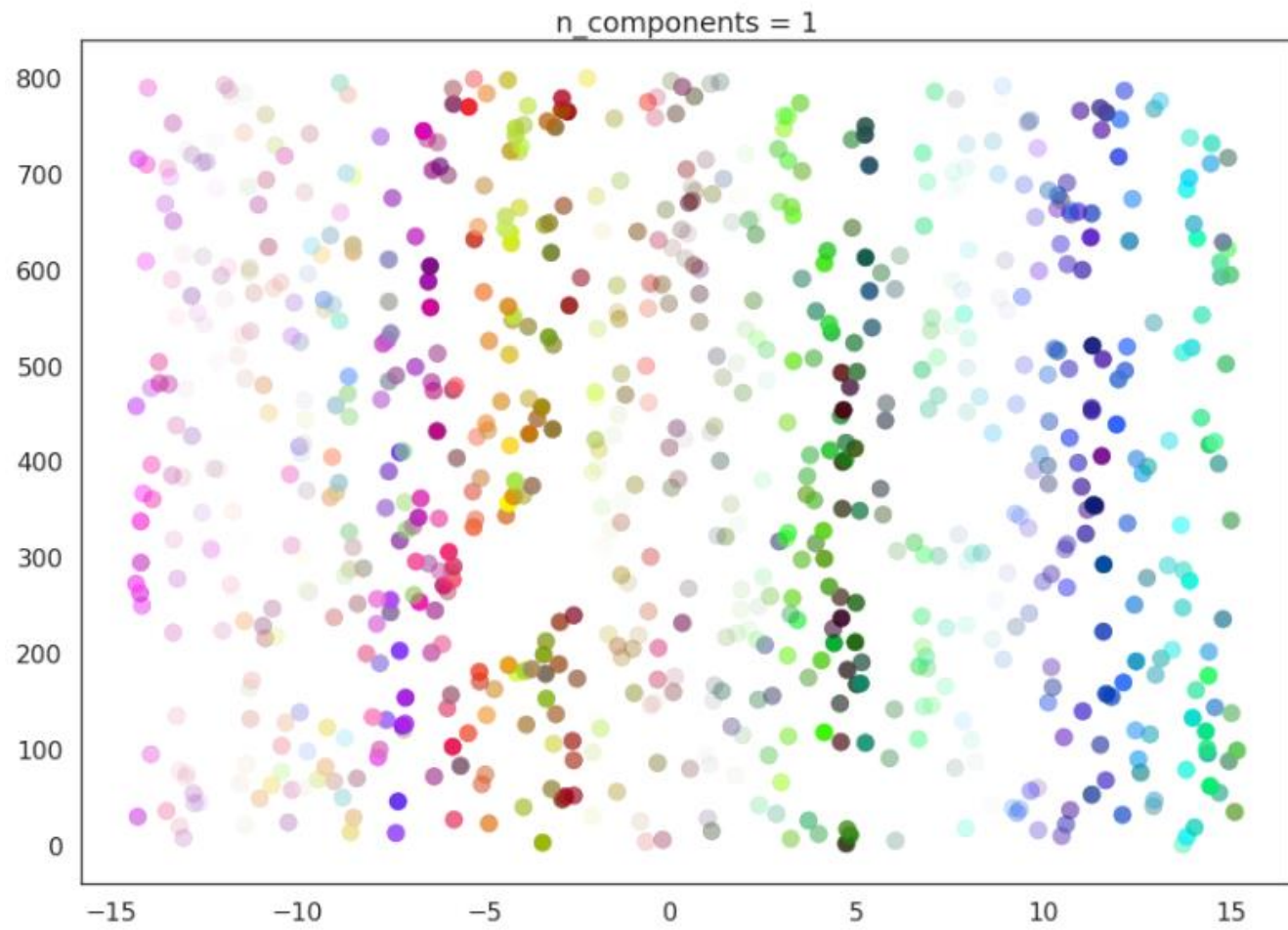
3.8 mnist 데이터 – UMAP



UMAP 파라미터

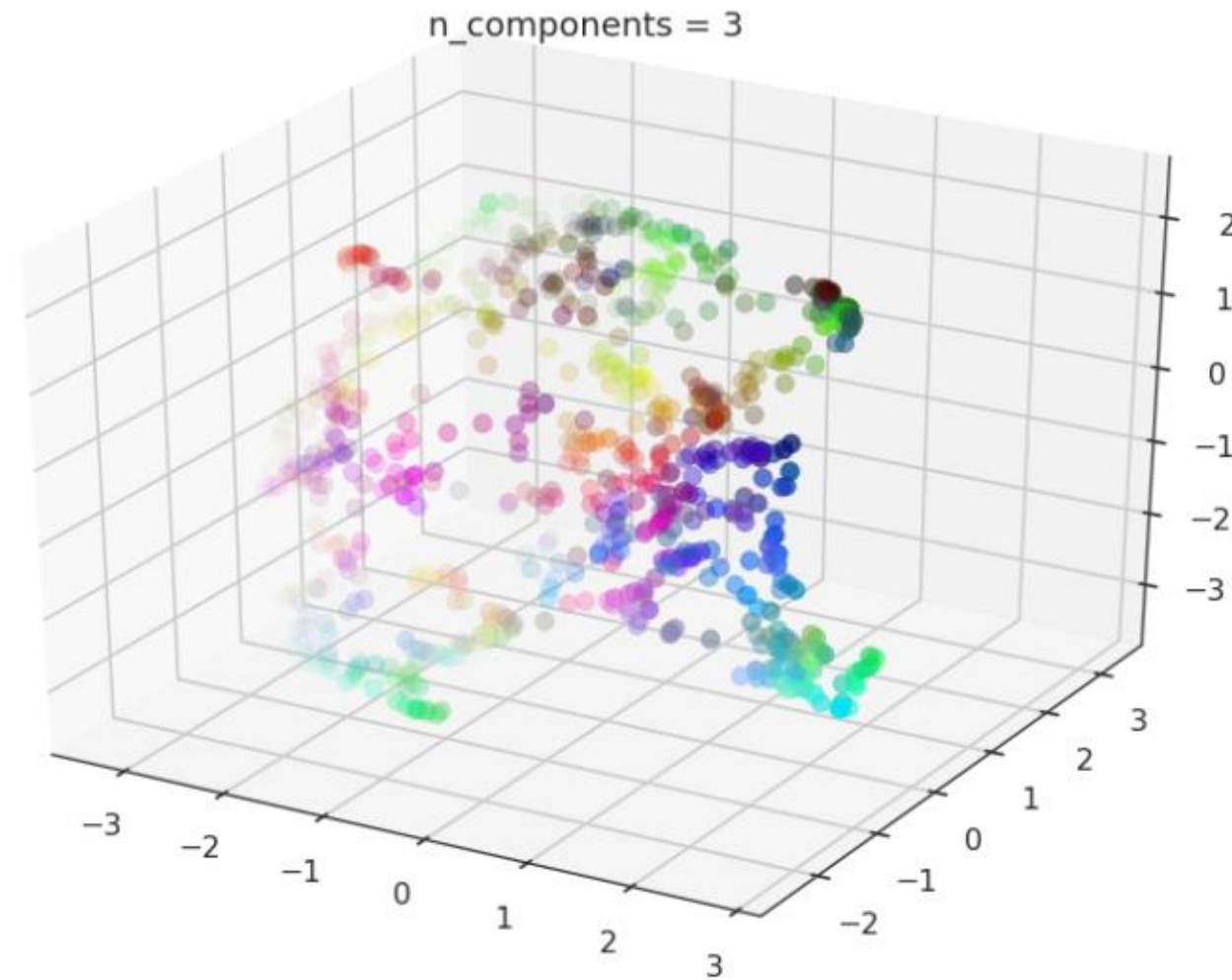
n_components: 몇 차원으로 줄일 것인지

```
draw_umap(n_components=1, title='n_components = 1')
```



```
draw_umap(n_components=3, title='n_components = 3')
```

```
/opt/anaconda3/envs/umap_dev/lib/python3.6/site-packages/sklearn/metrics/pairwise.py:257: RuntimeWarning:
return distances if squared else np.sqrt(distances, out=distances)
```



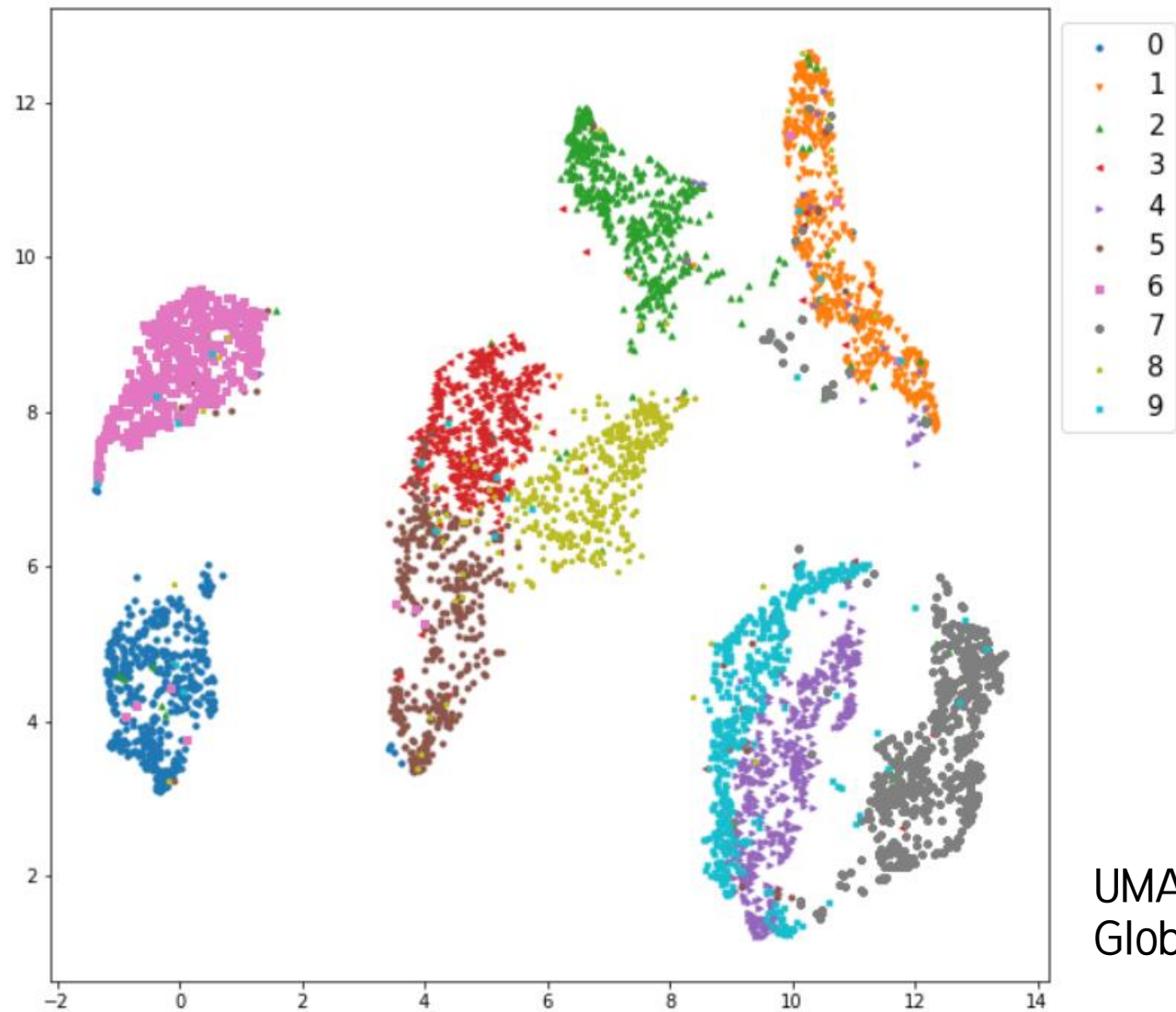
3.8 mnist 데이터 – UMAP

pip3 install umap-learn 를 먼저 아나콘다 프롬프트에 설치

```
import umap

um = umap.UMAP()
x_umap = um.fit_transform(train_x)

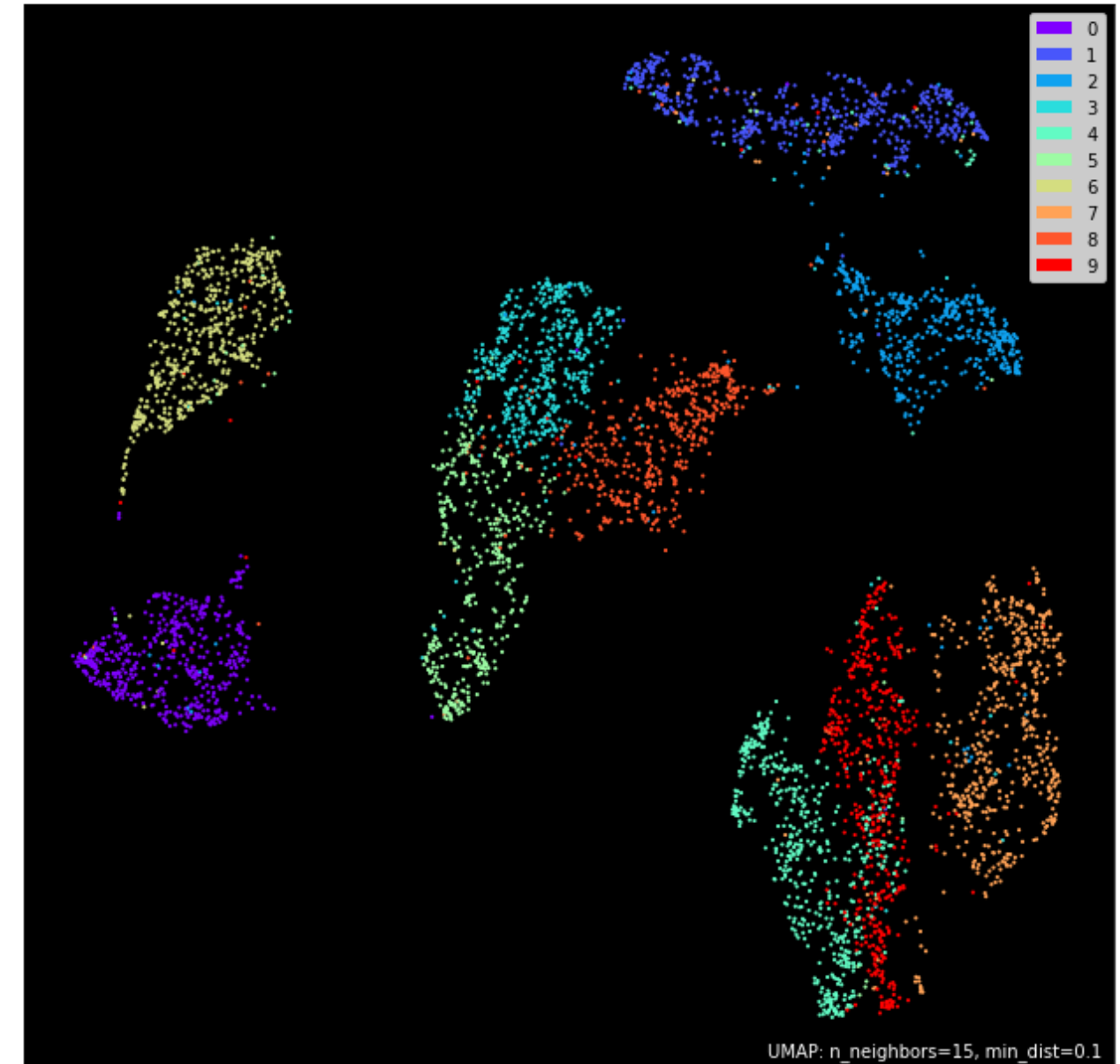
plt.figure(figsize=(10,10))
for i,marker in enumerate(markers):
    mask = train_y == i
    plt.scatter(x_umap[mask, 0], x_umap[mask, 1], label=i, s=10, alpha=1,marker=marker)
plt.legend(bbox_to_anchor=(1.00, 1), loc='upper left',fontSize=15)
```



UMAP 사용결과, 변수 간 clustering된 모습을 확인할 수 있으며
Global structure도 잘 나타난다.

```
umap.plot.points(mapper, labels=train_y, theme='fire')
```

<AxesSubplot:>

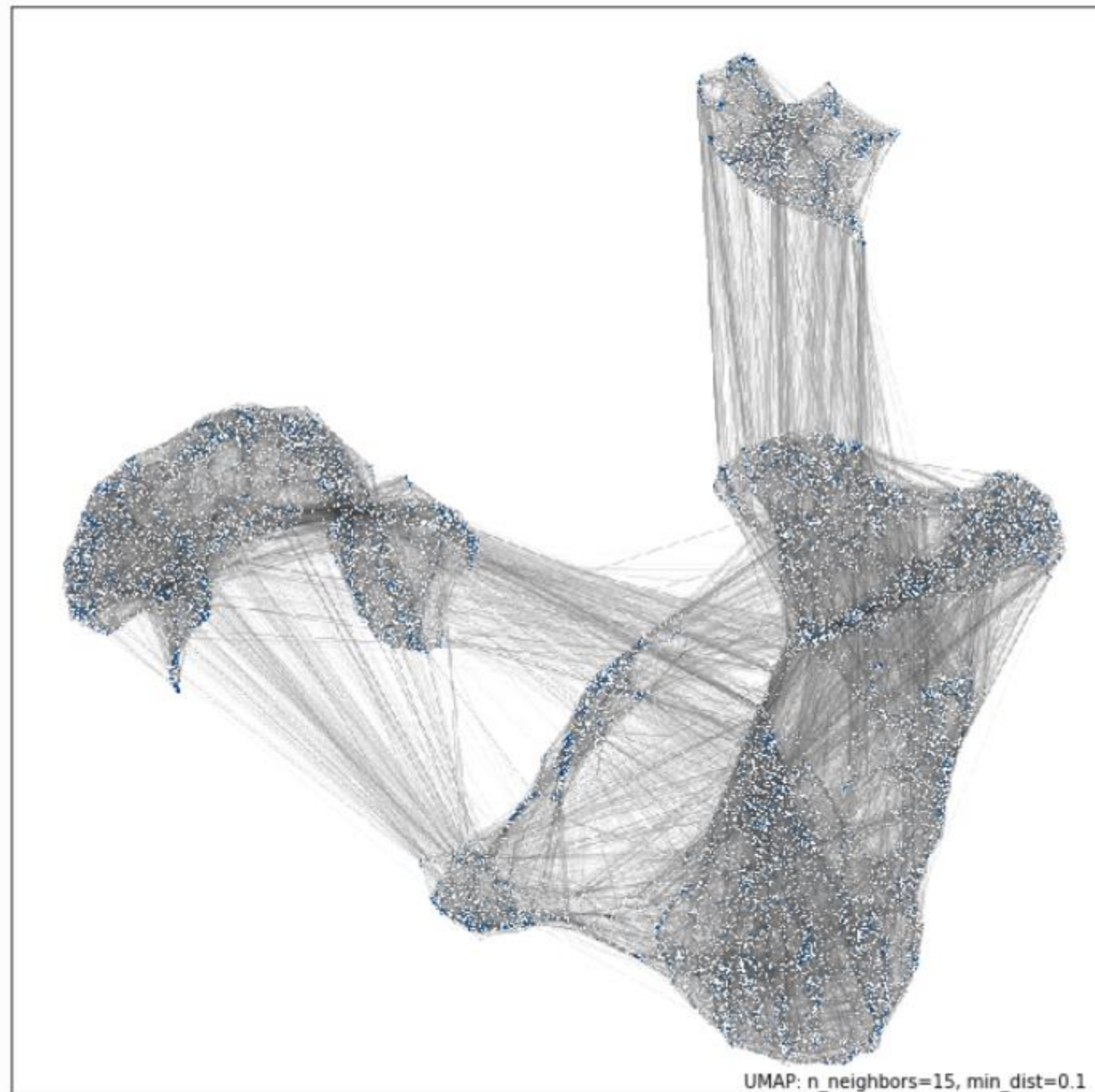


3.8 mnist 데이터 – UMAP

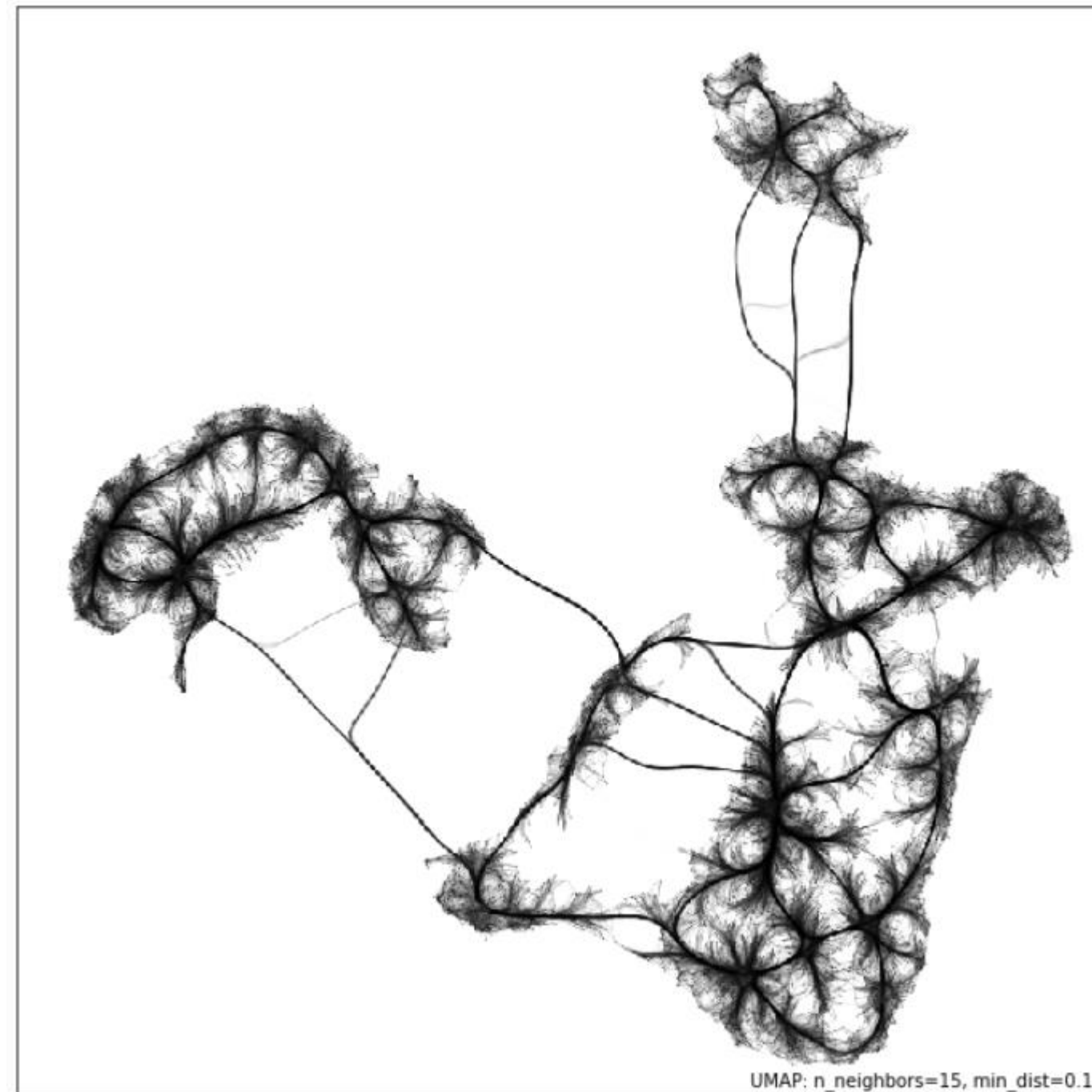
Connectivity확인

pip install umap-learn[plot]

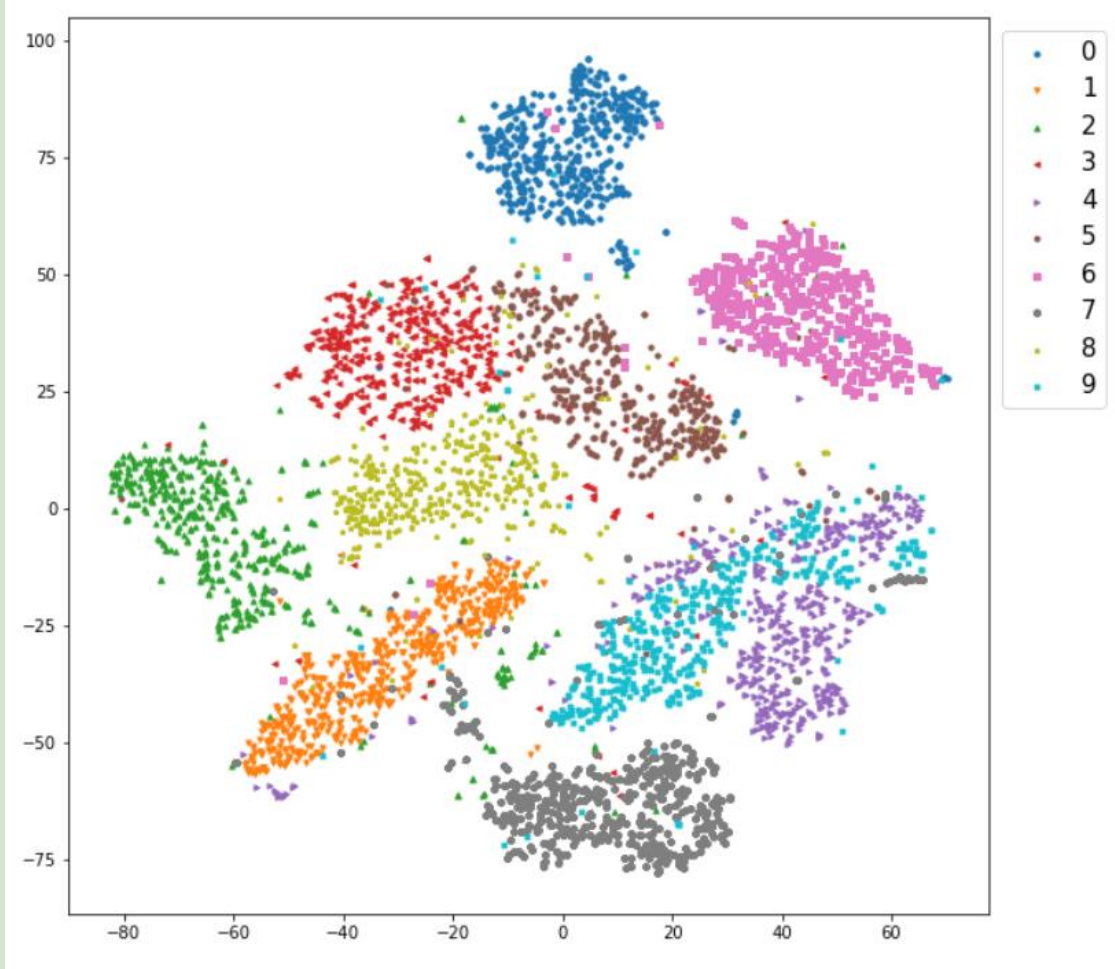
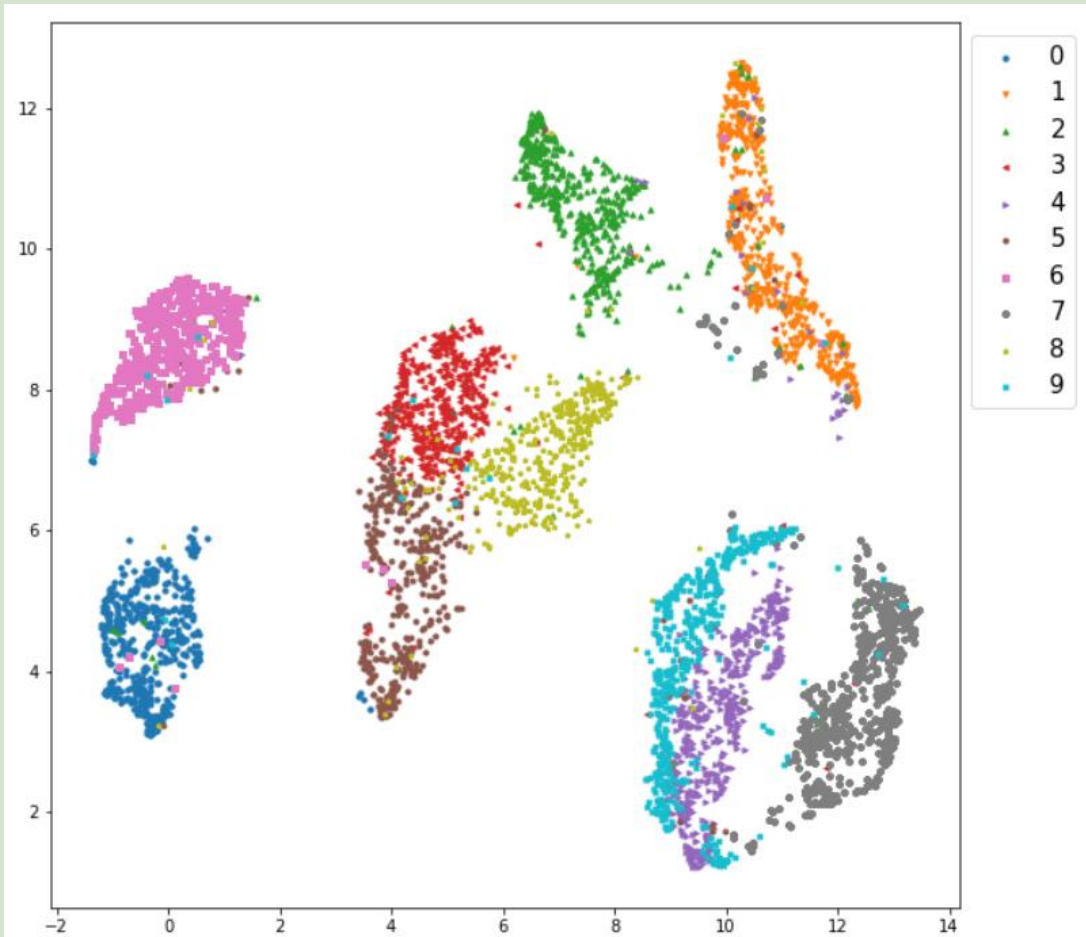
```
umap.plot.connectivity(mapper, show_points=True)
```



```
umap.plot.connectivity(mapper, edge_bundling='hammer')
```



3.9 t-SNE vs UMAP

t-SNE	UMAP
비지도 학습 / manifold learning	비지도 학습 / manifold learning
Perplexity가 낮을 경우 global structure보존이 거의 안됨	Global structure(outer shape)을 잘 보존
클러스터 간 관계가 잘 안보임	클러스터 간 관계가 t-SNE보다 의미 있게 연결됨
Sparse한 matrices에 바로 적용 불가능 PCA, SVD같은 차원 축소 선행 필요 작동 속도 느림	Sparse한 matrices에 바로 적용 가능 작동 속도 빠름
	

THANK YOU

