Data Analysis:

Visualizing High-Dimensional MNIST Data using t-SNE

ESC 2023 Fall Final Project 3조

목차 / List

1

Introduction & Mathematical Overview

PCA, t-SNE, UMAP

Introduction

차원 축소 알고리즘

PCA

Principal Component Analysis

t-SNE

t-distributed Stochastic Neighbor Embedding

UMAP

Uniform Manifold Approximation and Prediction

- n개의 관측치와 p개의 변수로 구성된 데이터를 상관관계가 없는 k개의 변수로 구성된 데이터로 요약
- 이때 요약된 변수는 기존 변수들의 선형 조합으로 생성됨
- 원래 데이터의 분산을 최대한 보존하는 새로운 축을 찾고 그 축에 데이터를 projection하는 기법

Introduction

차원 축소 알고리즘

PCA

Principal Component Analysis

t-SNE

t-distributed Stochastic Neighbor Embedding

UMAP

Uniform Manifold Approximation and Prediction

- 2002년 샘 로이스와 제프리 힌튼에 의해 개발된 SNE를 확장시킨 방법으로 비선형적인 차원 축소 방식
- SNE는 기존 데이터와 축소된 차원에서의 데이터의 차이를 최소화하며 local structure를 보존함
- 이러한 SNE에 t분포를 적용해 global structure를 반영한 것이 t-SNE임

Introduction

차원 축소 알고리즘

PCA

Principal Component Analysis

t-SNE

t-distributed Stochastic Neighbor Embedding

UMAP

Uniform Manifold Approximation and Prediction

- 고차원에서 데이터를 graph로 만들고 저차원으로 graph를 projection시킨 방법
- 각 node에서 길이 k의 radius을 그린 후 radius가 겹치는 정도로 connection의 weight를 결정함
- 리만 매니폴드, 퍼지이론 등에 기반해 이러한 구조를 저차원으로 이동시킴

Mathematical Overview

Cost Function of t-SNE

(1) High dimensional probabilities p 계산

p_{ij} 를 데이터 포인트 i와 j의 similarity를 반영하는 스코어라고 하자. 두 포인트 사이에 symmetric한 확률 값을 갖도록 다음과 같이 정의할 수 있다.

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

$$p_{i,j} = \frac{p(j|i) + p(i|j)}{2N}$$

(2) Low dimensional probabilities q 계산

마찬가지의 방법으로 다음과 같이 구한다.

$$q_{i,j} = \frac{(1+||y_i - y_j||^2)^{-1}}{\Sigma_{k \neq l}(1+||y_i - y_l||^2)^{-1}}$$

Mathematical Overview

Cost Function of t-SNE

(3) C(p,q) 정의 후 최소화

Kullback-Leibler divergence를 사용해 cost function을 정의한다. 이를 통해 구한 KL divergence를 최소화 시키는 저차원 공간상의 데이터 위치를 gradient optimization을 통해 구할 수 있다.

$$Cost = KL(P||Q) = \sum_{\mathrm{i}} \sum_{\mathrm{j}} p_{ji} log \frac{p_{ji}}{q_{ji}}$$

$$\frac{\partial C}{\partial y_i} = 4 \sum_{\mathrm{j}} (p_{ji} - q_{ji})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\partial C}{\partial \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)})$$

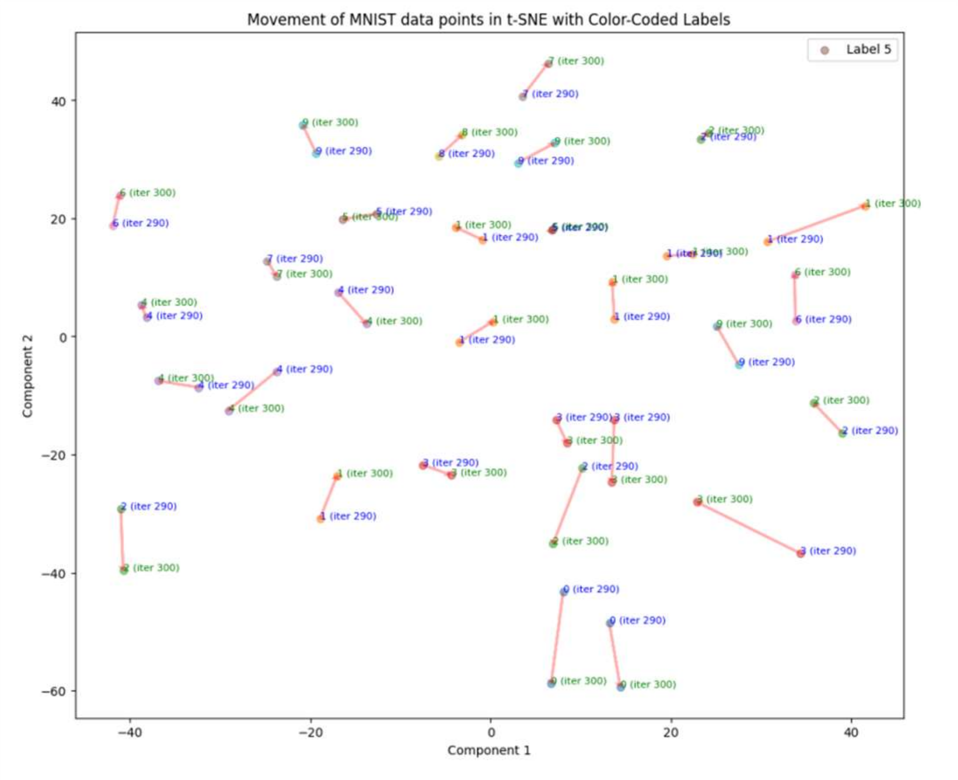
$\Upsilon^{(t)}$: iteration t에서의 solution

η: learning rate

$\alpha(t)$: iteration t에서의 momentum

Mathematical Overview

Cost Function of t-SNE



Iteration 290에서 300으로 갈 때,
저차원의 포인트들이 어떻게 움직이는지 나타나 있음

Mathematical Overview

Perplexity of t-SNE

다시 $p_{j|i}$를 살펴보면 σ_i를 선택해야 한다. 이때 특정한 값의 σ_i는 아래와 같은 entropy와 perplexity를 생성하게 된다.

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

$$Perp(P_i) = 2^{H(P_i)}$$

이때 $H(P_i)$는 Shannon entropy: $$H(P_i) = -\sum_j p_{j|i} log_2 p_{j|i}$$

Perplexity는 5-50 사이의 값에서 robust한 특성을 가지며 고려할 이웃 노드의 수를 결정한다.

따라서 perplexity의 값이 높을수록 더 많은 이웃 노드(global structure)를,

낮을수록 더 적은 이웃 노드(local structure)를 고려하게 된다.

2

MNIST Data: Implementation & Visualization

Clustering MNIST dataset

Importing Libraries

```
[ ] #importing library
    import numpy as np
    import pandas as pd

    # for visualization purpose
    import matplotlib.pyplot as plt
    from plotnine import ggplot, aes, geom_point, facet_wrap, theme, ggtitle # 별도 설치 필요 (pip install --user plotnine)

    # for doing PCA
    from sklearn.decomposition import PCA

    # for t-SNE implemention
    from sklearn.manifold import TSNE

    # for data loading
    from sklearn.datasets import fetch_openml

    # for comparing computation time
    import time
```

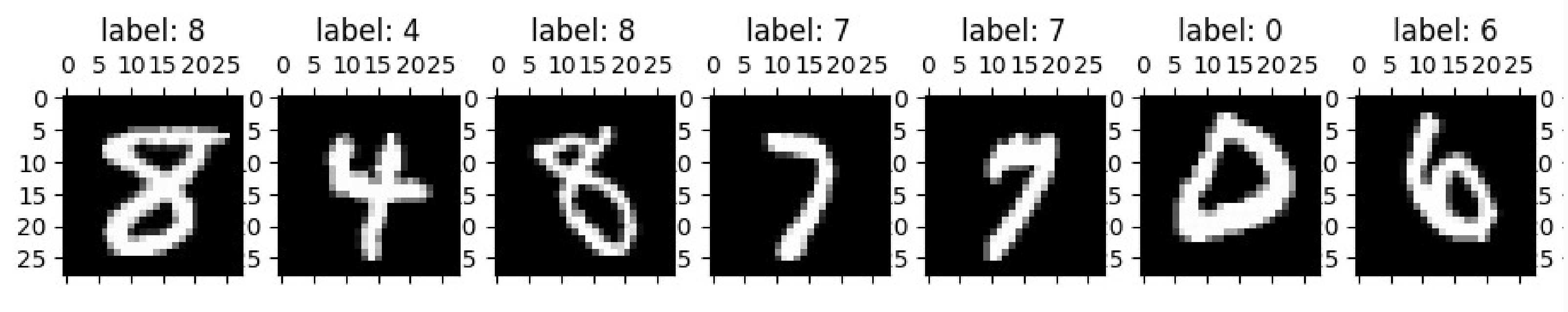
Data Preprocessing

MNIST Dataset

MNIST는 손으로 쓰여진 숫자 이미지로 구성된 70,000 * 784 크기의 데이터

결측치는 없으며 각 픽셀의 값이 0~255 사이의 값을 가짐

따라서 MNIST data의 경우 보통 정규화하는 방식인 특성별 정규화를 하는 것이 아니라, 각 셀에 255를 나누는 방식으로 normalize해서 각 셀의 값이 0~1의 값을 갖도록 함



Data Preprocessing

MNIST Dataset

```
[ ] pd.set_option('display.max_columns', 800)

    #데이터 불러오기
    mnist = fetch_openml('mnist_784')
    df = mnist.data
    columns_list = df.columns.to_list()

    #데이터 확인 및 정규화
    df.head()
    df.describe()
    df.info()
    df = df / 255.0

    #라벨 추가(실제 값)
    df['label'] = mnist.target
```

Data Preprocessing

t-SNE

- t-SNE는 $O(n^2)$의 시간 복잡도를 가지기 때문에 전체 데이터 중에서 7,000개만 뽑아서 구현해 보았음
- 계산 비용과 노이즈를 억제하기 위해 일차적으로 PCA를 사용해 30차원으로 축소한 뒤 t-SNE를 적용함

```
[ ] # t-sne는 계산비용이 커서 7000개만 사용
    n_sne = 7000

    # 계산 비용과 노이즈를 억제하기 위해 일차적으로 PCA를 통해 30차원으로 축소
    pca = PCA(n_components=30)
    pca_result = pca.fit_transform(df.loc[index[:n_sne], columns_list].values)
```

Implementation and Visualization

t-SNE

- t-SNE의 파라미터(number of iterations, perplexity, learning rate, momentum)을 조정하며 결과 비교

```
[ ] tsne1 = TSNE(n_components = 2, perplexity = 2, n_iter = 300)
    tsne_result1 = tsne1.fit_transform(pca_result)

    tsne2 = TSNE(n_components = 2, perplexity = 30, n_iter = 300)
    tsne_result2 = tsne2.fit_transform(pca_result)

    tsne3 = TSNE(n_components = 2, perplexity = 50, n_iter = 300)
    tsne_result3 = tsne3.fit_transform(pca_result)

    tsne4 = TSNE(n_components = 2, perplexity = 100, n_iter = 300)
    tsne_result4 = tsne4.fit_transform(pca_result)

    tsne5 = TSNE(n_components = 2, perplexity = 1000, n_iter = 300)
    tsne_result5 = tsne5.fit_transform(pca_result)

    tsne6 = TSNE(n_components = 2, perplexity = 5000, n_iter = 300)
    tsne_result6 = tsne6.fit_transform(pca_result)
```

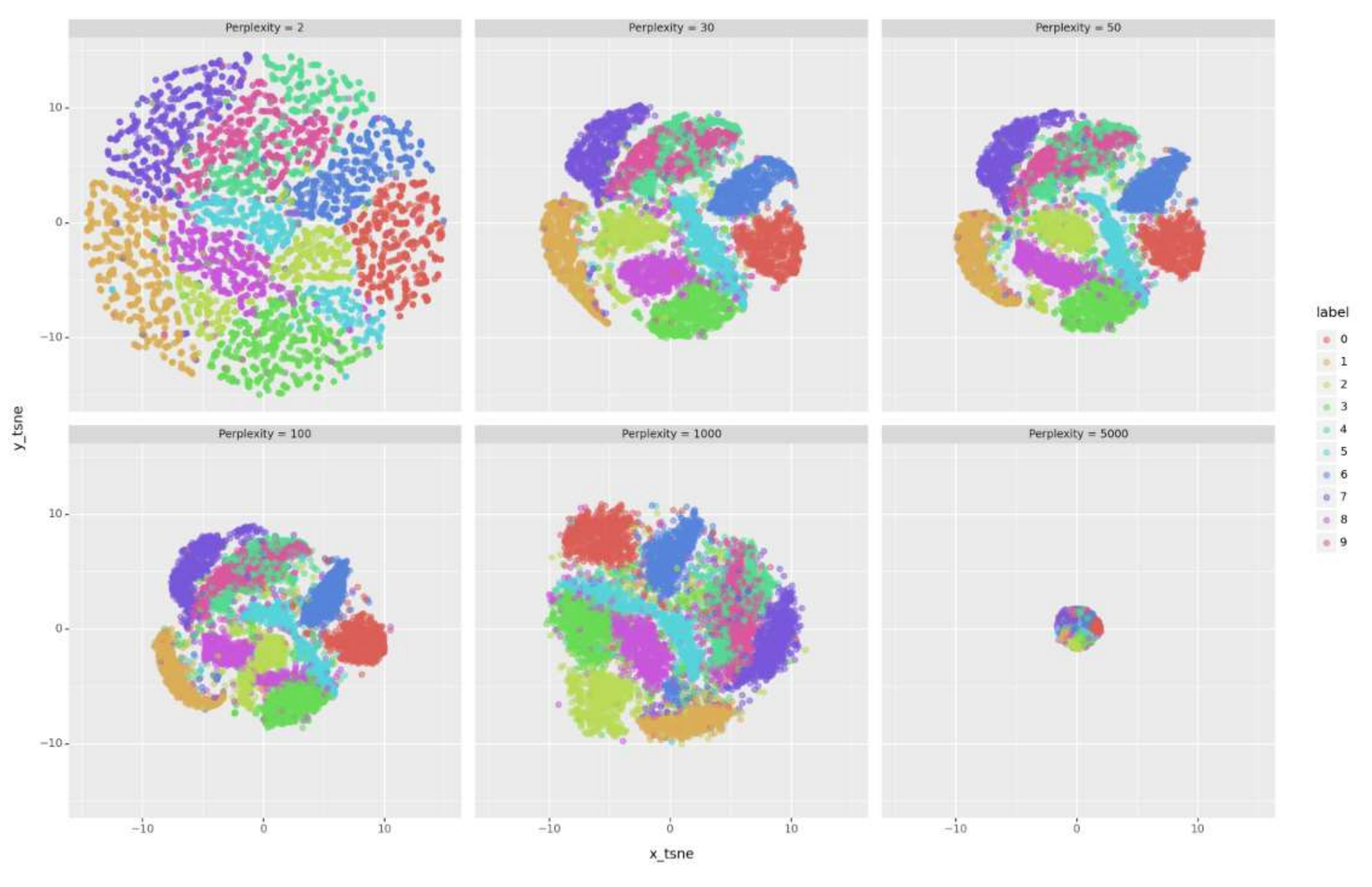
Implementation and Visualization

t-SNE

- 시각화 코드

```
from plotnine import ggplot, aes, geom_point, facet_wrap
import pandas as pd

# Create a DataFrame to store all tsne_results
df_list = []

perplexities = [2, 30, 50, 100, 1000, 5000]

# Convert 'result' to a categorical type with a specified order
perplexity_order = pd.Categorical(['Perplexity = {}'.format(p) for p in perplexities], categories=['Perplexity = {}'.format(p) for p in perplexities])

for i, tsne_result in enumerate([tsne_result1, tsne_result2, tsne_result3, tsne_result4, tsne_result5, tsne_result6]):
    df_tsne = df.loc[index[:n_sne], :].copy()
    df_tsne['x_tsne'] = tsne_result[:, 0]
    df_tsne['y_tsne'] = tsne_result[:, 1]
    df_tsne['result'] = 'Perplexity = {}'.format(perplexities[i])
    df_list.append(df_tsne)

# Combine all results into a single DataFrame
df_combined = pd.concat(df_list, ignore_index=True)

# Set the order of the 'result' column based on perplexity_order
df_combined['result'] = pd.Categorical(df_combined['result'], categories=perplexity_order, ordered=True)

# Plot using ggplot with facet_wrap and adjusted figure size
chart = ggplot(df_combined, aes(x='x_tsne', y='y_tsne', color='label')) + geom_point(size=2, alpha=0.5) + facet_wrap('~result', ncol=3) + theme(figure_size=(15, 10))
chart.draw()
```

Implementation and Visualization



Implementation and Visualization

t-SNE: Perplexity

- Perplexity가 너무 작거나 크지만 않으면 (5~50) 군집이 잘 형성됨 → robust
- 단 데이터 수에 비해 perplexity가 너무 크면 군집을 제대로 형성하지 못하고 점들이 균등하게 분포함
- 이는 perplexity가 너무 크면 local한 구조를 보존하기 못하기 때문임

→ 더 많은 이웃을 포함하며 작은 그룹이 무시됨

- 데이터 크기가 크면 perplexity를 크게 설정하는 것이 좋음
- Perplexity가 작으면 주변 데이터 포인트를 적게 고려하게 되어 local한 구조를 과도하게 강조하게 됨

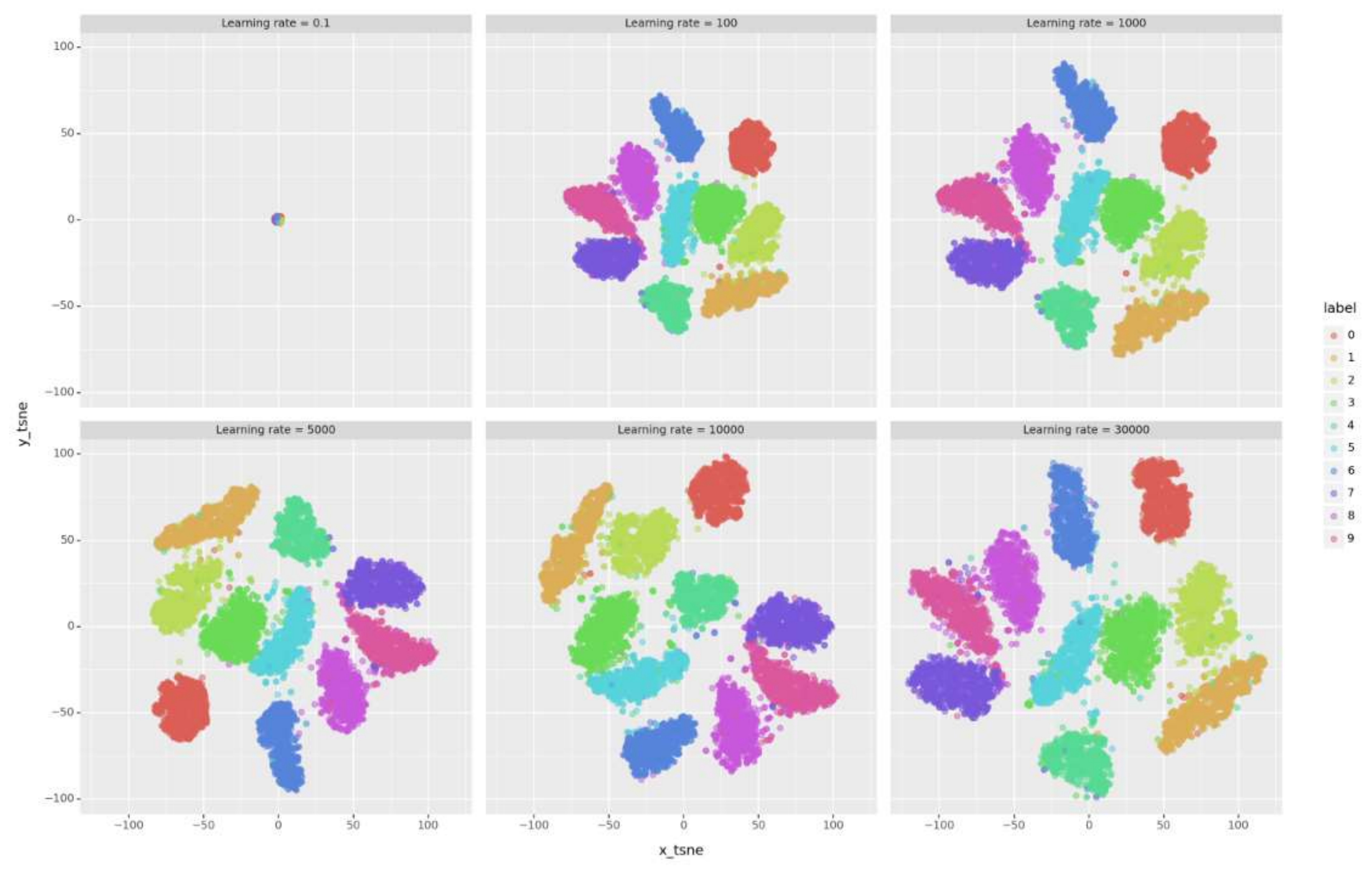
→ global한 구조를 놓치게 되고 특히 클러스터 간 거리를 적절히 표현하지 못함

Implementation and Visualization

t-SNE: Learning rate

```
[ ] tsne_results = []
    learning_rate = [0.1, 100, 1000, 5000, 10000, 30000]

    for learn in learning_rate:
        # Create and fit t-SNE model
        tsne = TSNE(n_components=2, learning_rate = learn, random_state=42) #perplexity default = 30
        tsne_result = tsne.fit_transform(df.loc[index[:n_sne], :])

        # Append the result to the list
        tsne_results.append(tsne_result)

    # Create a DataFrame to store all tsne_results
    df_list = []
    learn_order = pd.Categorical(['Learning rate = {}'.format(i) for i in learning_rate], categories=['Learning rate = {}'.format(i) for i in learning_rate])


    for i, learn in enumerate(learning_rate):
        tsne_result = tsne_results[i]  # Assuming tsne_results is a list containing your t-SNE results
        df_tsne = df.loc[index[:n_sne], :].copy()
        df_tsne['x_tsne'] = tsne_result[:, 0]
        df_tsne['y_tsne'] = tsne_result[:, 1]
        df_tsne['result'] = 'Learning rate = {}'.format(learn)
        df_list.append(df_tsne)

    # Combine all results into a single DataFrame
    df_combined = pd.concat(df_list, ignore_index=True)
    df_combined['result'] = pd.Categorical(df_combined['result'], categories=learn_order, ordered=True)

    # Plot using ggplot with facet_wrap and adjusted figure size
    chart = ggplot(df_combined, aes(x='x_tsne', y='y_tsne', color='label')) + geom_point(size=2, alpha=0.5) + facet_wrap('~result', ncol=3) + theme(figure_size=(15, 10))
    chart.draw()
```

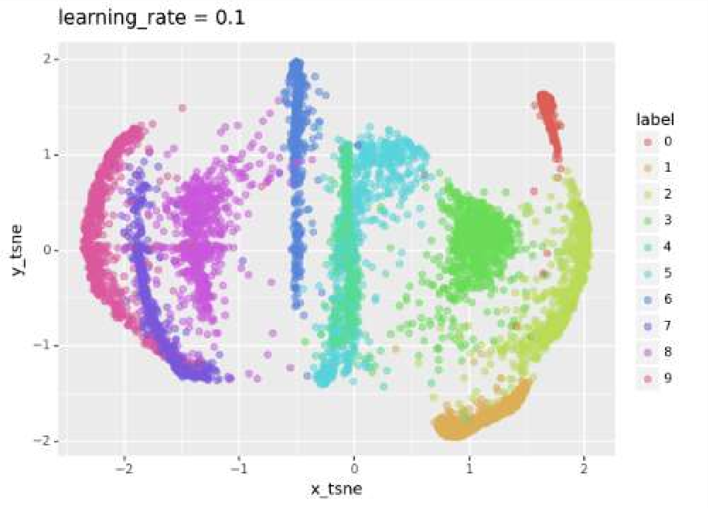
Implementation and Visualization



Implementation and Visualization

t-SNE: Learning rate

- Learning rate는 일반적으로 10~1000 사이 값을 사용함
- Learning rate의 기본값은 max(N/early_exaggeration/4, 50)



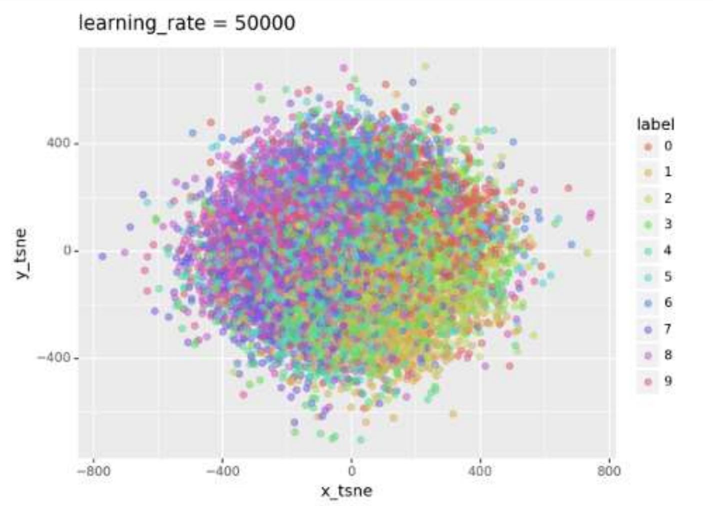
learning rate가 너무 낮으면 대부분의 포인트가 특이차가 거의 없는 조밀한 클라우드에서 압축된 것처럼 보일 수 있음

또한 learning rate가 너무 낮으면 local minimum에 빠질 수 있음
→ learning rate를 증가시켜주면 해결

Implementation and Visualization

t-SNE: Learning rate

- Learning rate는 일반적으로 10~1000 사이 값을 사용함
- Learning rate의 기본값은 max(N/early_exaggeration/4, 50)



Learning rate가 너무 높으면 데이터가 가장 가까운 이웃과 거의 같은 거리에 있는 '공'처럼 보일 수 있음

+ scikit-learn의 t-SNE에는 momentum을 조정하는 파라미터가 없음 → 다른 모델의 경사하강법과 다른 최적화 과정을 거치기 때문

3

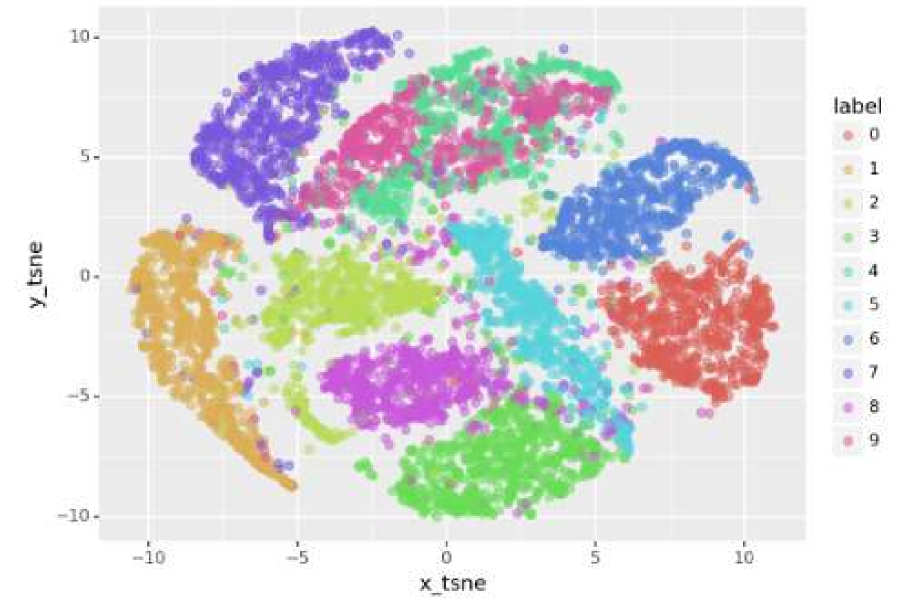
MNIST Data: Analysis & Interpretation

Comment on the clusters

Analysis and Interpretation

t-SNE: clustering

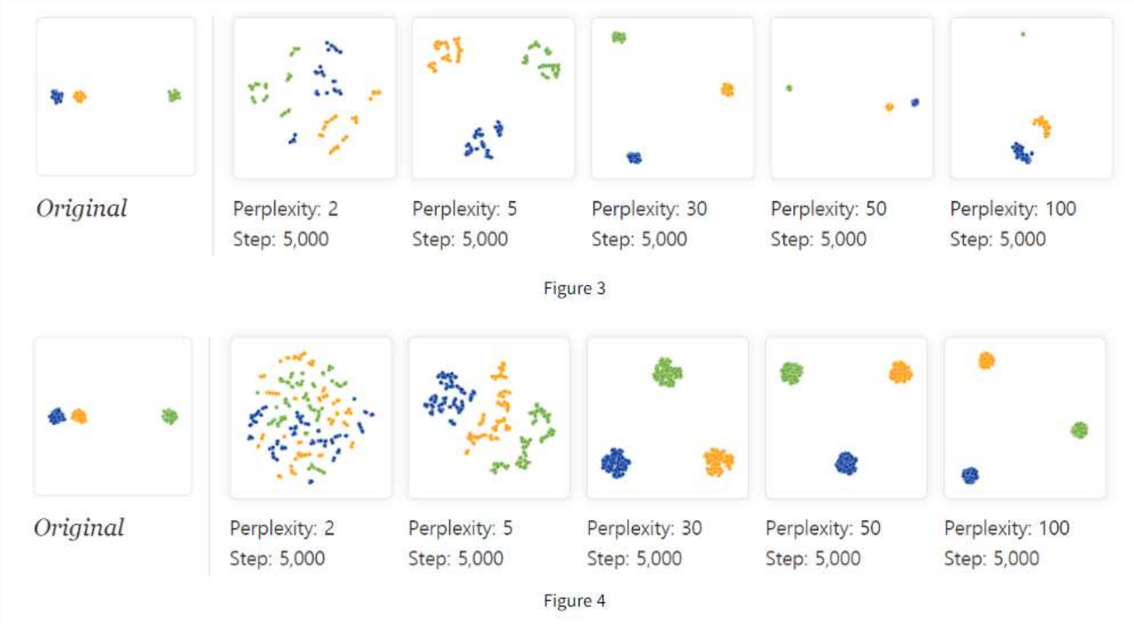
가장 성능이 좋아 보이는 t-SNE plot (perplexity=30)을 통해 결과를 분석함



- 4와 9의 군집이 겹쳐져 있음
- 8, 3, 5의 군집이 가까이 위치하며 이상치가 서로의 군집에 포함되어 있음
- 7의 이상치가 주로 9의 군집에 포함되어 있음
- 숫자가 비슷하게 생긴 것끼리 군집이 겹쳐져 있음
- 6, 0, 7, 1이 다른 군집과 거리가 떨어져 보임 → 의미는?

Analysis and Interpretation

t-SNE: Distance between clusters



출처:
https://hongl.tistory.com/175

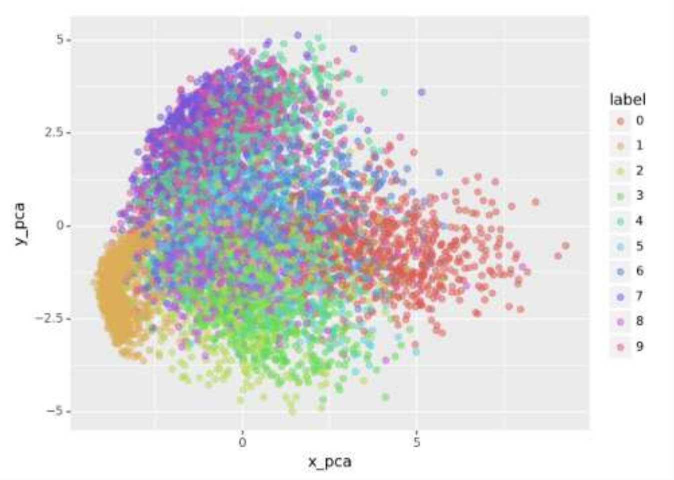
위의 그림에서 볼 수 있듯, t-SNE 결과 상에서 군집 간의 거리는 아무런 의미도 갖지 않음

4

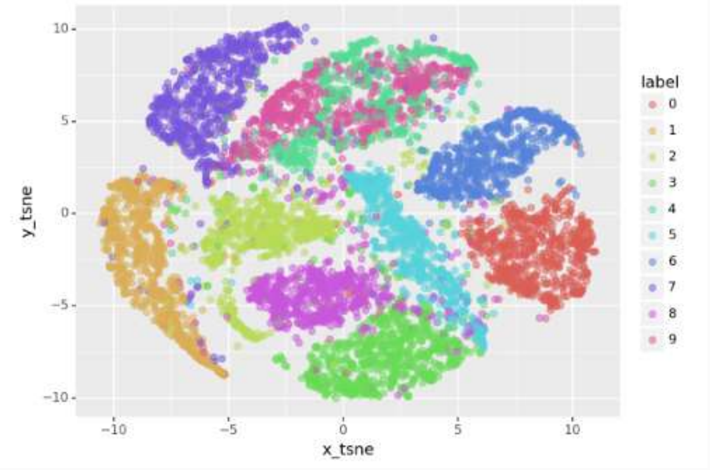
Comparison with PCA & UMAP

What insights can be gained?

Comparison with PCA



PCA



t-SNE (perplexity = 30)

- t-SNE와 비교하기 위해 마찬가지로 7,000개만 추출하여 PCA를 구현함
- PCA: 0과 1은 어느 정도 구분되나, 나머지 숫자에 대해선 2차원 상에서 거의 구분하지 못함

Comparison with PCA

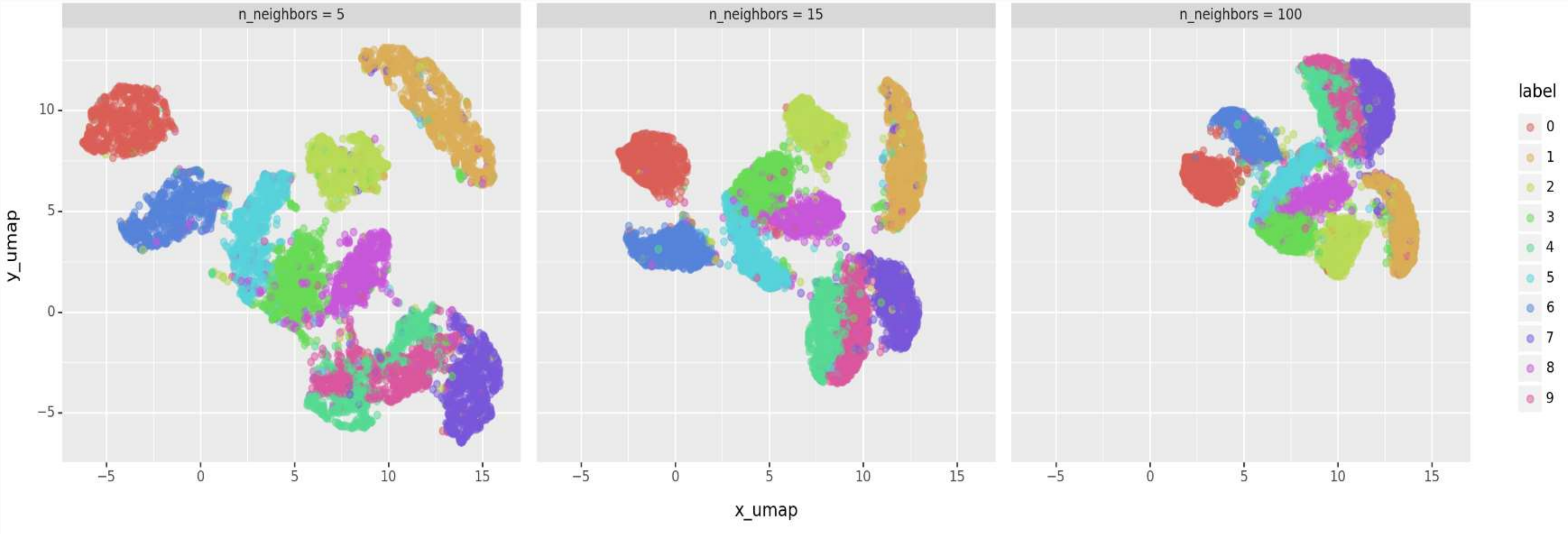
PCA

```
[ ] # pca 결과
    df_pca = df.copy()
    df_pca['x_pca'] = pca_result1[:, 0]
    df_pca['y_pca'] = pca_result1[:, 1]

    df_pca_7000 = df_pca.loc[index[:n_sne],:].copy() # t-sne 와 비교하기 위해 마찬가지로 7000개만 추출

    chart = ggplot(df_pca_7000, aes(x='x_pca', y='y_pca', color='label')) + geom_point(size=2, alpha=0.5)
    chart
```

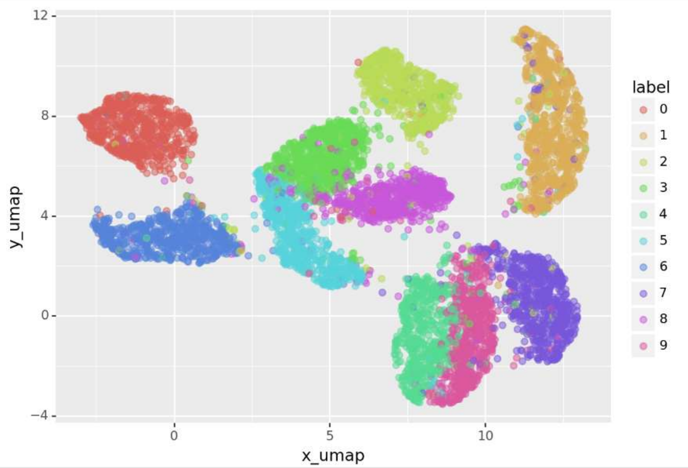
Comparison with UMAP

UMAP

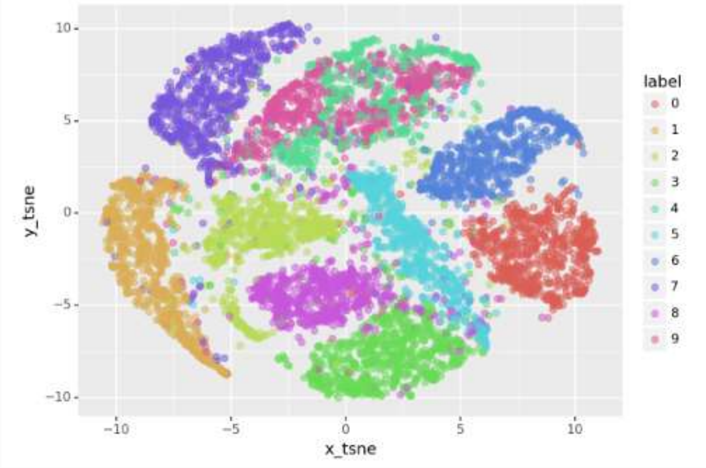
- t-SNE와 마찬가지로 일차적으로 PCA를 사용해 30차원으로 축소한 뒤 UMAP 적용
- n_neighbors가 작으면 local한 구조를 잘 살림



Comparison with UMAP



UMAP (nearest neighbors = 15)



t-SNE (perplexity = 30)

- UMAP이 t-SNE보다 시각화 잘 됨
- UMAP의 nearest neighbors는 t-SNE의 perplexity와 같은 개념이라고 볼 수 있음

Comparison with UMAP

T-SNE vs UMAP

```
print(f"T-SNE 소요 시간: {tsne_elapsed_time} 초")
```

T-SNE 소요 시간: 10.662742137908936 초

```
print(f"UMAP 소요 시간: {umap_elapsed_time} 초")
```

UMAP 소요 시간: 6.471599102020264 초

UMAP (nearest neighbors = 5)

t-SNE (perplexity = 2)

- 속도 역시 UMAP이 더 빠른 것을 확인할 수 있음

Comparison with UMAP

```
[ ] from umap import UMAP

    #UMAP
    umap1 = UMAP(n_components=2, n_neighbors=5, min_dist=0.25)
    umap_result1 = umap1.fit_transform(pca_result)

    umap2 = UMAP(n_components=2, n_neighbors=15, min_dist=0.25)
    umap_result2 = umap2.fit_transform(pca_result)

    umap3 = UMAP(n_components=2, n_neighbors=100, min_dist=0.25)
    umap_result3 = umap3.fit_transform(pca_result)

    #시각화
    df_list = []

    num_neighbors = [5, 15, 100]

    # Convert 'result' to a categorical type with a specified order
    neighbor_order = pd.Categorical(['n_neighbors = {}'.format(i) for i in num_neighbors], categories=['n_neighbors = {}'.format(i) for i in num_neighbors])

    for i, umap_result in enumerate([umap_result1, umap_result2, umap_result3]):
        df_umap = df.loc[index[:n_sne], :].copy()
        df_umap['x_umap'] = umap_result[:, 0]
        df_umap['y_umap'] = umap_result[:, 1]
        df_umap['result'] = 'n_neighbors = {}'.format(num_neighbors[i])
        df_list.append(df_umap)

    # Combine all results into a single DataFrame
    df_combined = pd.concat(df_list, ignore_index=True)

    # Set the order of the 'result' column based on perplexity_order
    df_combined['result'] = pd.Categorical(df_combined['result'], categories=neighbor_order, ordered=True)

    # Plot using ggplot with facet_wrap and adjusted figure size
    chart = ggplot(df_combined, aes(x='x_umap', y='y_umap', color='label')) + geom_point(size=2, alpha=0.5) + facet_wrap('~result', ncol=3) + theme(figure_size=(15, 5))
    chart.draw()
```

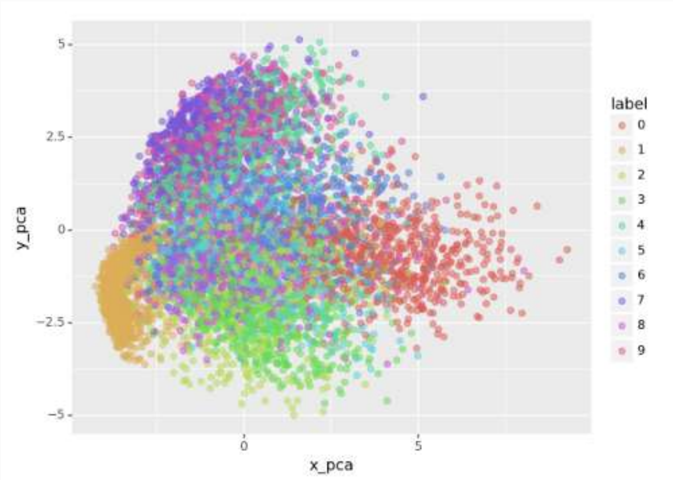
5

Conclusion

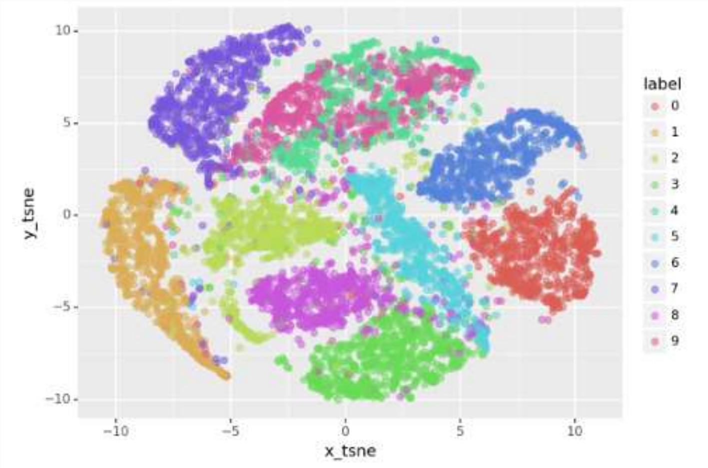
About dimensional reduction

PCA, t-SNE, UMAP

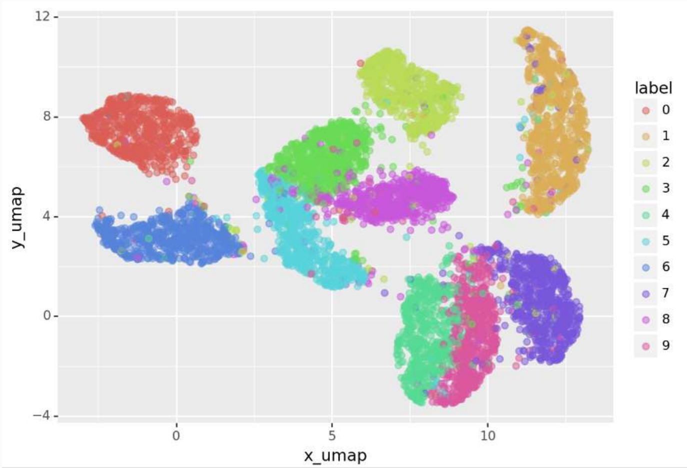
- UMAP, t-SNE, PCA 순으로 좋은 성능을 보이고 있음



PCA



t-SNE (perplexity = 30)

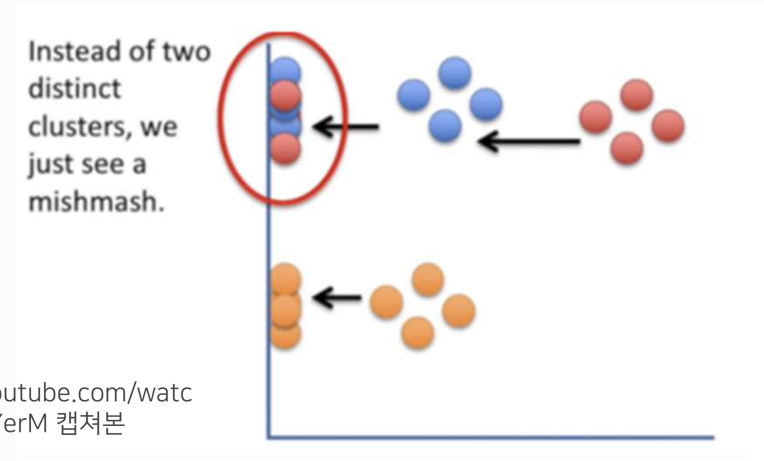


UMAP (nearest neighbors = 15)

PCA, t-SNE, UMAP

PCA

- Matrix factorization base의 차원 축소 알고리즘
- Hyperparameter가 없다는 점에서 유용함
- 선형 방식으로 projection하면서 차원의 축소, 군집된 데이터들이 뭉개지는 단점이 있음
- 이에 따라 시각화 결과가 t-SNE보다 좋지 못함



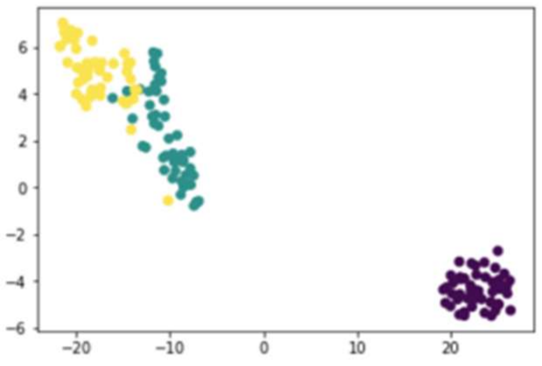
출처: https://www.youtube.com/watch?v=NEaUSP4YerM 캡쳐본

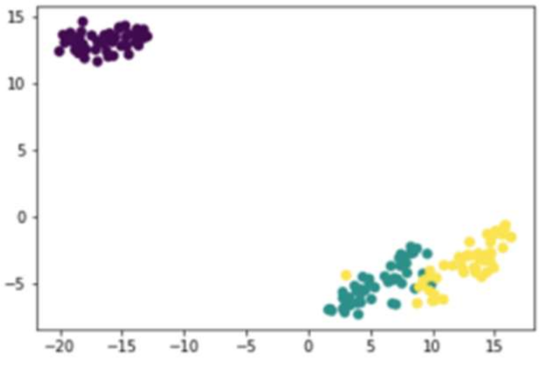
PCA, t-SNE, UMAP

t-SNE

- Neighboring graph base의 non-linear한 방법
- 고차원 벡터의 유사성을 저차원에서도 보존 (local neighbor structure 보존)
- 고차원 데이터를 2, 3차원으로 줄여 시각화하는데 활용함
- 데이터의 개수가 n개라면 연산량은 n의 제곱만큼 늘어난다는 단점이 있음
- 매번 돌릴 때마다 다른 시각화 결과가 나옴 (training과 prediction을 동시에 하므로 학습에 활용할 수 없음)

출처:
https://bcho.tistory.com/1210



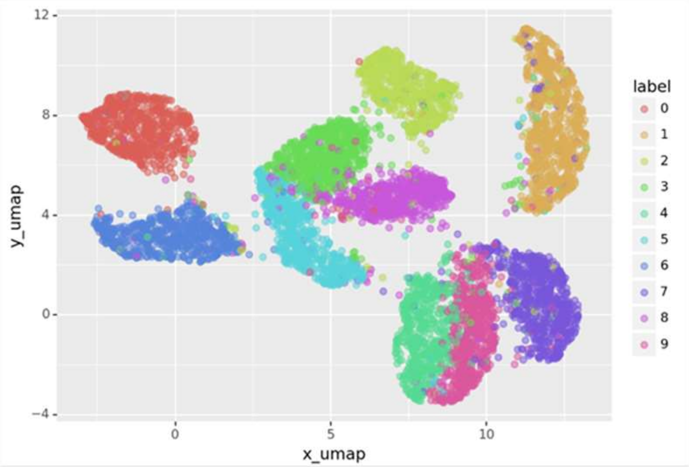


PCA, t-SNE, UMAP

UMAP

- Neighboring graph base로 하며 가장 좋은 성능을 내는 알고리즘
- 속도가 빠르며 embedding 차원 크기에 대한 제한이 없어서 일반적인 차원 축소 알고리즘으로 적용 가능
- Global structure을 더 잘 보존함
- 저차원에서 초기값을 initialize할 때 spectral embedding 방식을 사용하기 때문에 결과가 조금 더 일관적으로 나옴(학습에 활용할 수 있음)
- Top2vec이라는 논문에서 최적의 parameter를 제안한 바 있음

→ n_neighbors = 15, min_dist = 0.25



Contribution

- 곽지석: 자료 정리 및 시각화 자료 제작, 프로젝트 진행 방향 설정
- 송시은: 파이널 발표 자료 제작, 코드 실행 결과 정리 및 비교
- 이혜림: 코드 구성, 코드 실행 결과 정리 및 비교
- 임승현: 중간 발표 자료 제작, 중간 발표
- 장덕재: 자료 검토, 파이널 발표

감사합니다