[Paper Implementation]

Visualizing Data using t-SNE

학번: 2020021327

작성자: 조경선

[논문 요약]

논문명: Visualizing Data using t-SNE

저 자: Laurens van der Maaten

발표일: 2008년 11월

개제저널: Journal of Machine Learning Research

Paper Contents Summary

데이터 포인트 위치를 2차원 또는 3차원의 맵에 나타내어 고차원 데이터를 시각화하는 t- SNE 라는 새로운 기술을 제시하였다. 이는 그 동안 알려져 있던 SNE(Stochastic Neighbor Embedding)의 변형 기술이다. 하지만 SNE가 가지고 있던 몇 가지의 단점을 잘 해결하여 훨씬 더 나은 시각화 기술을 선보였다.

(1) Symmetric SNE

SNE 에서 정의한 조건부 확률은 symmetric 하지 않는다. 이는 고차원에 outlier 를 발생 시키고 이를 저차원으로 변경을 하면, 아주 낮은 영향도를 갖는다. 이런 문제를 해결하기 위해서 두 점사이의 joint probabilities 를 정의하여 symmetric SNE 를 완성하였다. 이의 장점은 gradient 의 계산이 아주 빨라졌다는 것이다.

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

(2) The Crowding Problem

고차원에서 저차원으로 변경을 하면, distance 의 개념이 서로 많이 달라지는 경우가 있다.
이를 보완하고자 Low Dimensional Domain 에서는 Student t-Distribution을 사용하기로 했다.
왜냐하면 t-Distribution은 Gaussian 보다 꼬리가 긴 분포를 보이고 있기 때문에 가까운 data는 더 가까이 먼 data는 더 멀리 배치할 수 있다.

(3) Mismatched Tails can Compensate for Mismatched Dimensionalities

t-SNE는 (1) 큰 두 data 사이의 거리를 이용하여 서로 다른 data를 모델링하고, (2) 작은 두 data 사이의 거리를 이용하여 유사한 data를 모델링 하는 것을 목적으로 한다.

또한 Gaussian 보다 t-Distribution은 지수가 포함되지 않아서, 계산을 더 빠르게 할 수 있다. 그래서 t-SNE의 비용함수는 그 어떤 것보다도 작다.

이를 증명하기 위해 t-SNE 와 Sammon mapping, Isomap, LLE 의 시각화 결과을 비교하였다.(추가자료로 CCA, SNE, MVU, Laplacian Eigenmaps 도 기재함) 비교를 위하여 사용한 Data Set 은 다음과 같다.

(1) the MNIST data set
(2) the Olivetti faces data set
(3) the COIL-20 data set
(4) the word-features data set
(5) the Netflix data set

Simple Algorithm

논문에서 제시한 Simple Algorithm 은 아래와 같다.

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for $t=1$ **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right)$
  end
end

이 중 SNE에서 변화된 로직인 Equation 4 와 Equation 5를 python 으로 구현해 보았다.

Equation 4

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

```
def equation4(params, samples, dimensions):
    """
    :param params: array, shape (n_params,)
        Unraveled embedding.
    :param samples: [int] sample 의 갯수
  :param dimensions: [int] 공간의 차수
  :return: [float] joint probabilities Q
    """

    X = params.reshape(samples, dimensions)

    from scipy.spatial.distance import pdist
    import numpy as np
    org_distance = pdist(X, "sqeuclidean")
    comput_distance = org_distance ** 2.0 + 1.0
    comput_distance = np.invert(comput_distance)
    Q = comput_distance / np.sum(comput_distance)

    return Q
```

Equation 5

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$$

```
def equation5(P, Q, org_distance, comput_distance):
    """
    :param P: [float] equation 1 의 결과인 joint probabilities
    :param Q: [float] equation 4 의 결과인 joint probabilities
    :param org_distance: [float] equation 4 에서 구한 원거리
    :param comput_distance: [float] equation 4 에서 구한 계산된 거리
    :return: [float] the gradient of the Kullback-Leibler divergence
    """
    return  4 * np.sum((P - Q) * org_distance * comput_distance)
```

Example

논문에서는 다양한 Data와 기법을 t-SNE 와 비교하였다. 그 중 수업시간에 배운 Isomap, LLE 와 t-SNE 를 라이브러리를 이용하여 Test 해보고자 한다. Test의 데이터는 MNIST 를 사용한다.

Step1. Dataset Preparation

먼저 Dataset의 전처리를 실행한다. 전처리 과정을 다음과 같다.

1. MNIST data를 sklearn datasets 에서 로드
2. Data Frame을 생성
3. n=10000 인 Random Data를 추출

전처리 code는 아래와 같다.

```
from __future__ import print_function
import time
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.manifold import TSNE, LocallyLinearEmbedding, Isomap
from sklearn.cross_decomposition  import CCA
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

# Load MNIST dataset
mnist = fetch_openml("mnist_784")

X = mnist.data / 255.0
y = mnist.target

# Dataframe 만들기
cols = ['pixel'+str(i) for i in range(X.shape[1]) ]

df = pd.DataFrame(X,columns=cols)
df['y'] = y
df['label'] = df['y'].apply(lambda i: str(i))

X, y = None, None

# For reproducability of the results
np.random.seed(42)
permutation = np.random.permutation(df.shape[0])

# Random 으로 추출한 Data 중 10000 개를 사용
n = 10000

df_subset = df.loc[permutation[:n],:].copy()
data_subset = df_subset[cols].values
```

이렇게 추출된 data_subset을 차원축소에 사용하고, df_subset 을 시각화에 사용한다.

Step2. Modified Dataset Transform

오픈소스로 나오있는 라이브러리를 이용하면, 차원축소가 쉽게 구현된다.

[t-SNE]

```
# t-SNE
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(data_subset)
```

[Isomap]

```
# Isomap
isomap = Isomap(n_components=2, n_neighbors=5)
isomap_results = isomap.fit_transform(data_subset)
```

[LLE]

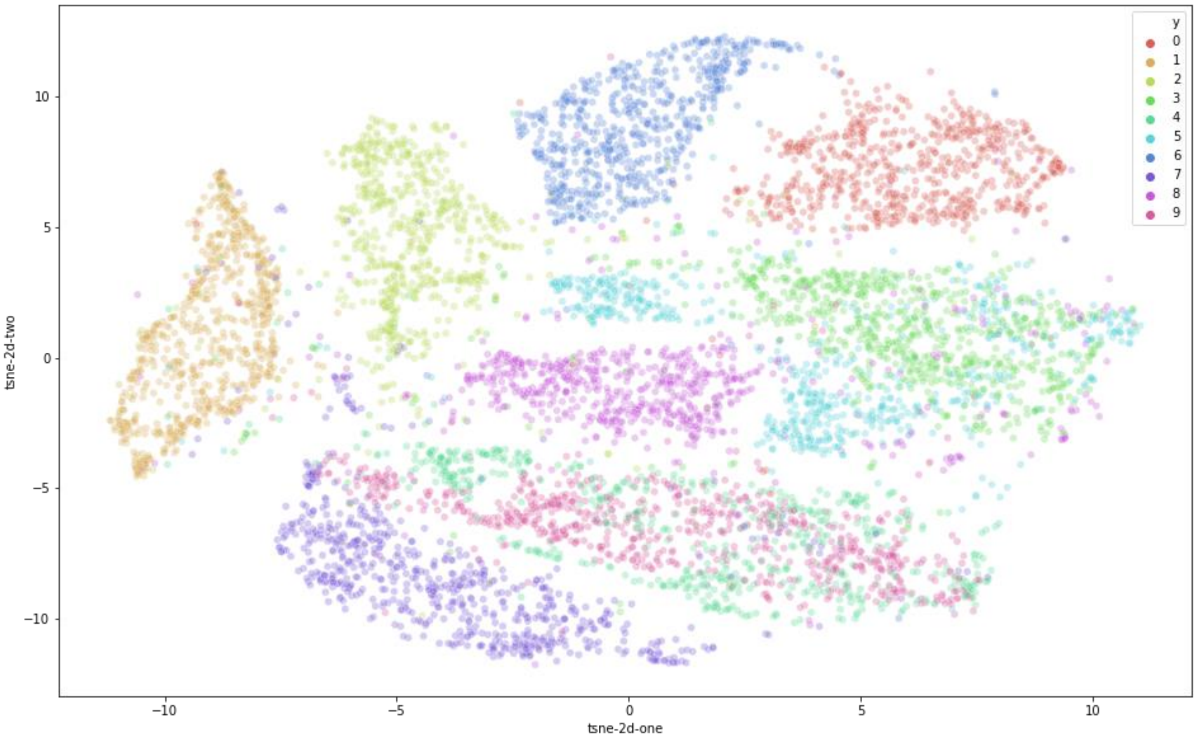
```
# LLE
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=5)
lle_results = lle.fit_transform(data_subset)
```

Step3. Visualization of Result

위 과정을 통해 나온 세가지 결과를 비교해 보자.

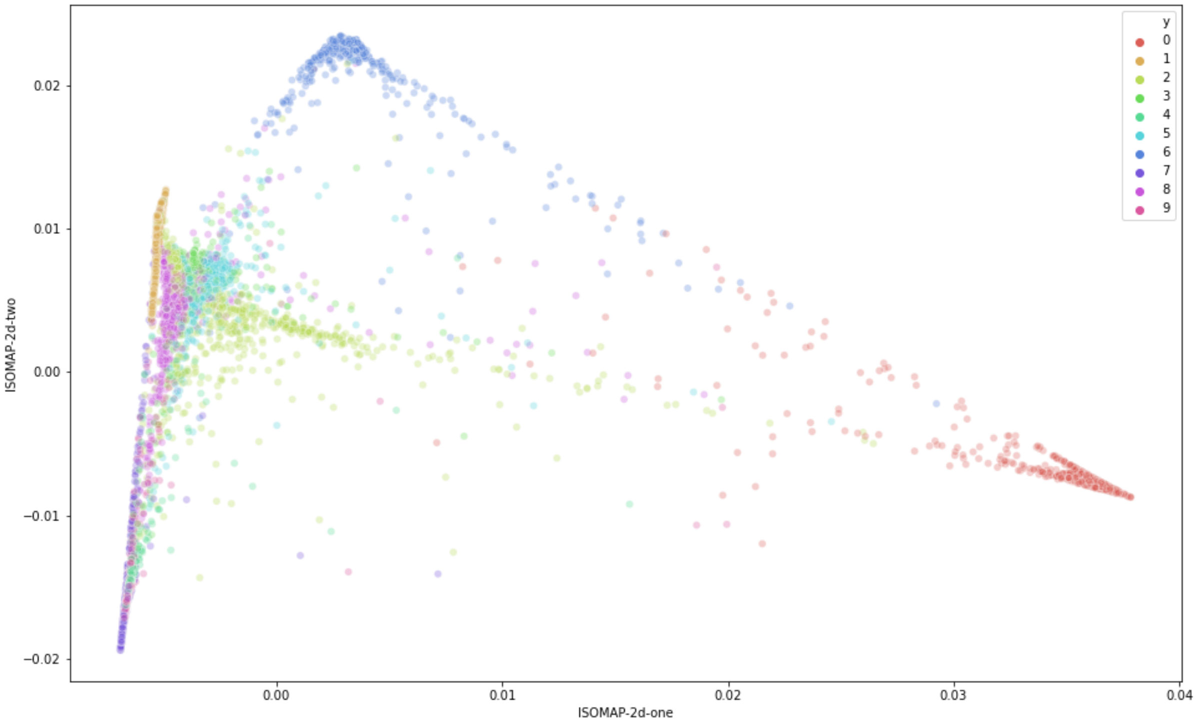
[t-SNE]

```
#tsn 시각화
df_subset['tsne-2d-one'] = tsne_results[:,0]
df_subset['tsne-2d-two'] = tsne_results[:,1]
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3
)
```



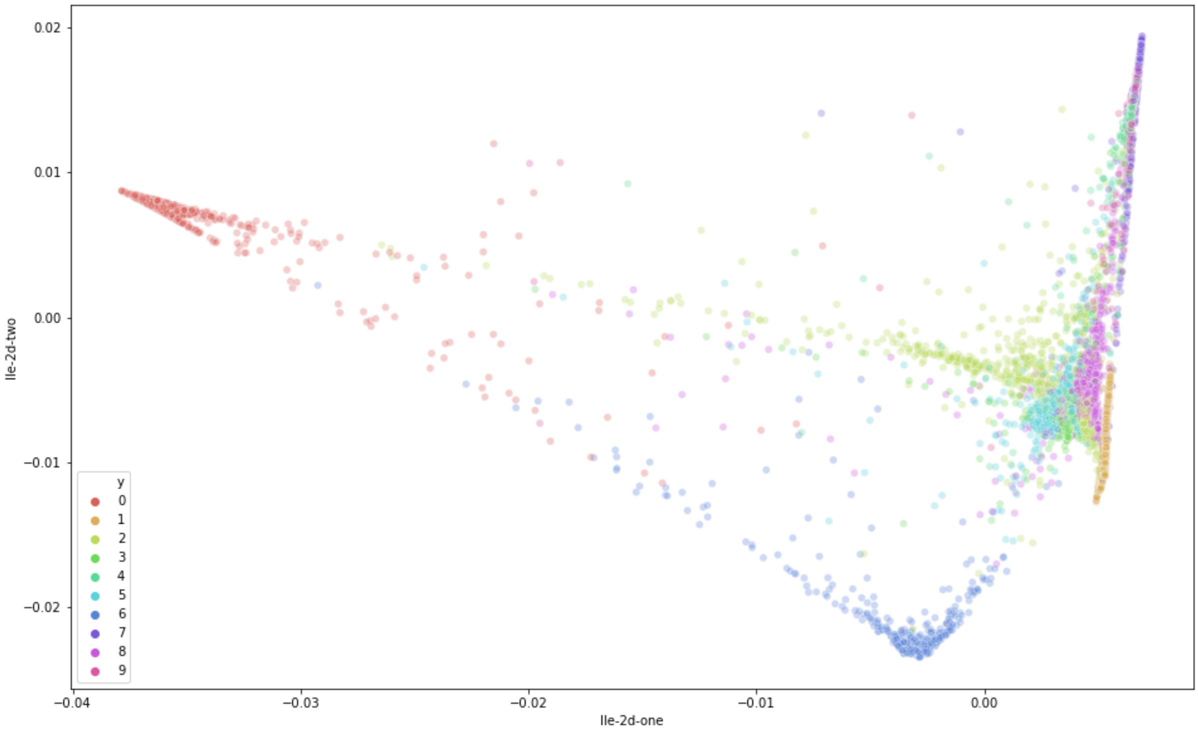
[Isomap]

```
# ISOMAP 시각화
df_subset['ISOMAP-2d-one'] = isomap_results[:,0]
df_subset['ISOMAP-2d-two'] = isomap_results[:,1]
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="ISOMAP-2d-one", y="ISOMAP-2d-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3
)
```



[LLE]

```
#LLE 시각화
df_subset['lle-2d-one'] = lle_results[:,0]
df_subset['lle-2d-two'] = lle_results[:,1]
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="LLE-2d-one", y="LLE-2d-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3
)
```



Step4. Conclusion of Result

라이브러리를 통하여 배운 t-SNE , Isomap, LLE 에 대하여 MNIST dataset 을 이용하여 그려보았다. 시각적으로 t-SNE 가 Isomap, LLE 보다 확연하게 잘 시각화되는 것을 알 수 있었다.

Conclusion

t-SNE 의 시초가 되는 논문을 읽고, 아주 간단한 식을 구현해 보았다. 해당 논문의 모든 algorithms 을 구현하였으면 좋았겠으나, 이번에는 현존하는 라이브러리로 그를 대신하였다. 라이브러리를 사용하는 것은 매우 간단한 작업으로 결과를 얻을 수 있었다. 하지만, 사용한 라이브러리의 code 를 확인한 결과, 사용한 라이브러리 함수에는 많은 parameter를 셋팅하고 있었다. 이는 t-SNE 이 이후 많은 연구를 통하여 발전된 형태를 함수에 녹여 있는 것으로 보였다.

따라서 사용한 라이브러리 함수에 적절한 parameter를 적용한다면, 더 좋은 결과를 얻을 수 있을 것으로 추정된다.