PharmaSUG 2025 - Paper AI-194

Visualize High Dimension Data Using t-SNE

Jun Yang, Avidity Biosciences

ABSTRACT

We introduce a novel technique named "t-SNE", designed for visualizing high-dimensional data by assigning each data point a location in a two or three-dimensional map. This method, a variant of Stochastic Neighbor Embedding (Hinton and Roweis, 2002), offers improved optimization ease and generates markedly superior visualizations by embedding the points from high dimension to low dimension trying to preserve the neighborhood of that point. t-SNE excels in creating a unified map that unveils structures across various scales, a crucial attribute when dealing with high-dimensional data distributed across distinct yet interrelated low-dimensional manifolds, such as images depicting objects from diverse classes and viewpoints.

To address the challenge of visualizing the structure within large datasets, we demonstrate how t-SNE leverages random walks on neighborhood graphs. This enables the implicit structure of the entire dataset to influence the display of a subset, enhancing the overall representation. We validate the efficacy of t-SNE across a diverse range of datasets, comparing its performance against various non-parametric visualization techniques, including Isomap, and Locally Linear Embedding. Notably, t-SNE consistently outperforms these techniques in producing superior visualizations for almost all datasets examined.

INTRODUCTION

High-dimensional datasets are increasingly common, often containing hundreds or thousands of features. Traditional visualization methods struggle to convey meaningful structure in such data. Dimensionality reduction techniques help mitigate this problem by projecting data into lower-dimensional spaces while preserving meaningful relationships. t-SNE, introduced by van der Maaten and Hinton in 2008, has gained widespread use for its ability to preserve local structures and produce visually interpretable clusters. This paper aims to provide a comprehensive overview of t-SNE and its practical applications.

BACKGROUND AND RELATED WORK

t-SNE builds upon earlier techniques like Stochastic Neighbor Embedding (SNE) by addressing key issues such as the crowding problem. It has been widely adopted in areas like single-cell RNA sequencing, image recognition, and word embedding visualization. Other techniques, such as Principal Component Analysis (PCA) also offer dimensionality reduction but differ in methodology and outcome.

The t-SNE ALGORITHM

t-SNE converts high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities. It constructs a probability distribution over pairs in the high-dimensional space and attempts to match this distribution in a low-dimensional space using a Student's t-distribution. The algorithm minimizes the Kullback-Leibler divergence between these two distributions through gradient descent. Given a set of high-dimensional data points, t-SNE computes pairwise similarities using a Gaussian distribution. The similarity between two points i and j in the high-dimensional space is given by:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)}$$

APPLICATIONS AND EXAMPLE

I will introduce how to apply t-SNE to the MNIST dataset available in sklearn.datasets and compare the results with PCA (principal component analysis) and t-SNE effectively separate digit clusters, revealing local structure that PCA failed to capture.

```
from __future__ import print_function
import time
import numpy as np
import pandas as pd

from sklearn.datasets import fetch_openml #scikit-learn
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
import os
os.environ['OMP_NUM_THREADS'] = '10'
```

These libraries are required to run PCA and t-SNE and plot diagrams for visualizing clusters of different handwritten digits.

- **NumPy** is used for manipulating numerical data and arrays.
- **Pandas** is used to read datasets and manage them using DataFrame structures.
- **Scikit-learn (sklearn)** provides machine learning tools, including popular algorithms like PCA and t-SNE.
- **Matplotlib** is used to create visualizations such as scatter plots.
- **Seaborn** is useful for creating advanced plots, including multiple subplots.

These libraries are not part of Python's standard library. To install them, use the following command:

```
Pip install numpy pandas scikit-learn matplotlib seaborn
```

To improve performance, I set the number of CPU cores to 10, which matches the number of cores available on my laptop.

```
mnist = fetch_openml('mnist_784')
X = mnist['data'] / 255.0
y = mnist['target']

df = pd.DataFrame(X)
print(df.head())
```

Load the MNIST dataset, which contains handwritten digits represented by 784 features along with their corresponding labels. The digits have been size-normalized and centered within fixed-size images. Due to the anti-aliasing technique used during normalization, the images contain varying gray levels. Since the images are grayscale, dividing the pixel values by 255 helps reduce computational complexity.

```
feat_cols = df.columns
df['y'] = y
df['y'] = df['y'].astype(int)
df['label'] = df['y'].apply(lambda i: str(i))
print(df.head())
```

Extract column names for further use and convert labeled result to integer for plotting.

```
np.random.seed(60)
rndperm = np.random.permutation(df.shape[0])
```

Set randomization for selecting random numbers to prove how it works.

```
plt.gray()
fig = plt.figure(figsize=(16, 7))
for i in range(0, 15):
    ax = fig.add_subplot(3, 5, i + 1, title="Digit: {}".format(str(df.loc[rndperm[i], 'label'])))
    ax.matshow(df.loc[rndperm[i], feat_cols].values.reshape((28, 28)).astype(float))

fig, ax = plt.subplots()
ax.matshow(df.loc[1, feat_cols].values.reshape((28, 28)).astype(float))
ax.set_title('A single plot')
plt.show()
```

Set plot as gray and show up first 15 digits. Also, display one digit and make it bigger for review.

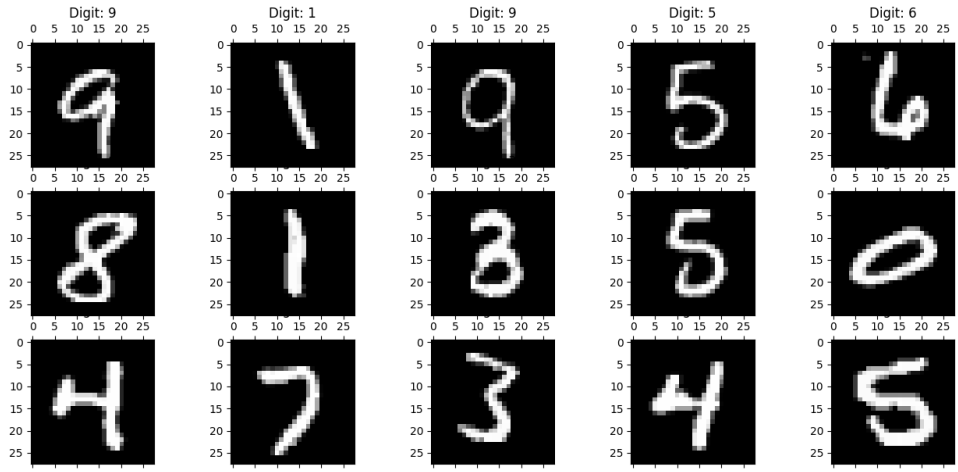

Figure 1: display what handwritten digits look like.

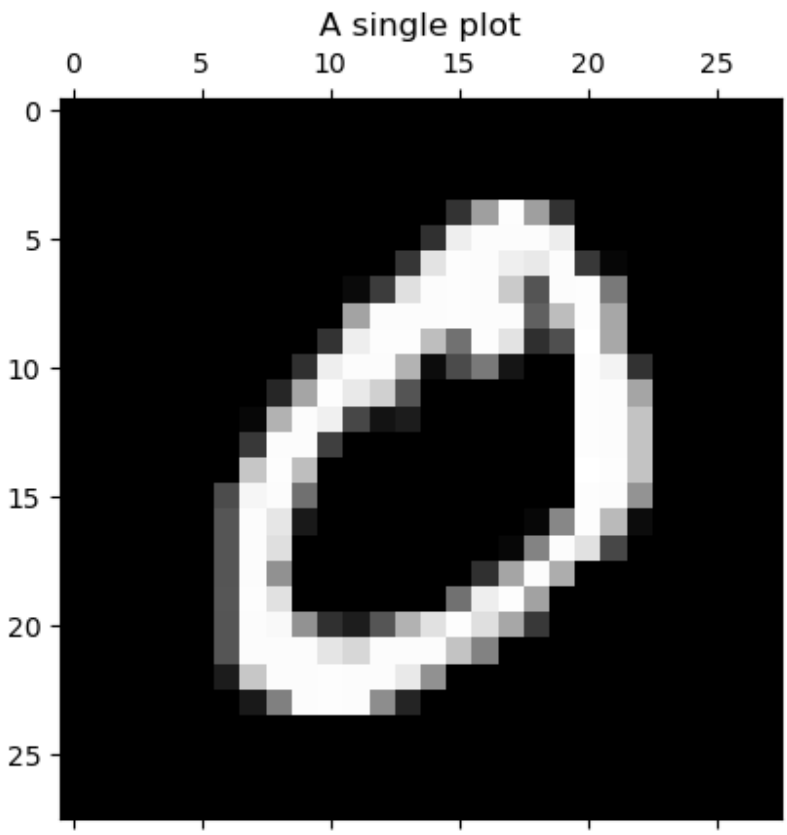

Figure 2: display the single handwritten digit.

```
pca = PCA(n_components=3)
pca_result = pca.fit_transform(df[feat_cols].values)
df['pca-one'] = pca_result[:, 0]
df['pca-two'] = pca_result[:, 1]
df['pca-three'] = pca_result[:, 2]

print(df.head())

print('Explained variation per principal component: {}'.format(pca.explained_variance_ratio_))

# number of components
n_pcs = pca.components_.shape[0]

# get the index of the most important feature on EACH component i.e. largest absolute value
# using LIST COMPREHENSION HERE
most_important = [np.abs(pca.components_[i]).argmax() for i in range(n_pcs)]
most_important_names = [feat_cols[most_important[i]] for i in range(n_pcs)]
print(most_important_names)
```

First, use PCA approach to select the 3 most important features and calculated variations.

```
Explained variation per principal component: [0.09746116 0.07155445 0.06149531]
['pixel524', 'pixel157', 'pixel633']
```

Figure 3: display most important features and calculated variations.

```
plt.figure(figsize=(16, 10))
sns.scatterplot(
    x="pca-one", y="pca-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df.loc[rndperm, :],
    legend="full",
    alpha=0.3
)
```

Plot 2 dimensional using PCA one and PCA two features.

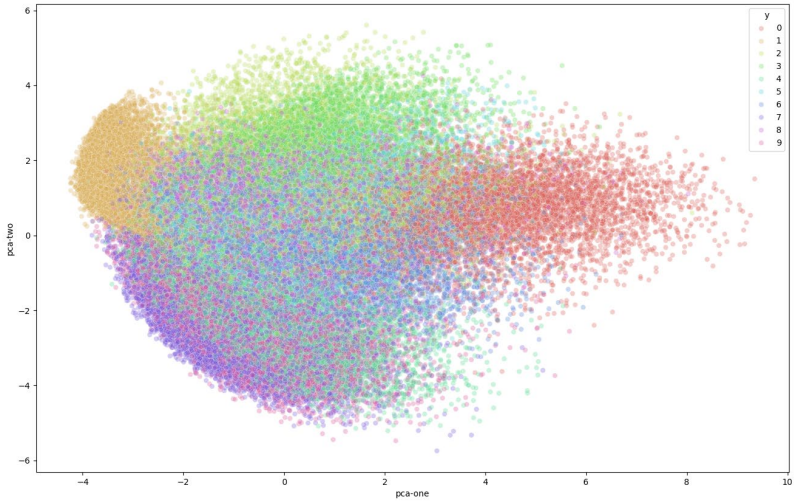

Figure 4: display all digits using 2 most important features. They are overlapping with each other.

```
fig = plt.figure(figsize=(16, 10))
ax = fig.add_subplot(projection='3d')
ax.scatter(
    xs=df.loc[rndperm, :]["pca-one"],
    ys=df.loc[rndperm, :]["pca-two"],
    zs=df.loc[rndperm, :]["pca-three"],
    c=df.loc[rndperm, :]["y"],
    cmap='tab10'
)
ax.set_xlabel('pca-one')
ax.set_ylabel('pca-two')
ax.set_zlabel('pca-three')
```

Plot 3 dimensional using PCA one, PCA two and PCA three features.

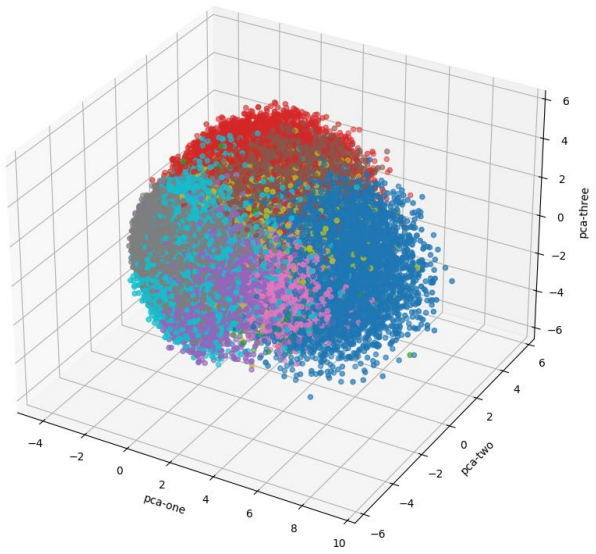

Figure 5: display all digits using 3 most important features. They are overlapping with each other.

```
N = 10000
df_subset = df.loc[rndperm[:N], :].copy()
data_subset = df_subset[feat_cols].values
pca = PCA(n_components=3)
pca_result = pca.fit_transform(data_subset)

df_subset['pca-one'] = pca_result[:, 0]
df_subset['pca-two'] = pca_result[:, 1]
df_subset['pca-three'] = pca_result[:, 2]

print('Explained variation per principal component: {}'.format(pca.explained_variance_ratio_))
```

Randomly select 10000 digits to run comparison with PAC and t-SNE.

```
Explained variation per principal component: [0.09746116 0.07155445 0.06149531]
```

Figure 6: display calculated variations.

```
time_start = time.time()
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(data_subset)

print('t-SNE done! Time elapsed: {} seconds'.format(time.time() - time_start))
```

Define parameters for t-SNE and run it. Set n_components to 2 for plotting 2 dimension diagram. Verbose helps display how the algorithm works. Perplexity helps balance attention between local and global aspects. Set n_iter to 300 which is the maximum number of iterations for the optimization.

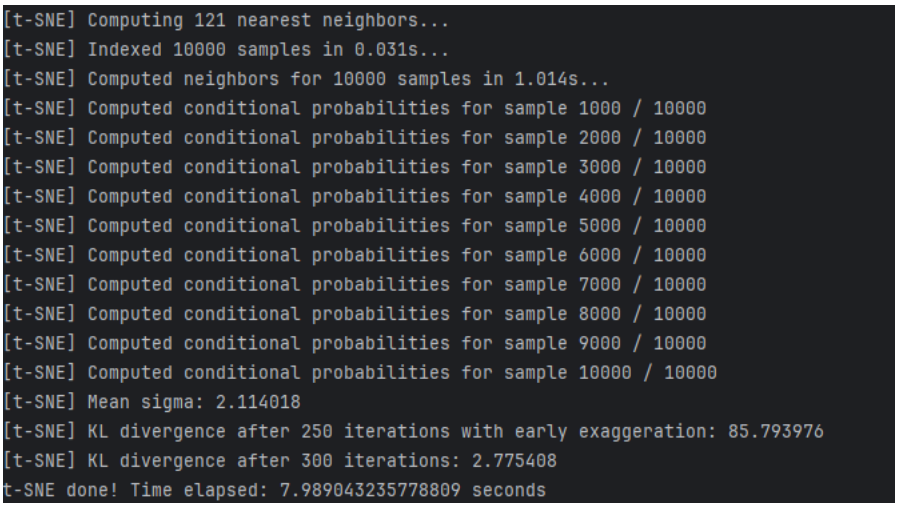

Figure 7: display log information on t-SNE

```
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
)
```

Plot a 2 dimensional to display the resulted dataset for t-SNE

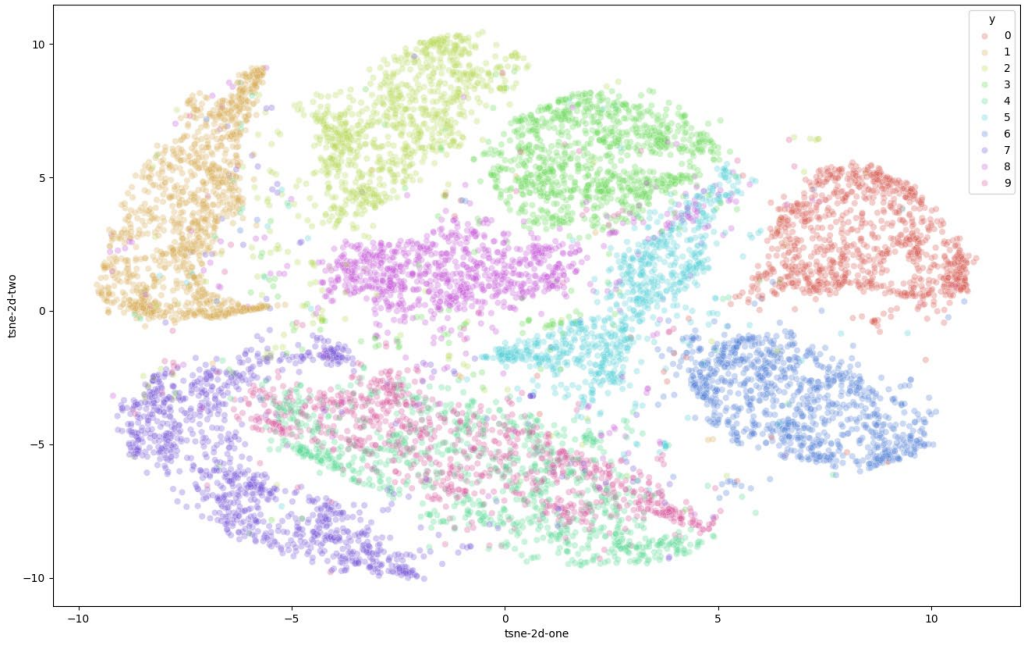

Figure 8: display different clusters of digits.

```
plt.figure(figsize=(16, 7))
ax1 = plt.subplot(1, 2, 1)
sns.scatterplot(
    x="pca-one", y="pca-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3,
    ax=ax1
)

ax2 = plt.subplot(1, 2, 2)
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,
    ax=ax2
)
```

Plot 2 dimensional in PCA and t-SNE and set them next to each other.

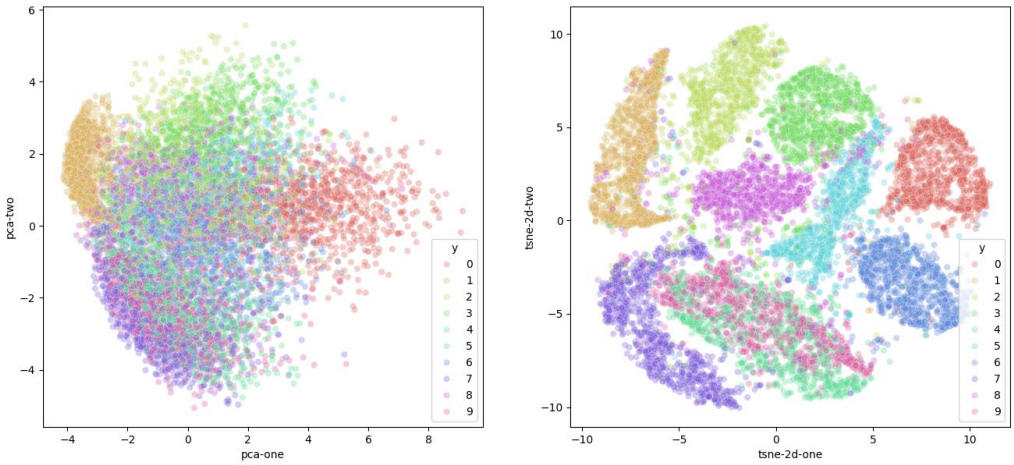

Figure 9: Left is using PCA and right is TSNE.

```
pca_50 = PCA(n_components=50)
pca_result_50 = pca_50.fit_transform(data_subset)

print('Cumulative explained variation for 50 principal components: {}'.format(np.sum(pca_50.explained_variance_ratio_)))
time_start = time.time()

tsne = TSNE(n_components=2, verbose=0, perplexity=40, n_iter=300)
tsne_pca_results = tsne.fit_transform(pca_result_50)

print('t-SNE done! Time elapsed: {} seconds'.format(time.time() - time_start))
```

To improve performance, I have tried to combine PCA and t-SNE. PCA helps select the most important features and t-SNE helps visualize the optimized dataset.

```
f_subset['tsne-pca50-one'] = tsne_pca_results[:, 0]
df_subset['tsne-pca50-two'] = tsne_pca_results[:, 1]
plt.figure(figsize=(16, 4))
ax1 = plt.subplot(1, 3, 1)
sns.scatterplot(
    x="pca-one", y="pca-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3,
    ax=ax1
)

ax2 = plt.subplot(1, 3, 2)
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,
    ax=ax2
)

ax3 = plt.subplot(1, 3, 3)
sns.scatterplot(
```

```
    x="tsne-pca50-one", y="tsne-pca50-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3,
    ax=ax3
)

plt.show()
```

Plot three 2 dimensional and set next to each other. The first is using PCA with 2 most important features. The second is using t-SNE with 2 most important features. The third is 2 most important features chosen by PCA with the 50 most important features.

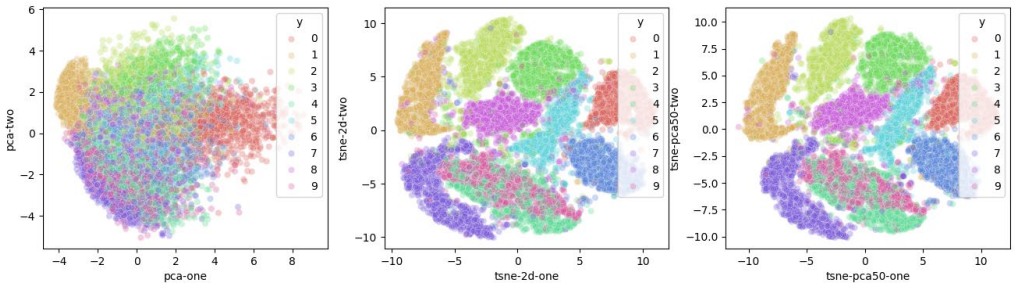

Figure 10: display three plots. It is clear the first plot has much more clear boundaries in each cluster.

DISCUSSION

t-SNE excels at visualizing local structure and forming visually distinct clusters. However, it struggles with global structure preservation and is sensitive to parameter settings. Additionally, it is computationally intensive and non-deterministic without a fixed random seed.

CONCLUSION

t-SNE is a powerful tool for visualizing high-dimensional data, especially when local structure is of primary interest. Despite its limitations, its intuitive output makes it a valuable technique in exploratory data analysis. The last figure clearly shows that the hybrid approach using PCA and t-SNE is better than t-SNE for data visualization and exploration.

References

[1] TSNE https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[2] Hinton, Geoffrey; Roweis, Sam (January 2002). Stochastic neighbor embedding (PDF).

[3] van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of Machine Learning Research, 9(Nov), 2579–2605.

[4] t-distributed stochastic neighbor embedding https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

[5] handwritten digits description
https://www.openml.org/search?type=data&sort=runs&id=554&status=active

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jun Yang
Avidity Biosciences
jun.yang@aviditybio.com