7 t-Distributed stochastic neighbor embedding

Mohammad Akhavan Anvari[a], Dara Rahmati[b], and Sunil Kumar[c]
[a]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran [b]Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran [c]Department of Mathematics, National Institute of Technology, Jamshedpur, Jharkhand, India

7.1 Introduction to t-SNE

7.1.1 What is t-SNE?

t-SNE is an algorithm primarily used in machine learning that reduces the dimensionality of high-dimensional, complex datasets that was developed by Maaten and Hinton [1]. Dimension reduction is an essential technique nowadays as many real-world datasets contain numerous features, making it challenging to visualize and understand the relationships between the data points [6]. t-SNE is a nonlinear dimensionality reduction algorithm that maps the high-dimensional data points to a lower-dimensional space while preserving the relationships between the data points as much as possible. t-SNE is suitable for assessing and visualizing complex and nonlinear relationships in the data, which does not rely on a linear relationship between the data points, unlike linear methods such as PCA. Instead, it models the local associations between the data points, enabling the discovery of nonlinear patterns and structures. This makes it an effective mechanism for examining and comprehending complex datasets in many fields like biology, computer vision, and natural language processing.

7.1.2 Why is t-SNE useful?

t-SNE is advantageous since it permits the display and investigation of high-dimensional datasets. In numerous real-world applications, the data we work with possess multiple qualities, making it challenging to comprehend the interrelationships between the data points. t-SNE provides a solution by reducing the dimensionality of the data, shifting it from a high-dimensional space to a lower-dimensional one while preserving as many of the relationships between the data points as possible. Its ability to show nonlinear correlations in the data, which are generally difficult to observe in high-dimensional space, is one of its key advantages. This renders it an effective tool for exploratory data analysis and pattern recognition. The capacity of t-SNE to generate highly interpretable data visualizations is an added advantage. The application illustrates the connections between the data points, allowing us to spot clusters of similar data points and their linkages. This facilitates the collection of insights into the data structure that may not be readily apparent in a high-dimensional setting.

7.1.3 Prerequisite

7.1.3.1 Gaussian distribution

In statistics, continuous random variables are often represented by the Gaussian distribution, also known as the normal distribution. The German mathematician Carl Friedrich Gauss was primarily responsible for its development. The two most notable features of the Gaussian distribution are the mean, which represents the central tendency of the data, also known as the highest point of the distribution, and the standard deviation, which represents the spread or dispersion of the data. Entropy quantifies the degree of disorder in a statistical distribution. An essential property of the Gaussian distribution is that it is a maximum entropy distribution, meaning it has the maximum uncertainty or randomness for a given mean and standard deviation. This property is derived from the Gaussian distribution having the most extensive spread for a given variance among all possible probability distributions with the same mean and variance.

The use of Gaussian distributions in t-SNE is essential for several reasons:

- **Nonlinearity**: The t-SNE algorithm is a nonlinear technique for reducing the dimensionality of data. This approach can capture intricate relationships between data points that cannot be achieved by linear methods such as PCA. The utilization of Gaussian distributions in t-SNE enables the modeling of pairwise similarities between points in a nonlinear manner, thereby facilitating the capture of nonlinear relationships.
- **Smoothness**: The Gaussian distribution is characterized by a continuous probability density function, indicating its smoothness. The smoothness attribute plays a crucial role in maintaining the local structure of data in the lower-dimensional representation by ensuring smooth and continuous pairwise similarities between data points.
- **Robustness**: The Gaussian distribution is a widely studied probability distribution with numerous established statistical properties. Its suitability for modeling pairwise similarities between data points makes it a robust choice, contributing to the stability and reliability of the t-SNE algorithm.
- **Parameterization**: The Gaussian distribution is characterized by two parameters, namely the mean, and variance, which can be manipulated to modulate the distribution's morphology. The optimization of the visualization can be achieved by adjusting the parameters of the Gaussian kernel, which enables the fine-tuning of the t-SNE algorithm to suit particular datasets.

7.1.3.2 Student's t-distribution

Student's t-distribution, also known as the t-distribution, is a probability distribution with heavier tails than the Gaussian or normal distribution. It is frequently employed in statistical inference, especially for verifying hypotheses and estimating confidence intervals when the sample size is limited. The t-distribution is defined by its degrees of freedom (df), which determine the distribution's appearance. Similar to the Gaussian distribution, but with larger tails, the distribution is symmetric and bell-shaped but with thicker tails. With an increase in degrees of freedom, the t-distribution approaches the Gaussian distribution. The distribution's tails are the primary distinction between the Gaussian distribution and the t-distribution. The tails of the Gaussian distribution are narrower, which attributes a lower probability to extreme values or data outliers. In contrast, the t-distribution has fatter tails, which means it allocates a greater probability to extreme values or data deviations. Consequently, the t-distribution is more resistant to outliers than the Gaussian distribution. In statistical inference, when the sample size is small and the data contains outliers, the t-distribution can provide more accurate estimates of the population parameters, such as the mean and standard deviation, which can help to preserve the local structure of the data in the lower-dimensional space.

7.1.3.3 Gradient descent algorithm

Gradient descent [4], also known as steepest descent in older notation, is a classical iterative first-order optimization technique utilized for determining the optimal solution on differentiable functions. The present algorithm employs the gradient of a function to compute the derivatives of a function at a specific point, resulting in the contemporary nomenclature of the function's slope at said point. To determine the slope of a function at a specific point x, formula (7.1) is utilized, as demonstrated below:

$$Slope = \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{7.1}$$

In this formula, f is a continuous function, and Δx is the rate of the change in x; to seek precision, we consider it as $epsilon$, the tiniest value.

The gradient descent algorithm employs a stochastic point on the function to explore a local minimum of a claimed function. The algorithm utilizes the specified point to compute the gradient value, thereby facilitating a one-step descent to the function's lower state. Subsequently, the algorithm will proceed to revisit the point where it had previously executed, thereby assuming a novel position. Executing numerous iterations of this process results in reaching either a local or a global minimum of the function.

The formulation of this operation can be defined as below:

$$\theta_{t+1} = \theta_t - D_x f(x). \tag{7.2}$$

In this formulation, f is the target function we want to find a minimum for, D_x is the gradient operator in point x, and θ is a point on the function in steps t and $t + 1$.

Optimizing the lower-dimensional representation of data in t-SNE is achieved through gradient descent, which aims to minimize the Kullback–Leibler divergence or in abbreviation the KL divergence [5] between the high-dimensional and lower-dimensional probability distributions. The modifications to the lower-dimensional depiction are executed by following the negative gradient of the cost function. This results in the lower-dimensional points being brought closer to their corresponding neighbors in the high-dimensional space, which will be elaborated on in subsequent sections.

7.1.4 Applications of t-SNE

As mentioned in early sections, t-SNE primarily uses dimension reduction, especially for nonlinear relational data; however, in some cases, scientists use the t-SNE algorithm for clustering. This algorithm is an adaptable method such that its applications could be used in many different industries, therefore most common applications of t-SNE are the following:

- t-SNE can be used to visualize high-dimensional word vectors in natural language processing (NLP) applications such as document categorization and sentiment analysis, which help scientists to understand relational links between words and sentences [2].
- t-SNE can be used to visualize the activation functions of deep neural networks to comprehend how they process picture data in image recognition.
- t-SNE can be used to show gene manifestation data in a way that reveals patterns and correlations between genes and biological samples in the context of gene expression analysis [3].
- Recommender systems like content-based recommendation, such as movies or music, t-SNE, can be used to show the relationships between users' behavior and candidate content.

These are only selected applications of t-SNE. The algorithm's capacity to analyze and investigate high-dimensional data renders it a significant aid in various disciplines.

7.2 Understanding the t-SNE algorithm

The t-SNE algorithm aims to map high-dimensional data points to a low-dimensional space while maintaining their pairwise similarity relation. In other words, it attempts to embed each data point in a lower-dimensional space so that related points remain close together in the low-dimensional space. At the same time, a more significant distance separates dissimilar points. The t-SNE algorithm works in two main stages: First, it builds a probability distribution over pairs of high-dimensional items, then constructs a corresponding distribution over pairs of low-dimensional objects and minimizes the Kullback–Leibler divergence between the two distributions. The stages are described in full below.

Step 1. Constructing the high-dimensional probability distribution:

In the first step of the t-SNE algorithm, a probability distribution is constructed over pairs of high-dimensional objects (e.g., data points). The goal of this step is to create a probability distribution that captures the similarity between pairs of objects, with similar objects having a higher probability of being selected. To achieve this, t-SNE constructs a Gaussian probability distribution around each data point, with the mean of the selected point and the variance of this determined by a user-defined parameter called the perplexity. The perplexity controls the balance between preserving global and local structure and is typically set between 5 and 50. The probability that point j is similar to point i is given by conditional probability as below:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i} \exp(-\|x_i - x_k\|^2/2\sigma_i^2)}, \tag{7.3}$$

where x_i and x_j are the high-dimensional feature vectors corresponding to data points i and j, $\|.\|$ denotes the Euclidean distance between the two vectors, and sigma is the variance of the Gaussian distribution, which is set such that the perplexity is satisfied. The denominator is a normalization constant that ensures that the probabilities sum to one over all pairs of points.

Step 2. Constructing the low-dimensional probability distribution:

In the second phase of the t-SNE method, a probability distribution over pairs of low-dimensional objects is produced (e.g., the embedded points in a 2D or 3D space). Note that this projection in the first step could be a random variable or we could use other dimension reduction algorithms like PCA as a starting point. This step's objective is to generate a probability distribution representing the similarity between pairs of embedded points, with related points having a greater chance of being picked. To do this, t-SNE builds a Student-t probability distribution around each embedded point, with the variance of the distribution based on the number of degrees of freedom (df). The probability that point j is similar to point i in the embedded space is given by:

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k\neq i} \exp(-\|y_i - y_k\|^2)}, \tag{7.4}$$

where y_i and y_j are the low-dimensional feature vectors corresponding to embedded points i and j. The denominator is a normalization constant that ensures that the probabilities sum to one over all pairs of points.

Step 3. Minimizing the Kullback–Leibler divergence:

In the last phase of the t-SNE technique, the Kullback–Leibler divergence between the two probability distributions created in stages 1 and 2 is minimized using the gradient descent algorithm. At each iteration, the algorithm computes the gradient of the divergence with respect to the positions of the data points in the low-dimensional space and updates their positions accordingly; stage 3. In this way, gradient descent helps t-SNE find a good representation of the data in lower dimensions. In the following sections, first, we will discuss the perplexity parameter and then the Kullback–Leibler divergence, which measures the difference between two probability distributions.

7.2.1 The t-SNE perplexity parameter

In t-SNE, the perplexity hyperparameter regulates the balance between local and global structure in the low-dimensional embedding. It measures the number of suitable nearest neighbors used to generate the high-dimensional similarity matrix.

The formula to compute perplexity is:

$$Perp(P_i) = 2^{H(P_i)}, \tag{7.5}$$

where P_i is the conditional probability distribution over the nearest neighbors of point i in the high-dimensional space, and $H(P_i)$ is the Shannon entropy of P_i, defined as:

$$H(P_i) = -\sum_j P_{j|i} \log_2 P_{j|i}. \tag{7.6}$$

The perplexity value controls the number of nearest neighbors to consider in each local neighborhood and is commonly set between 5 and 50 but consider that there is no optimal perplexity for all datasets. When the perplexity is low, the method concentrates more on the local data structure, and the resultant embedding preserves the fine-grained features of the data points inside each neighborhood. However, this may be achieved at the cost of not capturing the global data structure and may result in deformed clusters. When the perplexity is large, the method concentrates more on the global structure of the data, and the resultant embedding preserves the data's overall similarity structure. This may come at the expense of not obtaining the granular features of the data pieces inside each area.

Consider an example to understand how perplexity affects the behavior of t-SNE. Suppose we have a dataset of 1000 images of handwritten digits [13] (from 0 to 9), each represented as a 784-dimensional vector. We want to visualize the dataset in 2-dimensional space using t-SNE. We construct a high-dimensional similarity matrix based on the Euclidean distance between the data points. After that, we set the perplexity to 30 and then execute t-SNE to obtain a 2-dimensional embedding. The resulting embedding in Fig. 7.1 shows clear clusters of digits, with similar digits grouped (e.g., 0s with other 0s, 1s with other 1s, etc.). Next, we decrease the perplexity to a value of 5 and rerun t-SNE. The resulting embedding of Fig. 7.2 shows more fine-grained details, with each digit spread into a different cloud. However, some clusters could be more well-defined, and some digits are mixed in the embedding. Finally, we increase the perplexity to a value of 50 and rerun t-SNE. The results in Fig. 7.3 show a more precise global structure, with the digits separated into distinct clusters. However, some fine-grained details are lost, and some clusters must be more well-separated.

In conclusion, the perplexity parameter in t-SNE controls the balance between local and global structure in the low-dimensional embedding, and the optimal value depends on the specific characteristics of the dataset and the task at hand.

7.2.2 The t-SNE objective function

Kullback–Leibler divergence is a measure of the difference between two probability distributions, often denoted as P and Q. It measures how much information is lost when we use Q to approximate P. Mathematically, KL divergence is defined as follows:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}, \tag{7.7}$$

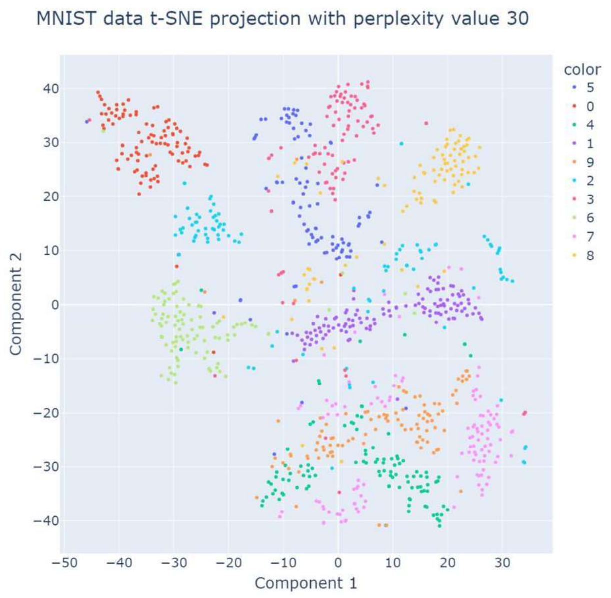

FIGURE 7.1 Illustration of the MNIST dataset in a 2D plot using t-SNE algorithm with the perplexity parameter set to 30.

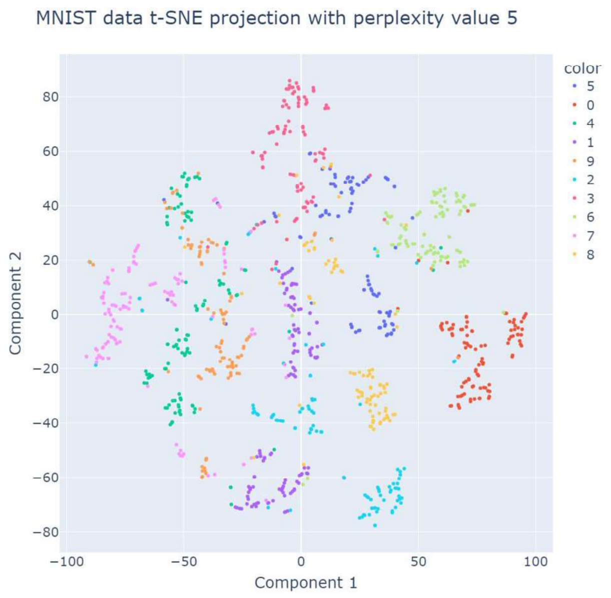

FIGURE 7.2 Illustration of the MNIST dataset in a 2D plot using t-SNE algorithm with the perplexity parameter set to 5.

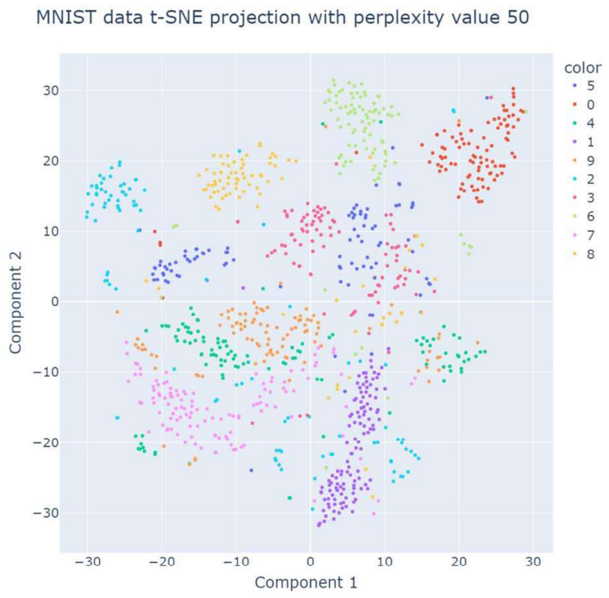

FIGURE 7.3 Illustration of the MNIST dataset in a 2D plot using t-SNE algorithm with the perplexity parameter set to 50.

where P and Q are two probability distributions over the same event space, and i indexes the events in the space. The logarithm in the formula ensures that the KL divergence is always non-negative, with a value of zero when P and Q are identical. KL divergence is not symmetric, meaning that $D_{KL}(P||Q)$ and $D_{KL}(Q||P)$ can have different values.

Let us consider an example to understand KL divergence better. Suppose we have a coin with an unknown bias. We toss the coin ten times and observe seven heads and three tails. We want to know the probability of getting heads or tails in the next toss. We can model the coin toss using two probability distributions: the accurate distribution, P, which represents the actual bias of the coin, and the estimated distribution, Q, which we compute based on the observed data. Let us say we believe that the coin is fair, so we set Q to be a uniform distribution:

$$\begin{aligned} P(H) = p, P(T) = 1 - p, \\ Q(H) = Q(T) = 0.5. \end{aligned} \tag{7.8}$$

We can now calculate the KL divergence between P and Q to measure how different our estimated distribution is from the actual distribution. Using the formula above, we obtain:

$$\begin{aligned} D_{KL}(P||Q) &= P(H) \times \log \frac{P(H)}{Q(H)} + P(T) \times \log \frac{P(T)}{Q(T)} \\ &= p \times \log(2p) + (1-p) \times \log(2(1-p)). \end{aligned} \tag{7.9}$$

If we assume that $p = 0.7$, we can calculate the value of $D_{KL}(P||Q)$ to be approximately 0.24. This tells us that the estimated distribution Q could better approximate the true distribution P. In other words, we lose 0.24 bits of information when we use Q to model the coin toss, compared to using the true distribution P. KL divergence is used in many areas of machine learning, including information theory, signal processing, and statistics. It is often used as a loss function in training generative models, where the goal is to approximate a complex, high-dimensional probability distribution with a simpler, low-dimensional distribution.

In t-SNE, the high-dimensional probability distribution P and the low-dimensional probability distribution Q are constructed based on the pairwise similarity of data points. The goal is to find a mapping of the data points from the high-dimensional space to the low-dimensional space that preserves their pairwise similarities. For this, t-SNE minimizes the KL divergence between P and Q by adjusting the embedding of each data point until the two probability distributions are as close as possible. The optimization is performed iteratively using gradient descent, and the embedding is adjusted to minimize the cost function:

$$C = \sum_i KL(P_{i|j}||Q_{i|j}) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{i|j} \log \frac{p_{i|j}}{q_{i|j}}, \tag{7.10}$$

where $P(i, j)$ is the probability that point i and point j are similar in the high-dimensional space, and $Q(i, j)$ is the probability that the embedded points i and j are similar in the low-dimensional space.

The gradient of the cost function with respect to the embedding of point i in the low-dimensional space is given by:

$$\frac{\partial C}{\partial y_i} = 4\sum_j (p_{i|j} - q_{i|j})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}, \tag{7.11}$$

where y_i is the low-dimensional embedding of point i, y_j is the low-dimensional embedding of point j, and $||.||$ denotes the Euclidean distance between the two embeddings.

The gradient is used to update the embedding of each data point in the low-dimensional space during each iteration of the optimization process. By minimizing the cost function, t-SNE produces a low-dimensional embedding of the high-dimensional data, preserving the pairwise similarities between the data points as much as possible.

7.2.3 The t-SNE learning rate

The learning rate in t-SNE controls the step size of the gradient descent algorithm used to optimize the low-dimensional embedding. It determines how much the embedding changes in each iteration of the algorithm. The formula to update the low-dimensional embedding at each iteration is:

$$y_i(t+1) = y_i(t) + \eta(t) \times d_i(t), \tag{7.12}$$

where $y_i(t)$ is the 2-dimensional embedding of data point i at iteration t, $\eta(t)$ is the learning rate at iteration t, and $d_i(t)$ is the gradient of the cost function with respect to $y_i(t)$. The learning rate is usually initialized to a high value and gradually reduced throughout the optimization. This is done to prevent the algorithm from becoming stuck in local optima and to allow it to explore different regions of the low-dimensional space. One common strategy for reducing the learning rate is to use a power-law decay, such as:

$$\eta(t) = \frac{\eta_0}{(1 + \frac{t}{a})}, \tag{7.13}$$

where η_0 is the initial learning rate, t is the current iteration, and a is a constant that controls the rate of decay. The learning rate also interacts with the gradient magnitudes in the high-dimensional space. When the gradient magnitudes are significant, a high learning rate can lead to unstable behavior and oscillations in the embedding. To prevent this, t-SNE uses early exaggeration, which multiplies the pairwise similarities in the high-dimensional space by a factor of 4 in the early iterations of the algorithm. This exaggerates the distances between the data points and creates more significant gradients, which allows the low-dimensional embedding to separate more quickly. To illustrate how the learning rate influences the behavior of t-SNE, consider the following example. Imagine we have a collection of 1000 photographs of handwritten numbers (0–9) where each image is recorded as a 784-dimensional vector. Using t-SNE, we wish to depict the dataset in two dimensions. The learning rate is initialized at 1, and 10 iterations of t-SNE are performed. The resultant embedding in Fig. 7.4 displays a few clusters of numbers, but most of the digits are still jumbled. Then, we decreased the learning rate to 0.5 and ran t-SNE 10 times more. The result in Fig. 7.5 displays groups of identical digits that are more clearly defined. Lastly, we decrease the learning rate to 0.1 and run t-SNE for ten additional iterations. Fig. 7.6 demonstrates even more distinct clusters, with minimal overlap between the digits.

The learning rate parameter governs the step size of the gradient descent process used to optimize the low-dimensional embedding. The unique properties of the dataset and the objective at hand determine the ideal value. A high learning rate can lead to unstable behavior, whereas a low learning rate can result in slow convergence and being stuck in local optima.

7.2.4 Implementing t-SNE in practice

To achieve a better understanding of the t-SNE algorithm we are going to implement it from scratch. First, we need to understand the pseudocode of the algorithm so we write exactly the pseudocode from the original paper of t-SNE:

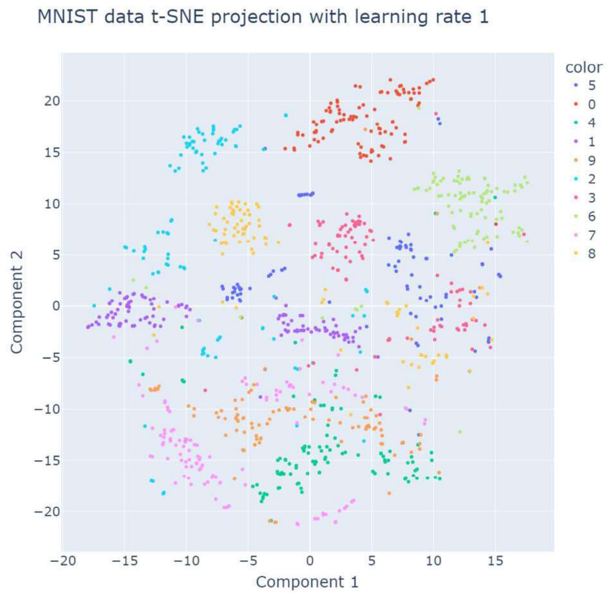

FIGURE 7.4 Illustration of the MNIST dataset in a 2D plot using t-SNE algorithm with the perplexity parameter set to 30 and a learning rate of 1.

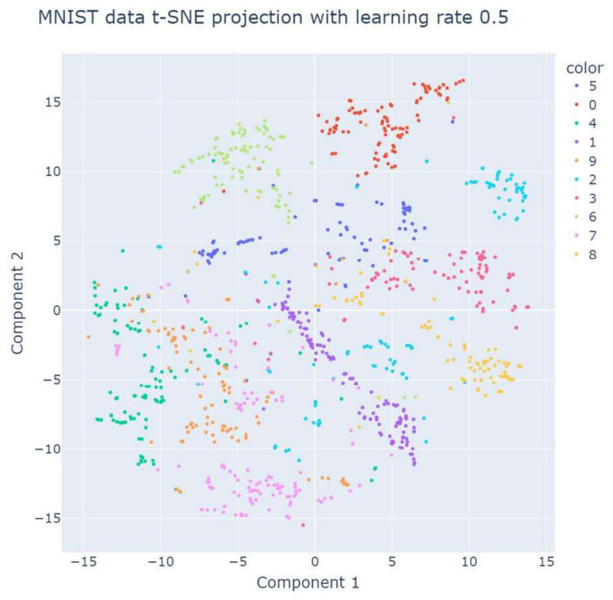

FIGURE 7.5 Illustration of the MNIST dataset in a 2D plot using the t-SNE algorithm with the perplexity parameter set to 30 and a learning rate of 0.5.

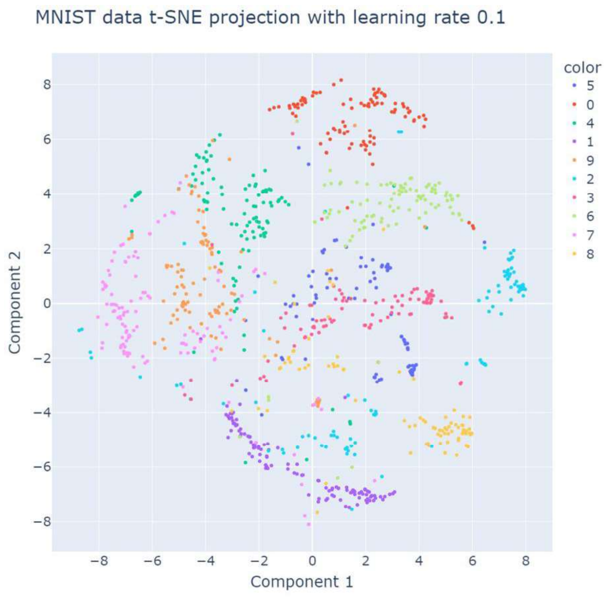

FIGURE 7.6 Illustration of the MNIST dataset in a 2D plot using the t-SNE algorithm with the perplexity parameter set to 30 and a learning rate of 0.1.

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

```
Data: dataset $\chi = \{x_1, x_2, ..., x_n\}$,
cost function parameters: perplexity *Perp*,
optimization parameters: number of iterations $T$, learning rate $\eta$, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
begin
compute pairwise affinities $p_{j|i}$ with perplexity *Perp* (using Eq. (7.3))
set $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$
sample initial solution $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
for $t = 1$ to $T$ do
    compute low-dimensional affinities $q_{ij}$ (using Eq. (7.12))
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Eq. (7.11))
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{t-1} + \eta\frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)})$
end for
```

Step 1: Import necessary libraries: We will need to import NumPy for numerical computations, and matplotlib for data visualization.

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

Step 2: Preparing the data: t-SNE is a dimensionality reduction algorithm, so you will need to start with high-dimensional data. In this example, let us assume you have a dataset with n samples and m features.

```
X = # Your data with shape (n, m).
```

Step 3: Compute pairwise similarity: t-SNE operates on pairwise similarity between samples. Compute the pairwise Euclidean distance between all samples in your dataset.

```
from sklearn.metrics.pairwise import euclidean_distances

distances = euclidean_distances(X)
```

Step 4: Compute similarity matrix: Transform the pairwise distance matrix into a similarity matrix using the Gaussian kernel.

```
def gaussian_kernel(distances, sigma=1.0):
    return np.exp(-(distances ** 2) / (2 * (sigma ** 2)))

similarities = gaussian_kernel(distances)
```

Step 5: Initialize the embedding: Initialize a low-dimensional embedding for your data. This can be done randomly or using another method.

```
# the desired dimensionality of the embedding
embedding_dim = 2
# initialize embedding randomly
Y = np.random.randn(n, embedding_dim)
```

Step 6: Define the cost function: The t-SNE cost function is defined as the Kullback–Leibler (KL) divergence between the high-dimensional and low-dimensional distributions.

```
def kl_divergence(P, Q):
    return np.sum(P * np.log(P / Q))

def compute_joint_probabilities(similarities, perplexity):
    # Compute pairwise conditional probability
    # using Gaussian kernel
    sum_s = np.sum(similarities)
    P = similarities / sum_s
    P = np.maximum(P, 1e-12) # to prevent division by zero
    P *= perplexity
    P = np.maximum(P, 1e-12) # to prevent division by zero
    P = P / np.sum(P)
    return P

def compute_gradient(P, Q, Y):
    # Compute gradient of cost function with respect to Y
    pq_diff = P - Q
    for i in range(n):
        grad[i, :] = np.sum(np.tile(pq_diff[:, i] * Y[i, :],
                             (n, 1)), axis=0)
    return grad
```

Step 7: Optimize the cost function: Use gradient descent to optimize the cost function with respect to the low-dimensional embedding.

```
def optimize_embedding(Y, P, learning_rate=200,
                       n_iter=1000):
    # Optimize the embedding using gradient descent
    Q = 1 / (1 + euclidean_distances(Y) ** 2)
    np.fill_diagonal(Q, 0)
    for i in range(n_iter):
        grad = compute_gradient(P, Q, Y)
        Y = Y - learning_rate * grad
        Q = 1 / (1 + euclidean_distances(Y) ** 2)
        np.fill_diagonal(Q, 0)
    return Y

P = compute_joint_probabilities(similarities,
                                perplexity=30)
Y = optimize_embedding(Y, P)
```

Step 8: Visualize the low-dimensional embedding: Finally, you can visualize the low-dimensional embedding using a scatter plot.

```
plt.scatter(Y[:, 0], Y[:, 1], c=color_labels)
```

For ease of usage, this algorithm is implemented in Python. We can use Scikit-Learn to feed data into the algorithm and create a plot visualization for better understanding. We will use this package and run the t-SNE algorithm on the Iris dataset.

Step 1: Importing: Scikit-Learn and other useful packages.

```
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
from sklearn.datasets import load_iris
import plotly.express as px
```

Step 2: Loading: The Iris dataset.

```
iris = load_iris()
x = iris.data
y = iris.target
```

Step 3: Initializing: t-SNE as an object with *n components* for selecting the dimension of output data. (For this example we use default values for hyperparameters of the t-SNE algorithm.)

```
tsne = TSNE(n_components=2, verbose=1, random_state=123)
x_transformed = tsne.fit_transform(x)
```

Step 4: Preparing output data for visualization.

```
df = pd.DataFrame()
df["y"] = y
df["component_1"] = x_transformed[:,0]
df["component_2"] = x_transformed[:,1]
```

Step 5: Visualizing output data.

```
px.scatter(data_frame=df, x="component_1",
y="component_2",
color=df.y.tolist(),
       labels={
                 "component_1": "Component 1",
                 "component_2": "Component 2",
              },
        title="Iris dataset T-SNE projection",
        width=1024, height=1024)
```

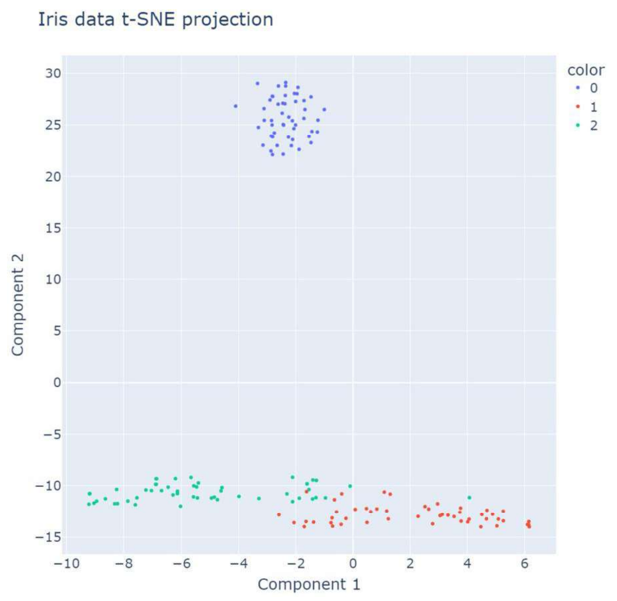

FIGURE 7.7 Illustration of the Iris dataset in a 2D plot using the t-SNE algorithm with default parameters.

Fig. 7.7 illustrates the result of the algorithm, and we can see that each label is shown in its own individual color. It is essential to point out that one of the labels is located a considerable distance from the others. In contrast, the other two labels have a similar structure but vary in the values of their component parts.

In the same way as before, we will try to run the algorithm on the much more difficult MNIST dataset, which is significantly more complicated than the Iris dataset. Parameter setting will be the same as the default value of the Scikit-Learn.

In Fig. 7.8, we see that each label, a digit number in this dataset, is differentiated by color. However, as is clear, some data from two or three other digits collided together. The reason for that is, first, the structure of those digits is similar, and the algorithm could not understand the difference. Secondly, the dimension of the MNIST dataset; 784 features is significantly higher than that of the Iris dataset, which is 4. Due to this, the algorithm struggles to distinguish some similar digits. Continuing this, we will explore our alternatives to resolve this issue.

7.3 Visualizing high-dimensional data with t-SNE

It is essential to note that t-SNE is unsuitable for all datasets and should be cautiously selected. For instance, it may not function optimally with highly structured or noisy datasets, and in this case, the algorithm will be computationally costly, particularly for massive datasets. Nevertheless, with correct parameter adjustment and preparation, t-SNE may be an effective visualization tool for complicated datasets.

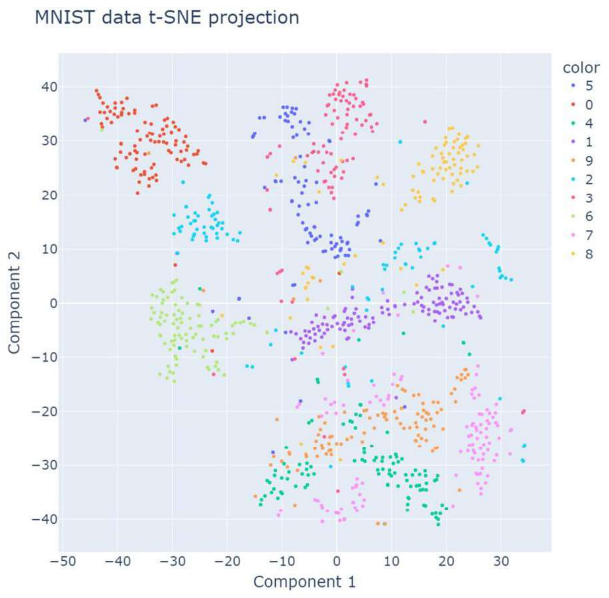

FIGURE 7.8 Illustration of the MNIST dataset in a 2D plot using the t-SNE algorithm with default parameters.

7.3.1 Choosing the right number of dimensions

In the t-SNE method, the number of dimensions in the lower-dimensional space is a hyperparameter that the user must provide. A smaller number of dimensions generally results in an easier-to-understand representation, but it may also lead to information loss. On the other hand, a visualization with more dimensions may gather more information, but it may also be more challenging to comprehend. The most common number of dimensions used in t-SNE visualizations is 2, which enables the data to be readily displayed on a 2D plot. Depending on the dataset and the task, it may be necessary to experiment with several dimensions to obtain the appropriate display. One approach to choosing the correct number of dimensions is a trial-and-error process, where the user tries different dimensions and evaluates the resulting visualizations. Another technique is to employ heuristics, such as picking the number of dimensions that captures a particular percentage of the data's variation or the number of dimensions that best isolates the various data clusters. Eventually, the number of dimensions for t-SNE visualization is determined by the data features and the research objectives.

It is important to remember that t-SNE is primarily a visualization tool and should be used with other analytical techniques to comprehend and analyze high-dimensional data thoroughly.

Using the t-SNE algorithm with its default parameters, we can obtain a 3D plot in Fig. 7.9 of the MNIST dataset that allows us to visualize the similarities and differences between handwritten digits with more details than the 2D plot we saw in the previous section.

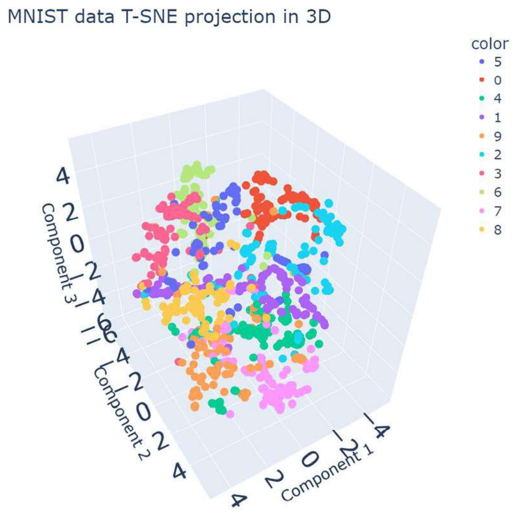

FIGURE 7.9 Illustration of the MNIST dataset in a 3D plot using the t-SNE algorithm with default parameters.

7.3.2 Interpreting t-SNE plots

t-SNE plots can be challenging, but there are a few key concepts to remember that help make sense of the visualizations:

- Understanding the basic idea behind t-SNE. The idea is to create a new representation of the data that is easier to visualize and analyze while preserving the essential relationships between the data points.
- Clusters of data points are the first item to search for in a plot created by the t-SNE algorithm. Each cluster represents a collection of related data points in the high-dimensional space. On the output plot, clusters may be represented by distinct colors or forms, making them simpler to detect. It is important to note that the distance between data points is not meaningless, but they do not represent high-dimensional spatial distances. Those near t-SNE plots are likely to be close in the high-dimensional space, whereas those far apart may not.
- Outliers are data points that are geographically far from any cluster. These may indicate unique or uncommon data items that are important to find and evaluate.
- After identifying the clusters and outliers, the next step is to look for patterns and trends in the data. For example, specific clusters might be close or far apart, or some are more tightly packed than others.
- Considering the context of the data that are analyzed is very important. Understanding the data context can help draw more meaningful conclusions from the t-SNE plot.

Finally, interpreting t-SNE plots requires understanding the basic concept behind t-SNE, identifying clusters and outliers, looking for patterns and trends, and considering the data context.

7.4 Advanced t-SNE techniques

This section will dive into two critical aspects of this algorithm worth exploring. The first aspect we will discuss is the usage of t-SNE for clustering datasets. Clustering refers to dividing a large dataset into smaller, more manageable subsets or clusters based on their inherent similarities or dissimilarities. The second aspect we will explore is the usage of t-SNE in conjunction with other dimensionality reduction methods. While t-SNE is a powerful tool, it can be even more effective when combined with dimensionality reduction methods such as principal component analysis (PCA) or linear discriminant analysis (LDA). This combination can further reduce the complexity of the data and highlight the essential features for analysis.

7.4.1 Using t-SNE for data clustering

t-SNE is a widely used algorithm for visualizing high-dimensional data in a low-dimensional space. While it can be recognized as a clustering method, it is important to note that it cannot be solely intended for clustering purposes [7]. It is better to use it as a dimensionality reduction algorithm as a preprocessing method for other clustering applications such as K-means, DBSCAN, and the others. With the help of t-SNE, this task becomes much more accessible and efficient. The algorithm identifies and highlights patterns and relationships within the data that may take time to be apparent through other methods.

7.4.2 Combining t-SNE with other dimensionality reduction methods

While t-SNE is a powerful tool for visualizing data, it has limitations, such as the sensitivity of t-SNE to the choice of parameters, like the perplexity parameter and computational complexity in larger datasets, which is one of the most significant drawbacks of this algorithm. In addition, the t-SNE algorithm sometimes needs help maintaining the global structure of the data, which might result in inaccurate representations of the information. Using t-SNE with other dimensionality reduction methods is one way to avoid the constraints imposed by this analysis technique. This collective approach is often used to speed up the computation of t-SNE and reduce its sensitivity to the choice of hyperparameters [9].

PCA is a linear dimensionality reduction technique that finds the principal components of the data. These principal components represent the directions in which the data varies the most, and they can be used to reduce the dimensionality of the data while retaining most of its variance [10]. By applying PCA before t-SNE, we can reduce the dimensionality of the data and remove any linear dependencies between the features, making the t-SNE computation more efficient and effective. In addition, by selecting the number of principal components based on the variance we want to retain, we can also control the amount of information passed to t-SNE, which can help us obtain meaningful visualizations. It is possible for us to take advantage of DCGAN to extract spectro-spatial features from hyperspectral pictures, and then we can use t-SNE to do dimensionality reduction and clustering on the features that we have extracted [11].

On the other hand, one could use t-SNE to first reduce the dimensionality of the data to a lower-dimensional space and then apply a different dimensionality reduction method, such as PCA or LDA (Linear Discriminant Analysis) [12], to reduce the dimensionality of the data further. When performed in this manner, employing t-SNE as a preprocessing step can assist in increasing the preservation of the global structure while lowering the sensitivity to the chosen parameters in the second dimensionality reduction algorithm. Nevertheless, combining t-SNE with various other strategies for dimensionality reduction may result in additional difficulties and trade-offs. For instance, the computing cost of the combined technique may be higher, and it may be more challenging to comprehend the visuals produced as a result.

7.5 Conclusion and future directions

t-SNE is a robust algorithm for visualizing high-dimensional datasets in lower dimensions, particularly useful for clustering and dimensionality reduction tasks. Due to its ability to capture nonlinear correlations in the data and maintain its local structure, t-SNE offers various benefits over other dimensionality reduction techniques like PCA. It is, nevertheless, susceptible to overfitting and is sensitive to the selection of parameters, it is thus possible that interactive investigation will be required in order to choose settings and verify outcomes [8]. Despite its limitations, t-SNE has proven to be a valuable tool for exploratory data analysis and pattern recognition. Computer vision, bioinformatics, and natural language processing are just a few of the many industries that have succeeded in using t-SNE for displaying high-dimensional information in lower dimensions. Due to its ability to meaningfully visualize complex high-dimensional data, it has been put to use in a wide variety of contexts, such as the visualization of gene expression patterns in single-cell RNA sequencing data, the identification of image clusters for use in computer vision, and the investigation of the structure of natural language data.

The t-SNE field of study may advance along several different paths in the future. Developing techniques for automatically identifying optimum parameter settings for t-SNE might be one area that receives attention as a potential emphasis area. This would assist in solving one of the primary issues presented by the algorithm, which is the situation's sensitivity to the chosen parameters. Another area of focus could be developing new ways to lower computation complexity for larger datasets, which in the current version is very demanding. In conclusion, t-SNE is a practical approach for representing high-dimensional data and may be used in various of contexts. While there are still obstacles to overcome, t-SNE research promises to continue to advance our understanding of complex data structures and relationships.

References

[1] L. Maaten, G. Hinton, Visualizing data using t-SNE, Journal of Machine Learning Research 9 (2008).

[2] G. Hu, M. Ahmed, M. L'Abbé, Natural language processing and machine learning approaches for food categorization and nutrition quality prediction compared with traditional methods, The American Journal of Clinical Nutrition 117 (2023) 553–563, https://www.sciencedirect.com/science/article/pii/S0002916522105526.

[3] D. Kobak, P. Berens, The art of using t-SNE for single-cell transcriptomics, Nature Communications 10 (2019) 5416, https://doi.org/10.1038/s41467-019-13056-x.

[4] H. Robbins, A stochastic approximation method, The Annals of Mathematical Statistics 22 (1951) 400–407.

[5] I. Csiszar, I-divergence geometry of probability distributions and minimization problems, Annals of Probability 3 (1975) 146–158, https://doi.org/10.1214/aop/1176996454.

[6] M. Blum, M. Nunes, D. Prangle, S. Sisson, A comparative review of dimension reduction methods in approximate Bayesian computation, Statistical Science 28 (2013) 189–208, https://doi.org/10.1214/12-STS406.

[7] G. Linderman, S. Steinerberger, Clustering with t-SNE, provably, CoRR, arXiv:1706.02582, 2017.

[8] M. Wattenberg, F. Viégas, I. Johnson, How to use t-SNE effectively, Distill (2016), http://distill.pub/2016/misread-tsne.

[9] H. Huang, Y. Wang, C. Rudin, E. Browne, Towards a comprehensive evaluation of dimension reduction methods for transcriptomic data visualization, Communications Biology 5 (2022) 719, https://doi.org/10.1038/s42003-022-03628-x.

[10] K. Pearson, LIII. On lines and planes of closest fit to systems of points in space, Zenodo, https://doi.org/10.1080/14786440109462720.

[11] R. Silva, P. Melo-Pinto, t-SNE: a study on reducing the dimensionality of hyperspectral data for the regression problem of estimating oenological parameters, Artificial Intelligence in Agriculture 7 (2023) 58–68, https://www.sciencedirect.com/science/article/pii/S2589721723000053.

[12] P. Cohen, P. Cohen, S.G. West, L.S. Aiken, Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences, Psychology Press, 1983.

[13] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324, https://doi.org/10.1109/5.726791.