t-SNE: Seeing the Invisible Patterns in High-Dimensional Data

---

Welcome to tutorial: The Problem with Seeing

Imagine standing in a room filled with thousands of paintings — each painted in a slightly different style. Some are similar, others completely different. But here's the catch:

You can only look at two walls at a time.
You can't fly above. You can't rotate. You can't see the whole space.

That's what it's like trying to understand **high-dimensional data** using only **2D or 3D plots**. It's a problem of **projection**: we need to flatten the data in a way that **doesn't lose the patterns that matter most**.

This is where dimensionality reduction — and especially **t-SNE** — becomes our superpower.

🔬 What Is t-SNE, really?

t-SNE stands for *t-distributed Stochastic Neighbor Embedding*. Yes, it sounds complex. But here's the core idea:

- It tries to preserve the **relative similarity between data points**.
- It keeps **close points close** and **far points far apart**, but only where it really matters in their **local neighborhood**.
- It uses a **probability distribution** to model similarities — not Euclidean distance alone.

In simple terms:

If two samples are neighbors in high-dimensional space, t-SNE will place them close in 2D or 3D as well.

Unlike PCA, which uses **linear algebra** to project axes of maximum variance, t-SNE is **non-linear**, preserving structure in a way our eyes can truly interpret.

Under the Hood: How t-SNE Works

Let's step into how t-SNE functions — in intuitive terms.

1. **Start in high-dimensional space**: Each pair of data points gets assigned a probability, based on how close they are.
    - This uses a Gaussian (normal) distribution.
    - The result: similar points have higher probability to be "neighbors."
2. **Move to low-dimensional space (e.g., 2D)**: t-SNE builds another probability distribution — this time using a **t-distribution**, which has heavier tails.
3. **Minimize KL-Divergence**:
    - It tries to make the two distributions (high-D and low-D) as similar as possible.
    - This is done through **gradient descent**, minimizing the difference between perceived and projected similarity.

Why t-distribution?

Because it avoids "crowding" in 2D — it spreads out clusters more naturally, avoiding the common PCA problem where everything bunches together.

***tip**:* Think of t-SNE as creating a **"map" of relationships**, not a projection of raw values.

Real-World Applications of t-SNE

Let's now talk about where t-SNE shines:

Domain	Use Case
NLP	Visualizing word embeddings (e.g., Word2Vec, BERT)
Bioinformatics	Understanding gene expression patterns
Computer Vision	Visualizing CNN feature vectors from image classification
Finance	Clustering customer behavior or risk profiles
Anomaly Detection	Spotting outliers visually in large datasets

In each of these, t-SNE allows **humans to see what models see** — something that's not just useful, but empowering.

A Quick Peek at Our Dataset

The dataset used in this tutorial is the **Digits dataset** from the scikit-learn library, a well-known benchmark for image-based machine learning tasks. It contains **1,797 grayscale images of handwritten digits (0–9)**, each formatted as an **8×8-pixel grid**, totaling **64 numerical features per image**. Each feature represents the intensity of a pixel, unrolled into a flat vector. The target labels (y) correspond to the actual digit in the image. This dataset is ideal for dimensionality

reduction because it is small, interpretable, and exhibits natural clusters — yet remains **high-dimensional enough** (64 features) to showcase the visualizing power of tools like PCA and t-SNE. Its built-in availability in sklearn.datasets also makes it perfect for hands-on exploration and instructional demos.

We use the **digits dataset** — 64 features representing 8x8 images of numbers.

```
digits = load_digits()
X = digits.data
y = digits.target
print("Data shape:", X.shape)
```

Let's visualize some:

```
fig, axes = plt.subplots(1, 10, figsize=(10, 1))
for i, ax in enumerate(axes):
    ax.imshow(digits.images[i], cmap='gray')
    ax.axis('off')
    ax.set_title(str(digits.target[i]), fontsize=10)
plt.suptitle("Sample Handwritten Digits")
plt.show()
```

Figure 1 Each image is an 8×8 grid, unrolled into 64 numerical features — invisible to the eye until visualized.

Why Standardize Before t-SNE?

Before applying t-SNE or PCA, we use:

```
from sklearn.preprocessing import StandardScaler
X_scaled = StandardScaler().fit_transform(X)
```

Why? Because pixel intensity may vary across features, and t-SNE uses distance metrics. We don't want one feature (like corner brightness) to dominate just because its values are larger.

A Quick Stop at PCA (Our Control Group)

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y, palette='tab10', legend='full', s=40)
plt.title("PCA Projection of Digits (2D)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="Digit")
plt.grid(True)
plt.show()
```

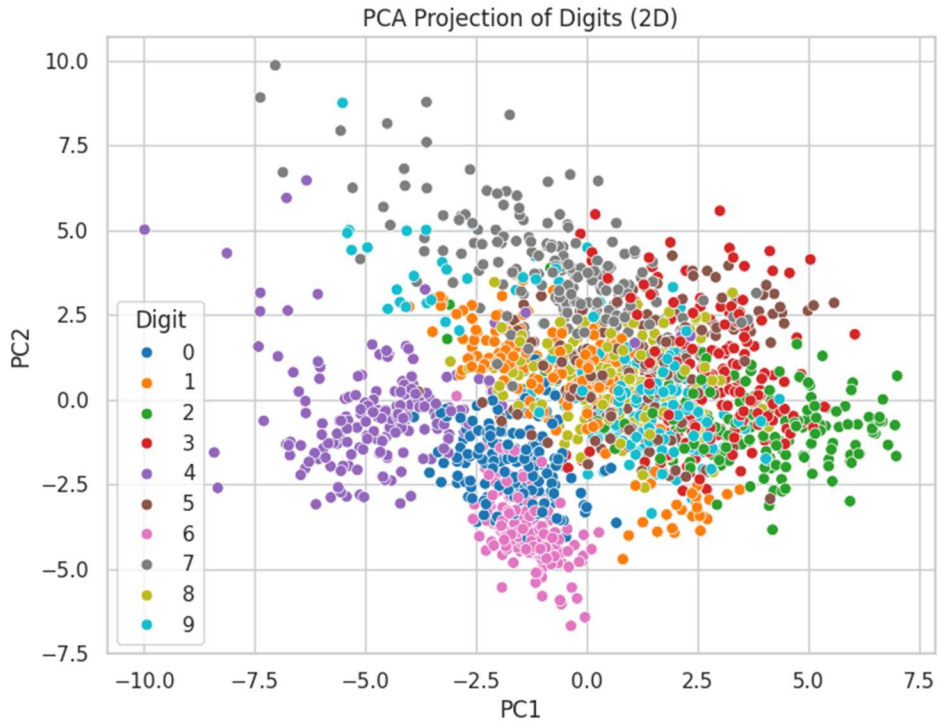

Figure 2 PCA projection: some structure is visible, but many digits overlap or blend.

Note: PCA retains **global variance** but not local structure. It often captures dominant trends but misses the nuances of clusters.

Enter t-SNE: Time to See the Invisible

```
tsne = TSNE(n_components=2, perplexity=30, n_iter=1000, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

plt.figure(figsize=(10, 7))
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=y, palette='tab10', legend='full', s=40)
plt.title("t-SNE Projection of Digits (2D)")
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.legend(title="Digit")
plt.grid(True)
plt.show()
```

Now we plot:

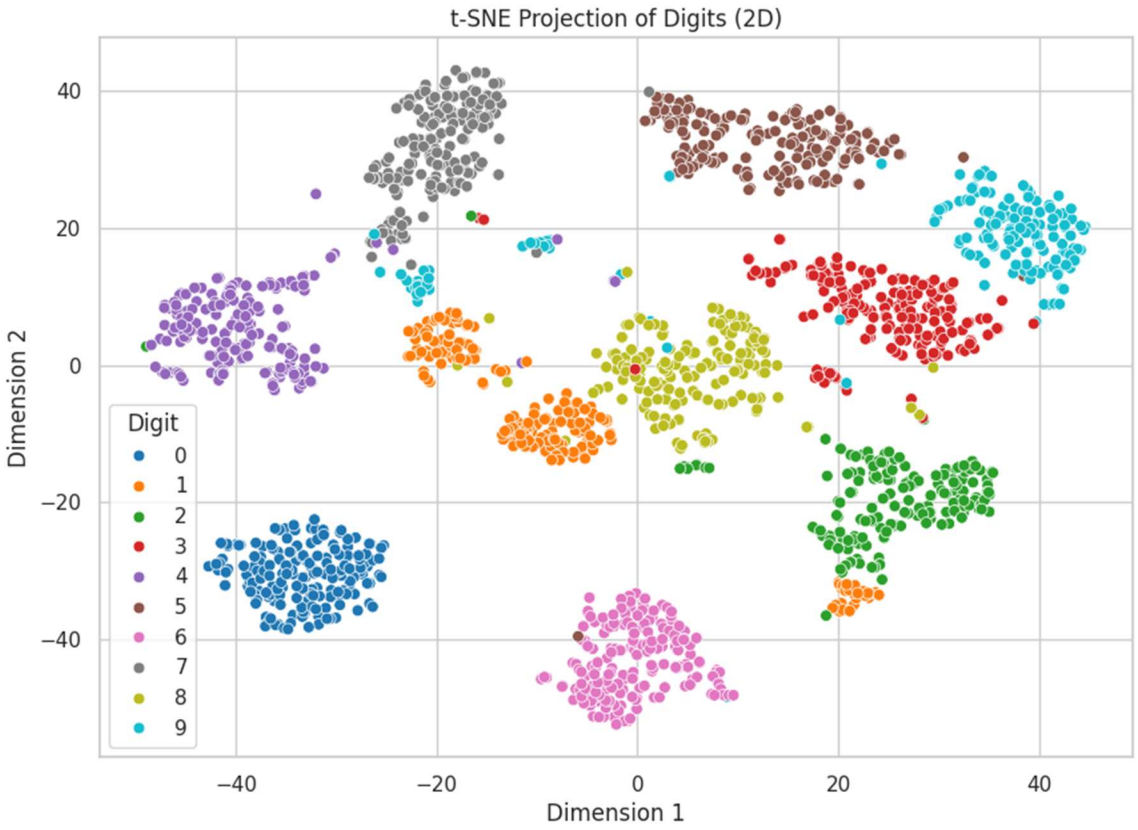

Figure 3 t-SNE projection: crisp clusters emerge, each digit forming a well-defined group.

You can now **see** the 2s, the 7s, the 8s — each as a **separate cloud of similarity**.

Comparing PCA and t-SNE Visually

Technique	Preserves	Use Case	Pros	Cons
PCA	Global variance	Preprocessing, data compression	Fast, interpretable	Linear only
t-SNE	Local neighborhoods	Visualization, clustering	Reveals fine structure	Slow, non-deterministic, not for modeling

---

When NOT to Use t-SNE

t-SNE is not for:

- Supervised learning or classification
- Feature selection
- Large-scale batch processing (use UMAP instead)

Use t-SNE only for exploration and understanding.

Summary: What We Learned

- t-SNE gives us a **new way to look** at data we couldn't understand before.
- It's **non-linear**, **visual**, and focuses on **preserving similarity**.
- It works beautifully on **high-dimensional data**, revealing clusters and structure that PCA often hides.
- It's not a replacement for modeling — but a powerful **thinking and understanding tool**.

GitHub Repository Structure

This repository contains all files needed to reproduce the t-SNE visualization tutorial using the Digits dataset. Below is the recommended structure:

File / Folder	Description
raw.githubusercontent.com/AGHANAKS/t-SNE-tutorial/refs/heads/main/tsne_visualization_digits_tutorial.ipynb	Main notebook with all steps: data loading, PCA, and t-SNE

raw.githubusercontent.com/AGHANAKS/t-SNE-tutorial/refs/heads/main/README_tSNE_Tutorial.md	Project overview and instructions
raw.githubusercontent.com/AGHANAKS/t-SNE-tutorial/refs/heads/main/requirements_tSNE_Tutorial.txt	Python packages required to run the notebook

---

GitHub Link

```
GitHub Repository:
https://github.com/AGHANAKS/t-SNE-tutorial.git
```

References:

- van der Maaten, L., & Hinton, G. (2008). *Visualizing data using t-SNE*. Journal of Machine Learning Research, 9(Nov), 2579–2605.
- Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python*. JMLR, 12, 2825–2830.
- Wattenberg, M., Viégas, F., & Johnson, I. (2016). *How to Use t-SNE Effectively*. Distill