t-SNE: Visualizing High-Dimensional Data

Preserving Neighborhoods in Two Dimensions

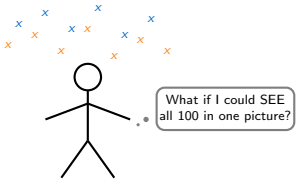
Methods & Algorithms

MSc Data Science – Spring 2026

How Do You Spot a Market Crash in 100 Variables?

The Visualization Challenge

- You monitor 100 risk indicators: VIX, credit spreads, correlations, order flow...
- A dashboard with 100 time series is overwhelming – you cannot see the "big picture"
- What if you could collapse all 100 signals into a single 2D map?



The Question

Can we project 100D data into 2D so that similar market states cluster together – revealing regimes, anomalies, and transitions?

van der Maaten & Hinton (2008) introduced t-SNE – now the standard tool for visualizing high-dimensional data

Learning Objectives

After this lecture, you will be able to:

1. **Derive** t-SNE's KL divergence objective and understand the role of the gradient
2. **Analyze** how perplexity controls the balance between local and global structure
3. **Evaluate** t-SNE's limitations compared to PCA and UMAP
4. **Apply** t-SNE to detect market regimes and spot financial anomalies

Prerequisites

Probability (conditional distributions), PCA basics (from the PCA lecture), Python with scikit-learn.

All learning objectives target Bloom's level 4+ (derive, analyze, evaluate, apply)

What Is t-SNE in Plain English?

The Dinner Guests Analogy

- Imagine seating 200 dinner guests at round tables
- Friends should sit together; strangers should be at different tables
- t-SNE does this with data: **convert distances to "friendship probabilities"** in high-D, then find a 2D seating plan that preserves those friendships

Core Idea

If two points are close in 100D (high probability of being neighbors), they must land close in 2D. If they are far, push them apart.

t-SNE in 3 Steps:

1. Compute pairwise "similarity" in high-D (Gaussian kernel)

2. Initialize random 2D layout

3. Iteratively adjust 2D positions to match high-D similarities (minimize KL divergence)

"t" = Student-t kernel in low-D

t-SNE is for visualization only – the 2D coordinates are not features for downstream models

Why Not Just Use PCA?

PCA's Linear Limitation

- Recall the Swiss roll from the PCA lecture: PCA projects onto a flat plane, destroying the manifold structure
- Points that are *far apart* on the surface end up *close together* after PCA projection
- Colors interleave instead of forming clean clusters

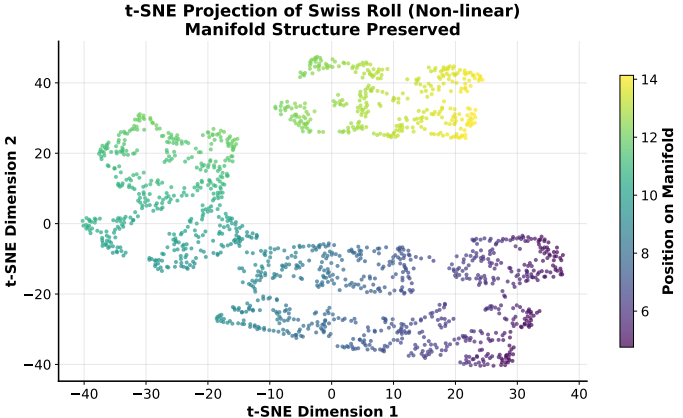
	PCA	t-SNE
Swiss roll	Flattens, colors mix	Unrolls, colors separate
Preserves	Global variance	Local neighborhoods
Best for	Linear structure	Non-linear manifolds

Key Insight

PCA captures directions of maximum *spread*. t-SNE captures who is *near* whom. For discovering clusters in complex data, neighborhoods matter more.

PCA is a global method (variance); t-SNE is a local method (neighborhoods) – they answer different questions

How Does t-SNE Preserve Neighborhoods?



t-SNE "unrolls" the Swiss roll – points near each other on the surface stay near each other in 2D

Step 1: Compute Pairwise Similarities in High-D

Conditional Probability (Gaussian Kernel)

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

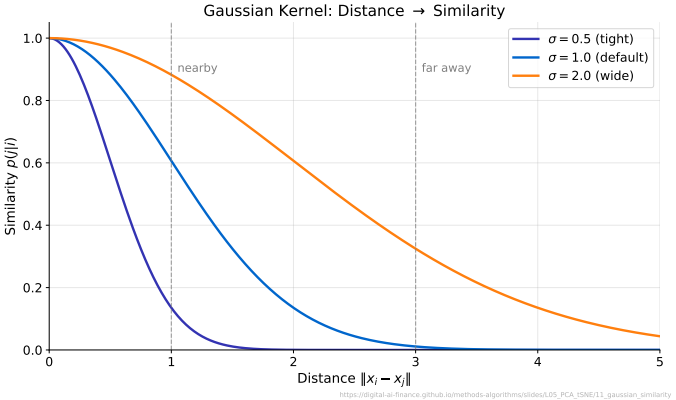
- $p_{j|i}$: probability that x_i would pick x_j as a neighbor
- σ_i: bandwidth, set automatically so each point has **perplexity** effective neighbors
- Symmetrize: $p_{ij} = (p_{j|i} + p_{i|j}) / 2n$
- Closer points $\Rightarrow$ higher p_{ij}. Far points $\Rightarrow$ $p_{ij} \approx 0$.

Intuition

The Gaussian kernel converts Euclidean distances into "how likely is x_j to be a neighbor of x_i?" – points within σ_i get high probability, distant points get near zero.

Each point gets its own σ_i – dense regions use small σ, sparse regions use large σ

How Does the Gaussian Kernel Convert Distance to Similarity?



- Small σ = only very close points are "neighbors" (tight focus)
- Large σ = distant points still get some similarity weight
- **Perplexity controls σ**: it sets how many effective neighbors each point has

Each point x_i gets its own σ_i – t-SNE adapts automatically to local density

Step 2: Similarities in Low-D (Student-t Kernel)

Student-t Kernel in 2D

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

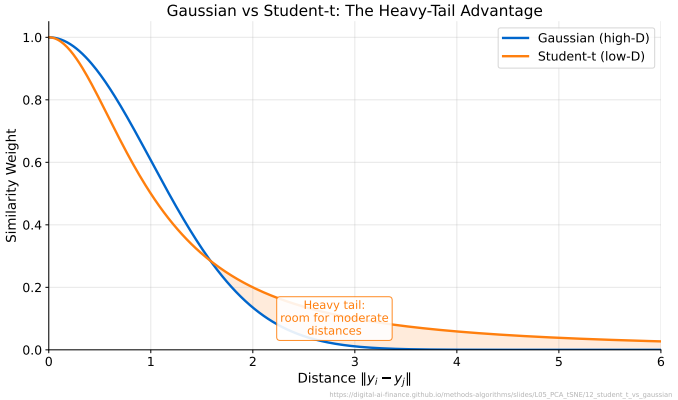
- y_i, y_j: positions in the 2D embedding
- The denominator normalizes across all pairs so q_{ij} sums to 1
- **Why Student-t?** Its heavy tails give moderate-distance pairs more room
- Unlike the Gaussian, Student-t does not decay as fast – this prevents crowding

Goal

Make q_{ij} match p_{ij} as closely as possible by adjusting the 2D coordinates y_i.

Student-t with 1 degree of freedom = Cauchy distribution – heavier tails than Gaussian, which is the key to avoiding the crowding problem

Why Does the Student-t Kernel Solve the Crowding Problem?



- Gaussian decays fast: moderate-distance pairs get almost zero weight
- Student-t has **heavy tails**: moderate pairs keep meaningful weight
- Result: clusters stay tight, but gaps between clusters become visible

The orange shaded region shows where Student-t gives more "room" than Gaussian – this prevents the crowding problem

Step 3: Minimize KL Divergence

t-SNE Objective Function

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

What the KL divergence penalizes:

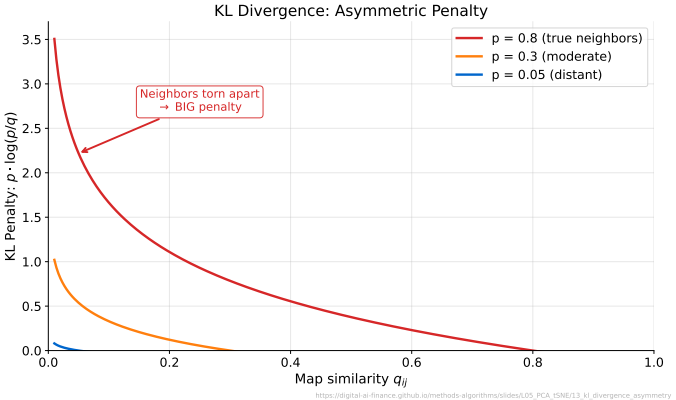
- **High p_{ij}, low q_{ij}**: nearby points in high-D that are far apart in 2D $\Rightarrow$ **large penalty**
- **Low p_{ij}, high q_{ij}**: distant points in high-D that end up close in 2D $\Rightarrow$ **small penalty**
- This asymmetry means: t-SNE is very good at keeping neighbors together, less strict about pushing non-neighbors apart

Consequence

t-SNE reliably preserves local structure (clusters). It does *not* reliably preserve global structure (distances between clusters).

KL divergence is asymmetric – $\mathrm{KL}(P\|Q) \neq \mathrm{KL}(Q\|P)$. The choice $\mathrm{KL}(P\|Q)$ prioritizes keeping nearby points together.

Why Does KL Divergence Prioritize Keeping Neighbors Together?



- When true neighbors (high p) get low map similarity (low q): **huge penalty**
- When distant points (low p) end up close (high q): small penalty
- This asymmetry means: t-SNE will sacrifice global accuracy to preserve local neighborhoods

The asymmetry explains why t-SNE cluster sizes and inter-cluster distances are unreliable

The Crowding Problem: Why Student-t?

Gaussian in Low-D: Crowding

- In high-D, there is room for many equidistant neighbors (surface of a hypersphere grows as r^{d-1})
- In 2D, there is much less room
- With a Gaussian kernel, moderate-distance points get squashed into the center
- Result: one big blob instead of distinct clusters

Student-t in Low-D: Solution

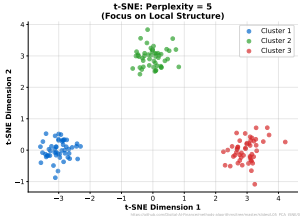
- Heavy tails allow moderate-distance pairs to be further apart in 2D
- Tight clusters form for true neighbors
- Clear gaps between clusters
- The "t" in t-SNE stands for this Student-t trick

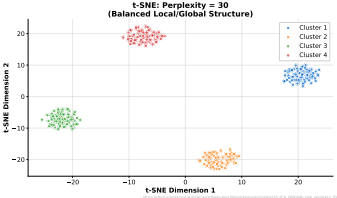
Example

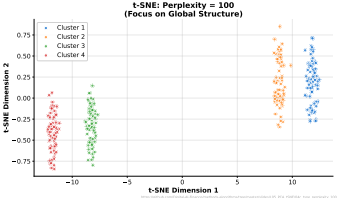
10 equidistant points on a circle in 100D: Gaussian mapping $\longrightarrow$ overlapping blob. Student-t mapping $\longrightarrow$ evenly spaced ring.

The crowding problem is why early SNE (2002, Gaussian in both spaces) produced poor visualizations – the Student-t fix (2008) was the breakthrough

What Does Perplexity Do?







- **Left** (perplexity=5): very local focus, many small fragments
- **Center** (perplexity=30): balanced – the sklearn default
- **Right** (perplexity=100): global focus, clusters may merge

Perplexity $\approx$ effective number of neighbors. Always try at least 3 values before drawing conclusions.

Perplexity Guidelines

Perplexity Range	Behavior	When to Use
5–10	Very local, risk of fragmentation	Small datasets ($n < 100$)
20–30	Balanced local/global (sklearn default)	General purpose
50–100	More global, smoother layout	Large datasets ($n > 5{,}000$)
> 100	Over-smoothed, clusters merge	Rarely useful

Rules of Thumb:

- Perplexity should be smaller than $n/3$ (where n = number of points)
- Always try at least 3 values (e.g. 5, 30, 100) before interpreting
- If patterns change drastically with perplexity, they are artifacts – not real structure

Technical Detail

Perplexity sets σ_i via binary search so that the entropy of $p_{j|i}$ equals $\log_2(\text{perplexity})$. Higher perplexity = larger σ_i = more neighbors considered.

Wattenberg et al. (2016) "How to Use t-SNE Effectively" – interactive demo and essential reading

t-SNE Caveats: What You Cannot Infer

What IS Meaningful

- Points within a cluster are truly similar in high-D
- The existence of clusters (local structure) is real
- Relative positions *within* a cluster are approximately preserved

What is NOT Meaningful

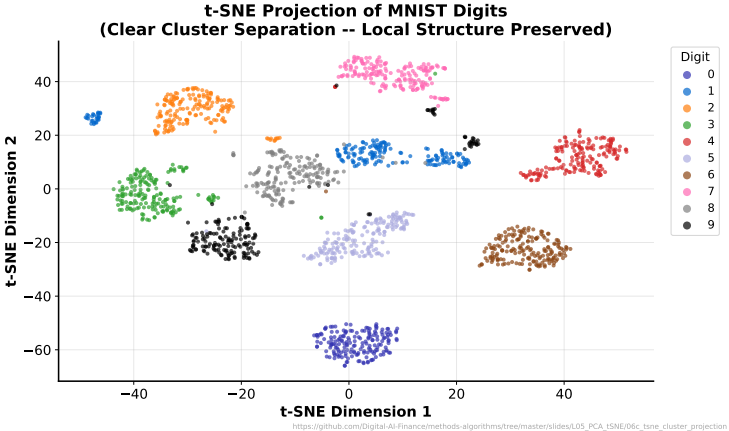
- **Cluster sizes**: a big cluster is not "more spread out" in high-D
- **Inter-cluster distances**: a gap between clusters means nothing about true distance
- **Axis orientation**: axes are random (no "x-axis = feature 1")
- **Reproducibility**: different random seeds produce different layouts

Golden Rule

t-SNE is for **exploration**, not measurement. Use it to generate hypotheses, then test them with statistical methods.

Never report "cluster A is closer to cluster B than to cluster C" based on a t-SNE plot – that distance is meaningless

How Does Cluster Preservation Compare?



t-SNE separates clusters that PCA merges – this is its key advantage for visualization of non-linear structure

Market Regime Detection with t-SNE

The Setup

- Collect daily return vectors of 50 stocks over 5 years
- Each day is a point in 50D (one dimension per stock)
- Run t-SNE to project into 2D
- Color points by date

What You See

- Clusters = **market regimes** (bull, bear, crisis)
- 2008 trading days cluster together, separate from 2005 days
- Transitions between regimes appear as bridges between clusters
- Outlier days (flash crashes) appear isolated

Interpretation

The cross-sectional return pattern on crisis days is similar to other crisis days but different from calm days. t-SNE makes this visible.

Regime detection via t-SNE complements hidden Markov models – it provides visual confirmation and exploratory insight

Anomaly Detection: Spotting Outliers Visually

Outliers in the t-SNE Map

- Points far from any cluster in the 2D embedding are potential anomalies
- In finance: rogue trading days, flash crashes, data errors, or unprecedented market events
- t-SNE does not label anomalies – it makes them **visible** for human investigation

Outlier Pattern	Possible Cause
Isolated point far from all clusters	Unprecedented event (e.g. COVID crash day)
Small cluster of 2–3 points	Repeated anomaly (e.g. flash crash pattern)
Point between two clusters	Regime transition day

Complementary Approach

Use t-SNE for visual exploration, then confirm with statistical tests (Mahalanobis distance, isolation forest).

t-SNE is a complement to, not a replacement for, statistical outlier detection methods

UMAP: The Modern Alternative

Property	t-SNE	UMAP
Speed	Slow ($O(n^2)$, Barnes-Hut: $O(n \log n)$)	Fast ($O(n^{1.14})$)
Global structure	Poorly preserved	Better preserved
Reproducibility	Random (set seed)	More deterministic
New point projection	Not supported	Supported (transform)
Theory	KL divergence	Cross-entropy on fuzzy sets
Clusters	Excellent	Excellent

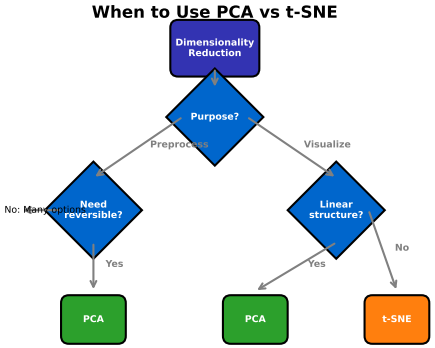
- UMAP (McInnes et al., 2018) follows the same idea: find a 2D layout preserving neighborhoods
- **Key advantage**: UMAP scales to millions of points and preserves more global structure
- **Key similarity**: both are for visualization and exploration – same caveats apply

When to Choose UMAP over t-SNE

Use UMAP when $n > 10{,}000$, when you need to project new data, or when global structure matters.

UMAP is not a strict upgrade – t-SNE may produce tighter local clusters. Try both and compare.

t-SNE vs PCA vs UMAP: Decision Framework



PCA: Fast, linear, reversible, for preprocessing

t-SNE: Slow, non-linear, visualization only, preserves local structure

https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/07_decision_flowchart

	PCA	t-SNE	UMAP
Speed	+++	+	++
Global	+++	+	++
Local	+	+++	+++
Interp.	+++	–	–
Reprod.	+++	+	++

Decision rule:
Need features? $\rightarrow$ PCA
Need clusters? $\rightarrow$ t-SNE or UMAP
$n > 10$K? $\rightarrow$ UMAP

In practice, run PCA first (reduce to 30–50D), then t-SNE or UMAP on the PCA output for best results

The t-SNE Gradient

Gradient of the KL Divergence

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

Interpreting the Gradient:

- When $p_{ij} > q_{ij}$: points are neighbors in high-D but too far in 2D $\Rightarrow$ **attractive force** pulls y_i toward y_j
- When $p_{ij} < q_{ij}$: points are not neighbors but too close in 2D $\Rightarrow$ **repulsive force** pushes y_i away from y_j
- The $(1 + \|y_i - y_j\|^2)^{-1}$ term modulates the force: nearby pairs feel stronger forces

Optimization

t-SNE uses gradient descent with momentum. Early exaggeration (multiplying p_{ij} by 4 for the first 250 iterations) helps form initial clusters.

The gradient has a physical interpretation: each point is pulled by its neighbors and pushed by non-neighbors, like a spring system

When to Use t-SNE — and When Not To

Pros

- Reveals clusters invisible to linear methods
- Handles non-linear manifolds (Swiss roll, clusters)
- Produces visually stunning, interpretable 2D plots
- Standard tool – reviewers expect it

Cons

- Slow: $O(n^2)$ naive, $O(n \log n)$ with Barnes-Hut
- Non-reproducible without fixed seed
- Cannot project new points (no `transform`)
- Uninterpretable axes and distances

Rule of Thumb

Use t-SNE for exploratory visualization of $n < 10{,}000$ points. For larger datasets or when you need projection, use UMAP.

t-SNE is the most-cited visualization method in ML – but it is also the most misinterpreted. Know the caveats.

t-SNE vs Logistic Regression Feature Space

When PCA + Logistic Regression Fails

- Logistic regression draws a linear boundary in feature space
- If classes are not linearly separable after PCA projection, LR fails
- **t-SNE visualization shows why**: classes that overlap in PCA's 2D view may form distinct clusters in t-SNE's 2D view
- This tells you the signal exists – you just need a non-linear model (SVM, random forest, neural net)

Scenario	PCA 2D	t-SNE 2D
Linear classes	Separated	Separated
Non-linear classes	Overlapping	Separated
No structure	Overlapping	Overlapping

Diagnostic Use

If t-SNE shows clusters but PCA does not, your data has non-linear structure. Switch from linear models to non-linear ones.

t-SNE as a diagnostic: it tells you whether the signal is there, even if your current model cannot capture it

Hands-on: Detect Market Regimes

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
import numpy as np

np.random.seed(42)
# Simulate 500 trading days x 50 stocks
returns = np.random.randn(500, 50) * 0.02

X = StandardScaler().fit_transform(returns)

# Try 3 perplexities
for perp in [5, 30, 100]:
    emb = TSNE(n_components=2, perplexity=perp,
               random_state=42, n_iter=1000).fit_transform(X)
    print(f"Perplexity={perp}: shape={emb.shape}")
```

Full notebook: `notebooks/L05_tsne.ipynb`

Always set `random_state` for reproducibility and `n_iter=1000` (minimum) for convergence

Key Takeaways

Concepts

- t-SNE minimizes KL divergence between high-D and low-D similarity distributions
- Student-t kernel in low-D solves the crowding problem
- Perplexity controls local vs. global balance
- Cluster sizes and distances are meaningless

Practice

- Always try 3+ perplexity values before interpreting
- Do not read distances or cluster sizes from the plot
- Use for exploration only, not as model features
- Consider UMAP for speed and global structure

One-Sentence Summary

t-SNE converts high-D distances to neighbor probabilities, then finds a 2D layout that preserves those neighborhoods – revealing clusters invisible to linear methods.

t-SNE is a lens for looking at your data. PCA is a tool for transforming it. Use both.

What's Next: Embeddings and RL

t-SNE Visualizes Learned Features. Next: How Are They Learned?

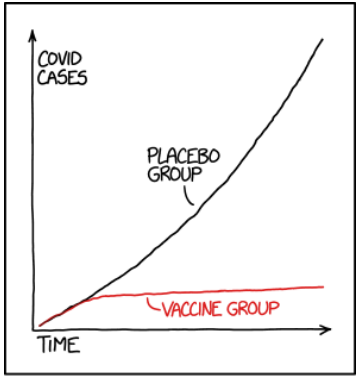
- t-SNE shows that high-dimensional data has structure – but *who creates* those high-dimensional features?
- **Word embeddings**: neural networks learn 300D representations of words where similar words are neighbors
- **Representation learning**: deep networks learn features end-to-end from raw data
- t-SNE is the standard way to *visualize* what these networks have learned

Coming Up

Next lecture: word embeddings (Word2Vec, GloVe), learned representations for financial text, and an introduction to reinforcement learning.

The famous "king – man + woman = queen" visualization is made with t-SNE applied to Word2Vec embeddings

Closing Thought



XKCD #2400 by Randall Munroe (CC BY-NC 2.5)