L05: PCA & t-SNE

Deep Dive: Theory, Derivations, and Financial Applications

Methods and Algorithms

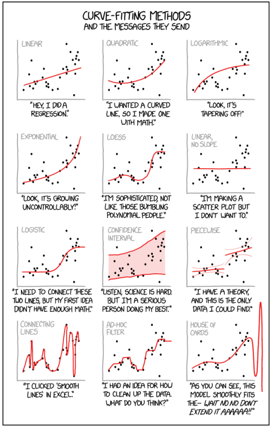
MSc Data Science

Spring 2026

Outline

1. PCA Foundations
2. PCA in Finance
3. t-SNE Theory
4. PCA vs t-SNE Comparison
5. Finance Applications
6. Implementation
7. Practice
8. Summary

Finding Signal in the Noise


CURVE-FITTING METHODS
AND THE MESSAGES THEY SEND
LINEAR
"HEY, I DID A REGRESSION."
QUADRATIC
"I WANTED A CURVED LINE, SO I MADE ONE WITH MATH."
LOGARITHMIC
"LOOK, IT'S TAPERING OFF!"
EXPONENTIAL
"LOOK, IT'S GROWING UNCONTROLLABLY!"
LOESS
"I'M SOPHISTICATED, NOT LIKE THOSE BUMBLING POLYNOMIAL PEOPLE."
LINEAR, NO SLOPE
"I'M MAKING A SCATTER PLOT BUT I DON'T WANT TO."
LOGISTIC
"I NEED TO CONNECT THESE TWO LINES, BUT MY FIRST IDEA DIDN'T HAVE ENOUGH MATH."
CONFIDENCE INTERVAL
"LISTEN, SCIENCE IS HARD. BUT I'M A SERIOUS PERSON DOING MY BEST."
PIECEWISE
"I HAVE A THEORY, AND THIS IS THE ONLY DATA I COULD FIND."
CONNECTING LINES
"I CLICKED 'SMOOTH LINES' IN EXCEL."
AD-HOC FILTER
"I HAD AN IDEA FOR HOW TO CLEAN UP THE DATA. WHAT DO YOU THINK?"
HOUSE OF CARDS
"AS YOU CAN SEE, THIS MODEL SMOOTHLY FITS THE- WAIT NO NO DON'T EXTEND IT AAAAAA!!"

XKCD #2048 "Curve Fitting" by Randall Munroe (CC BY-NC 2.5)

Learning Objectives

By the end of this deep dive, you will be able to:

1. **Derive** PCA from variance maximization via Lagrangian optimization and prove the SVD–PCA equivalence theorem
2. **Evaluate** dimensionality reduction methods using quantitative criteria (variance explained, reconstruction error, trustworthiness)
3. **Analyze** the t-SNE gradient, crowding problem, and perplexity–entropy relationship
4. **Critique** PCA limitations for nonlinear data and apply yield curve decomposition in fixed-income risk management

Finance Applications:

- Yield curve PCA (level/slope/curvature)
- Portfolio risk factor decomposition
- Market regime detection via t-SNE

Bloom's Levels 4–5: Analyze, Evaluate, Create

PCA: Variance Maximization Objective

Goal: Find orthogonal directions of maximum variance in the data.

Mean-Centering (Required First Step):

$$X_c = X - \bar{X} \quad \text{where } \bar{X}_j = \frac{1}{n}\sum_{i=1}^{n} X_{ij}$$

Key Properties of PCA:

- **Linear:** Components are linear combinations of original features
- **Uncorrelated:** Principal components have zero cross-correlation
- **Partially reversible:** Lossy reconstruction when $k < p$ (discards low-variance directions)

PCA does NOT require Gaussian data — it is optimal for Gaussian distributions, but valid for any distribution with finite second moments.

Centering is not optional: PCA on uncentered data maximizes distance from origin, not variance.

Mathematical Foundation

Covariance Matrix (from centered data):

$$\Sigma = \frac{1}{n-1} X_c^T X_c \quad \text{where } X_c \in \mathbb{R}^{n \times p}$$

Eigendecomposition:

$$\Sigma \mathbf{v}_k = \lambda_k \mathbf{v}_k, \quad k = 1, \ldots, p$$

- $\mathbf{v}_k$ = eigenvector (principal direction)
- λ_k = eigenvalue (variance along that direction), $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$

Projection to k dimensions:

$$Z = X_c W_k \quad \text{where } W_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k] \in \mathbb{R}^{p \times k}$$

Σ is symmetric positive semi-definite, guaranteeing real non-negative eigenvalues and orthogonal eigenvectors.

Why Eigenvectors? The Optimality Proof

Find direction $\mathbf{w}$ maximizing projected variance:

$$\max_{\mathbf{w}} \; \mathbf{w}^T \Sigma \mathbf{w} \quad \text{subject to } \|\mathbf{w}\| = 1$$

Lagrangian:

$$\mathcal{L} = \mathbf{w}^T \Sigma \mathbf{w} - \lambda(\mathbf{w}^T \mathbf{w} - 1)$$

First-order condition:

$$\nabla_{\mathbf{w}} \mathcal{L} = 2\Sigma\mathbf{w} - 2\lambda\mathbf{w} = 0 \implies \Sigma\mathbf{w} = \lambda\mathbf{w}$$

Result: The optimal $\mathbf{w}$ is an eigenvector of Σ.

Substituting back: $\mathbf{w}^T \Sigma \mathbf{w} = \mathbf{w}^T \lambda \mathbf{w} = \lambda$. Maximum variance is achieved at λ_1 (largest eigenvalue), so PC1 = eigenvector for λ_1.

Constrained optimization proves eigenvectors are the unique optimal solution (up to sign).

SVD–PCA Equivalence

Theorem: The principal components of X_c are the right singular vectors of X_c.
Proof sketch:

1. SVD of centered data: $X_c = USV^T$ where $S = \text{diag}(s_1, \ldots, s_r)$
2. Form $X_c^T X_c = VS^T U^T USV^T = VS^2 V^T$
3. Covariance: $C = \frac{1}{n-1} X_c^T X_c = \frac{1}{n-1} VS^2 V^T$
4. Therefore $CV = V \cdot \frac{S^2}{n-1}$ — columns of V are eigenvectors of C

Eigenvalues: $\lambda_k = s_k^2/(n-1)$ where s_k are singular values.

Why SVD is preferred: Avoids forming $X_c^T X_c$, which squares the condition number. SVD is numerically more stable for ill-conditioned data.

All practical implementations (scikit-learn, R, MATLAB) use SVD internally, not eigendecomposition.

Variance Explained and Component Selection

Explained Variance Ratio (EVR):

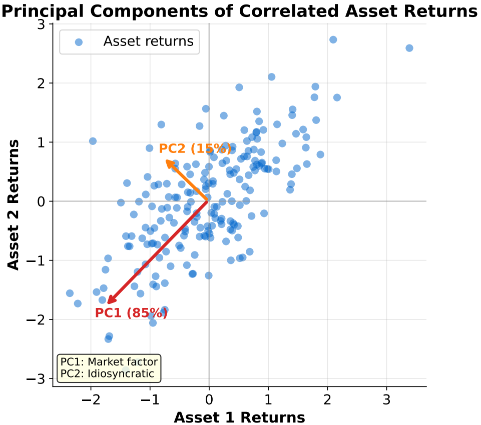
$$\mathrm{EVR}_k = \frac{\lambda_k}{\sum_{j=1}^{p} \lambda_j}, \qquad \text{Cumulative: } \mathrm{CVR}_k = \sum_{i=1}^{k} \mathrm{EVR}_i$$

Rules for choosing k:

- **Variance threshold:** Keep 80–95% of total variance (domain-dependent)
- **Scree plot elbow:** Visual inspection for diminishing returns
- **Kaiser criterion:** Retain components with $\lambda > 1$ — **valid only for correlation matrix** (standardized data)
- **Parallel analysis:** Compare eigenvalues against random data permutations (most rigorous)

In finance, 3 components explain 98%+ of yield curve variance; for equities, 5–10 may be needed.

Principal Components Visualization



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/02_principal_components

PC1 captures the dominant variance direction; PC2 captures orthogonal residual variation.

Reconstruction from Principal Components

From k components back to original space:

$$\hat{X} = ZW_k^T + \bar{X} \quad \text{(add the mean back for original scale)}$$

Reconstruction Error (Frobenius norm):

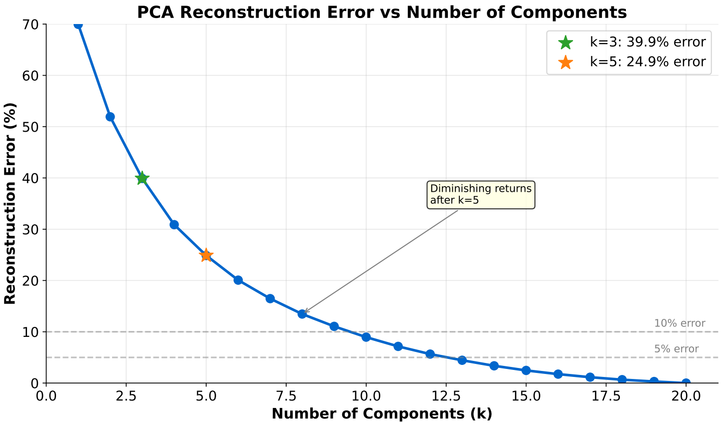
$$\|X - \hat{X}\|_F^2 = \sum_{i=k+1}^{p} \lambda_i$$

Key insights:

- Error equals the sum of *discarded* eigenvalues
- PCA gives the *optimal* rank-k approximation (Eckart–Young theorem)
- More components = less error, but diminishing returns

The Eckart–Young theorem guarantees no other linear method can do better for a given k.

Reconstruction Error vs. Number of Components



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/03_reconstruction

Adding more components always reduces error, but with diminishing marginal improvement.

Yield Curve PCA: The Canonical Example

Yield curves decompose into $\sim$3 principal components:

- **PC1 = Level** (parallel shift): explains $\sim$85% of variance
- **PC2 = Slope** (steepening/flattening): explains $\sim$10%
- **PC3 = Curvature** (butterfly): explains $\sim$3%

Together: 98%+ of all yield curve movements in 3 numbers.

Industry Applications:

- Used daily in bank risk management systems worldwide
- Foundation for duration-neutral hedging strategies
- Basis for curve trades and butterfly spreads

Litterman & Scheinkman (1991): the foundational paper establishing level/slope/curvature decomposition.

Yield Curve PCA: Worked Example

Typical PC Loadings by Maturity:

Maturity	PC1 (Level)	PC2 (Slope)	PC3 (Curvature)
1Y	0.42	−0.58	0.50
2Y	0.44	−0.37	−0.20
5Y	0.46	0.06	−0.62
10Y	0.45	0.41	−0.10
30Y	0.43	0.60	0.56

Interpretation:

- **PC1:** Near-uniform loadings — all rates move together (parallel shift)
- **PC2:** Negative short-end, positive long-end — curve steepens or flattens
- **PC3:** Positive at extremes, negative in middle — butterfly movement

Loadings from US Treasury yields; signs are conventional (eigenvectors are unique up to sign).

Portfolio Risk Decomposition with PCA

Applying PCA to equity return covariance:

- **PC1 = Market factor:** Uniform loadings, explains ∼60% of variance
- **PC2 = Sector rotation:** Growth vs. value or sector tilts
- **PC3+:** Increasingly idiosyncratic risk factors

Example: 10-Stock Portfolio

- PC1 explains 62% of portfolio variance (broad market exposure)
- First 3 PCs explain 81% (market + two sector factors)
- Remaining 7 PCs: stock-specific noise

Use Cases:

- Risk factor modeling and attribution
- Dimensionality reduction for trading signal extraction

PCA-based risk factors are model-free alternatives to explicit factor models (Fama–French).

Statistical Inference for PCA

Bootstrap Confidence Intervals:

- Resample data with replacement, recompute PCA each time
- Track loading stability and eigenvalue confidence bands

Parallel Analysis (most rigorous for k):

- Compare observed eigenvalues to eigenvalues from random data
- Keep components where $\lambda_{\text{data}} > \lambda_{\text{random}}$

Cross-Validation:

- Split data, fit PCA on training set, evaluate reconstruction on holdout

For t-SNE: Run multiple times with different random seeds; clusters that are unstable across runs are likely artifacts.

MSc-level: always quantify uncertainty in dimensionality reduction choices.

PCA Limitations

When PCA Falls Short:

- **Non-linear relationships:** Curved manifolds projected incorrectly
- **Cluster misalignment:** Maximum variance may not separate clusters
- **Outlier sensitivity:** A single outlier can rotate principal components

Solutions and Alternatives:

- **Kernel PCA:** Implicit non-linear mapping via kernel trick
- **Robust PCA:** Decomposes data into low-rank + sparse (outlier-resistant)
- **t-SNE / UMAP:** Non-linear methods optimized for visualization

PCA assumes linear structure — Gaussian NOT required, but PCA is only provably optimal (in the maximum-variance sense) for Gaussian data.

Non-linear data demands non-linear methods — motivating the t-SNE section that follows.

t-SNE: Core Idea

t-Distributed Stochastic Neighbor Embedding (van der Maaten & Hinton, 2008)

Intuition:

- Convert pairwise distances to probabilities (“who is my neighbor?”)
- **High-D:** Use Gaussian kernel to define neighbor probabilities
- **Low-D:** Use Student’s t-distribution (heavy tails)
- Minimize KL divergence between the two probability distributions

Critical distinction:

- t-SNE is a **visualization method**, NOT a preprocessing step
- Output coordinates have no interpretable axes or distances
- Cannot embed new points without re-running on entire dataset

t-SNE reveals IF clusters exist; it does not tell you HOW clusters relate to each other.

t-SNE: Mathematical Formulation

High-dimensional conditional similarity:

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

Symmetrization (joint probability):

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

Low-dimensional similarity (Student's t-distribution, 1 d.f.):

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

Objective: Minimize KL divergence $\mathrm{KL}(P \| Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$

Symmetrization ensures each point contributes equally regardless of local density.

The Crowding Problem

Why Student's t instead of Gaussian in low-D?

The problem:

- In high-D, moderate distances are *common* (volume grows exponentially)
- Mapping to 2D with Gaussian: moderate distances collapse to small distances
- Result: all points crowd into center, clusters become indistinguishable

The solution — heavy-tailed t-distribution:

- Heavy tails allow dissimilar points to be placed *far apart* in 2D
- Nearby points stay nearby (Gaussian and t agree at short distances)
- Far-away points get pushed apart (t-distribution decays slower)

This is the key innovation of t-SNE over the original SNE (Hinton & Roweis, 2002).

The crowding problem is fundamental to all high-to-low dimensional embeddings, not just t-SNE.

KL Divergence and the Gradient

KL Divergence (asymmetric):

$$\mathrm{KL}(P\|Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

- Penalizes large p_{ij} with small q_{ij} heavily (nearby $\rightarrow$ must stay nearby)
- Tolerates large q_{ij} with small p_{ij} (distant $\rightarrow$ can be anywhere)
- This asymmetry is why t-SNE preserves *local* structure

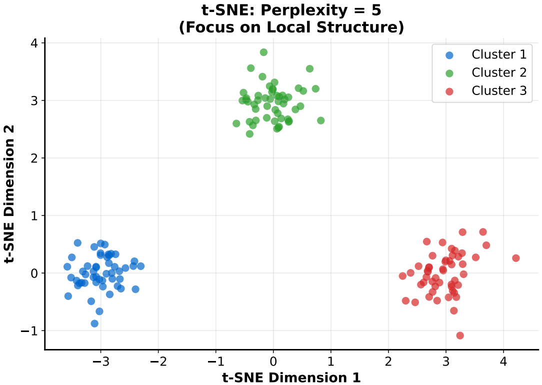
Gradient (the force on each embedding point):

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

Interpretation: Attractive forces ($p_{ij} > q_{ij}$) pull neighbors closer; repulsive forces ($p_{ij} < q_{ij}$) push non-neighbors apart.

Optimized via gradient descent with momentum; early exaggeration amplifies attractive forces in initial iterations.

Perplexity: Low (Local Focus)



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/04a_tsne_perplexity_5

Perplexity = 5
$\text{Perp} = 2^{H(\sigma_i)}$

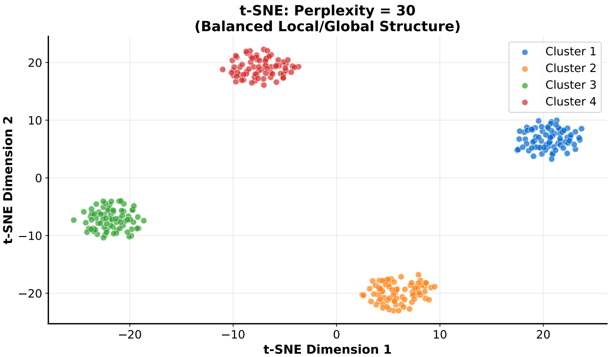
Effectively considers ∼5 nearest neighbors per point.

Effect:

- Very tight, fragmented clusters
- May split true clusters
- Captures fine local detail

Low perplexity: good for detecting sub-structure within clusters, but risks over-fragmentation.

Perplexity: Default (Balanced)



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/04b_tsne_perplexity_30

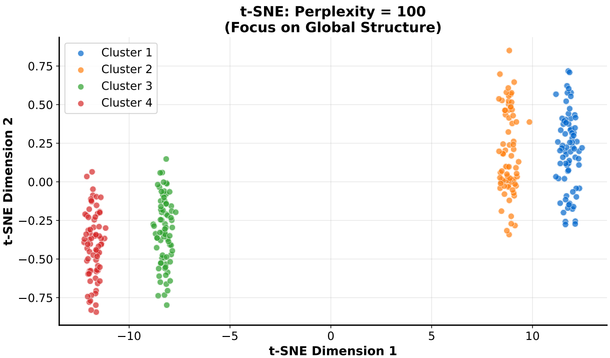
Perplexity = 30
The standard default value for most implementations.

Effect:

- Balanced local and global structure
- Clusters are well-separated
- Robust starting point

Perplexity 30 is the recommended starting point; always compare with other values.

Perplexity: High (Global Focus)



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/04c_tsne_perplexity_100

Perplexity = 100
Considers ∼100 neighbors — broader context.

Effect:

- More spread-out embedding
- Clusters may merge
- Better global relationships

High perplexity: approaches a more global view but may obscure fine cluster boundaries.

Perplexity Guidelines

Perplexity $= 2^H$ where $H = -\sum_j p_{j|i} \log_2 p_{j|i}$ is entropy.
Intuitively: the effective number of neighbors each point "considers."

Practical ranges:

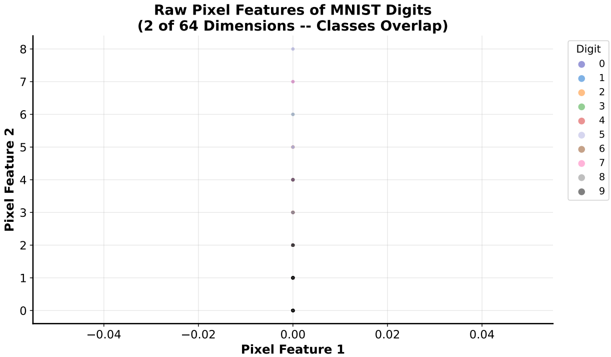
- **Low (5–10):** Fine-grained local detail; risk of fragmentation
- **Medium (30–50):** Balanced; recommended default range
- **High (100+):** Global structure emphasis; clusters may blend

Hard constraints and guidelines:

- Must be $< n$ (number of data points)
- Larger datasets tolerate higher perplexity
- **Always run multiple perplexities** to validate findings

If results change dramatically with perplexity, the clusters may not be robust.

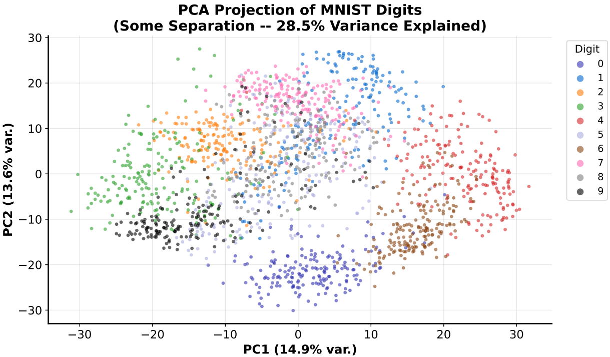
Cluster Preservation: Original Data



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/06a_original_clusters

MNIST digits (64 dimensions): classes overlap severely when viewed in raw pixel space.

Cluster Preservation: PCA Projection



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/06b_pca_cluster_projection

PCA finds max-variance directions — provides partial digit separation but clusters overlap.

t-SNE Caveats and Best Practices

Important Limitations:

- **Non-deterministic:** Different random seeds produce different layouts
- **Complexity:** $O(n^2)$ naive; Barnes-Hut approximation gives $O(n \log n)$
- **No out-of-sample:** Cannot embed new points without full re-computation
- **Hyperparameter sensitive:** Perplexity, learning rate, iterations all matter

Best Practices:

- **PCA first:** Reduce to 30–50 dimensions before t-SNE (faster, denoises)
- **Multiple seeds:** Run 3–5 times, keep structures that persist
- **Don't over-interpret:** Cluster sizes and inter-cluster distances are meaningless

t-SNE shows IF clusters exist; cluster sizes, gaps, and densities are artifacts of the algorithm.

UMAP: The Modern Alternative

Uniform Manifold Approximation and Projection (McInnes et al., 2018)

Advantages over t-SNE:

- **Faster:** $O(n)$ after nearest-neighbor graph construction
- **Better global structure:** Preserves more large-scale relationships
- **Out-of-sample embedding:** Can map new points via learned transform

Key Hyperparameters:

- `n_neighbors`: Similar role to perplexity (local vs. global balance)
- `min_dist`: Controls how tightly points cluster (0.0 = tight, 1.0 = spread)

Based on Riemannian geometry and algebraic topology (fuzzy simplicial sets).

UMAP is increasingly the default choice for visualization in production ML pipelines.

When to Use Which Method

Use PCA When:

- Preprocessing features for downstream ML models
- Linear relationships are expected or sufficient
- Reversibility needed (reconstruction, denoising)
- Speed matters (real-time, large datasets)

Use t-SNE / UMAP When:

- Visualizing high-dimensional data in 2D/3D
- Exploring cluster structure and local neighborhoods
- Non-linear manifolds expected in the data
- Exploratory data analysis (not final inference)

Common Pipeline: Standardize → PCA (30–50 dims) → t-SNE/UMAP (2D)

PCA and t-SNE are complementary, not competitors — use them together in a pipeline.

Market Regime Detection with t-SNE

Pipeline:

1. Compute rolling features: volatility, correlations, returns (window = 20–60 days)
2. Standardize features (zero mean, unit variance)
3. PCA to 15–20 dimensions (remove noise)
4. t-SNE to 2D for visualization

Typical regimes discovered:

- **Calm:** Low volatility, moderate correlations
- **Volatile:** Elevated volatility, sector dispersion
- **Crisis:** High volatility, high correlation ("all correlations go to 1")

Validation required: Cluster labels should align with known market events (2008, 2020).

Regime detection is exploratory — clusters must be validated against economic fundamentals.

PCA Preprocessing for ML Pipelines

Standard Pipeline:

$$\texttt{StandardScaler} \rightarrow \texttt{PCA(n\_components=0.95)} \rightarrow \texttt{Classifier}$$

Benefits:

- Reduces multicollinearity (orthogonal features)
- Removes noise in low-variance components
- Speeds up training for high-dimensional data

Cautions:

- PCA is unsupervised — may discard features that are low-variance but highly predictive
- Always compare ML performance with and without PCA preprocessing
- Consider supervised alternatives: LDA, feature selection

PCA preprocessing is a bias–variance tradeoff: reduces overfitting but may lose signal.

Dimensionality Reduction: Method Comparison

Method	Type	Speed	New Points?	Best For
PCA	Linear	Fast	Yes	Preprocessing, denoising
Kernel PCA	Non-linear	Medium	Approximate	Non-linear structure
t-SNE	Non-linear	Slow	No	2D visualization
UMAP	Non-linear	Fast	Yes	Visualization + embedding
Autoencoder	Non-linear	Slow	Yes	Complex non-linear features

Decision heuristic:

- Need reversibility or preprocessing? → PCA
- Need visualization? → UMAP (or t-SNE)
- Need non-linear feature extraction? → Autoencoder

No single method dominates — choice depends on goal (preprocessing vs. visualization vs. feature learning).

PCA in scikit-learn

Core API:

- PCA(n_components=k) — keep *k* components (integer)
- PCA(n_components=0.95) — keep 95% variance (float)
- pca.explained_variance_ratio_ — variance per component
- pca.inverse_transform(Z) — reconstruct original space

Implementation details:

- Uses randomized truncated SVD internally (not eigendecomposition)
- Automatically centers the data (subtracts mean)
- For sparse data: use TruncatedSVD instead (no centering)

Key practice: Always StandardScaler().fit_transform(X) before PCA when features have different units or scales.

PCA on covariance matrix (raw data) vs. correlation matrix (standardized) gives different results.

t-SNE and UMAP in Practice

t-SNE (`sklearn.manifold.TSNE`):

- `TSNE(n_components=2, perplexity=30, random_state=42)`
- Key params: `perplexity`, `learning_rate`, `n_iter`
- Set `random_state` for reproducibility; run multiple seeds to validate

UMAP (`umap-learn` package):

- `UMAP(n_components=2, n_neighbors=15, min_dist=0.1)`
- Key params: `n_neighbors` (≈ perplexity), `min_dist`
- Supports `.transform()` for new data points

Best practice pipeline: PCA to 30–50 dims first, then t-SNE/UMAP to 2D. This improves speed and removes noise.

Install UMAP via: `pip install umap-learn` **(not** `umap`**, which is a different package).**

Hands-on Exercises

Exercise 1: Yield Curve Decomposition

- Apply PCA to historical yield curve data (5 maturities)
- Interpret PC1/PC2/PC3 loadings as level/slope/curvature
- Compute cumulative variance explained

Exercise 2: PCA vs. t-SNE on Digits

- Compare PCA (2D) and t-SNE (2D) projections on MNIST digits
- Evaluate cluster separation qualitatively and via silhouette score

Exercise 3: Perplexity Sensitivity

- Run t-SNE with perplexity $\in \{5, 15, 30, 50, 100\}$
- Identify which clusters are robust across perplexities

See course notebook: L05_pca_tsne.ipynb for starter code and datasets.

Key Takeaways

PCA:

- Maximizes variance via eigendecomposition / SVD
- Optimal linear dimensionality reduction (Eckart–Young)
- Finance: yield curve = level + slope + curvature (3 PCs, 98%+ variance)

t-SNE:

- Neighbor embedding with KL divergence objective
- Crowding problem solved by t-distribution heavy tails
- Visualization only — do not use for preprocessing or inference

Pipeline: Standardize → PCA (preprocessing) → t-SNE/UMAP (visualization)
Finance: Yield curve PCA, portfolio risk factors, market regime detection

Dimensionality reduction is both a standalone tool and an essential preprocessing step.

Closing Thought

"After reducing 200 dimensions to 3,
the risk manager asked:
'Which 197 did we lose?'

Answer: 'The ones that were just noise. . .
statistically speaking.'"

— Inspired by XKCD #2400 "Statistics" by Randall Munroe

XKCD #2400 by Randall Munroe (CC BY-NC 2.5). The real question: how do you know it was noise?

References: Textbooks and Papers

Textbooks:

- Jolliffe, I.T. (2002). *Principal Component Analysis*, 2nd ed. Springer.
- James et al. (2021). *ISLR*, Chapter 12: Unsupervised Learning.
- Hastie et al. (2009). *ESL*, Chapter 14: Unsupervised Learning.

Foundational Papers:

- Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space.
- van der Maaten, L. & Hinton, G. (2008). Visualizing Data using t-SNE. *JMLR*, 9, 2579–2605.
- McInnes, L. et al. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.

The van der Maaten & Hinton (2008) paper has 40,000+ citations — one of the most cited ML papers.

References: Documentation and Finance

Software Documentation:

- scikit-learn: `sklearn.decomposition.PCA`
- scikit-learn: `sklearn.manifold.TSNE`
- UMAP documentation: `https://umap-learn.readthedocs.io/`

Finance Applications:

- Litterman, R. & Scheinkman, J. (1991). Common Factors Affecting Bond Returns. *Journal of Fixed Income*, 1(1), 54–61.

Interactive Resources:

- Wattenberg, M. et al. (2016). How to Use t-SNE Effectively. *Distill*. `https://distill.pub/2016/misread-tsne/`

The Distill article is essential reading for understanding t-SNE pitfalls and best practices.

Appendix

Advanced Topics and Proofs

Supplementary material for self-study and reference

Appendix slides are not covered in lecture — provided for advanced students and exam preparation.

Full Eigenvalue Derivation

PC1: Solve $\max_{\mathbf{w}_1} \mathbf{w}_1^T \Sigma \mathbf{w}_1$ s.t. $\|\mathbf{w}_1\| = 1$.
Lagrangian gives $\Sigma \mathbf{w}_1 = \lambda_1 \mathbf{w}_1$. Solution: $\mathbf{w}_1 =$ eigenvector for λ_1 (largest).

PC2: Solve $\max_{\mathbf{w}_2} \mathbf{w}_2^T \Sigma \mathbf{w}_2$ s.t. $\|\mathbf{w}_2\| = 1$ **and** $\mathbf{w}_2^T \mathbf{w}_1 = 0$.
Lagrangian: $\mathcal{L} = \mathbf{w}_2^T \Sigma \mathbf{w}_2 - \lambda(\mathbf{w}_2^T \mathbf{w}_2 - 1) - \mu(\mathbf{w}_2^T \mathbf{w}_1)$
First-order: $2\Sigma \mathbf{w}_2 - 2\lambda \mathbf{w}_2 - \mu \mathbf{w}_1 = 0$
Multiply by $\mathbf{w}_1^T$: $\mu = 2\mathbf{w}_1^T \Sigma \mathbf{w}_2 = 0$ (since Σ symmetric and $\mathbf{w}_1 \perp \mathbf{w}_2$).
Thus $\Sigma \mathbf{w}_2 = \lambda_2 \mathbf{w}_2$ with $\lambda_2 =$ second largest eigenvalue.

Induction: PCk is the eigenvector for λ_k with orthogonality to all previous PCs.

The orthogonality constraint propagates cleanly because Σ is symmetric.

SVD–PCA Equivalence: Full Proof

Step 1 — SVD: $X_c = USV^T$ where $U \in \mathbb{R}^{n\times n}$, $S \in \mathbb{R}^{n\times p}$, $V \in \mathbb{R}^{p\times p}$.
Step 2 — Gram matrix: $X_c^T X_c = (USV^T)^T(USV^T) = VS^TU^TUSV^T = VS^2V^T$
(since $U^TU = I$, and $S^TS = S^2$ collects s_k^2 on diagonal)

Step 3 — Covariance: $C = \frac{1}{n-1}X_c^T X_c = V \cdot \frac{S^2}{n-1} \cdot V^T$

This is the eigendecomposition of C: columns of V are eigenvectors, $\frac{s_k^2}{n-1}$ are eigenvalues.

Step 4 — Eigenvalues: $\lambda_k = s_k^2/(n-1)$.

Numerical note: $\text{cond}(X_c^T X_c) = \text{cond}(X_c)^2$. SVD avoids this squaring, reducing round-off error for ill-conditioned data.

This equivalence is why `numpy.linalg.svd` is preferred over `numpy.linalg.eig` for PCA.

t-SNE Gradient Derivation

Starting from the KL objective: $\mathrm{KL}(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_{i \neq j} p_{ij} \log p_{ij} - \sum_{i \neq j} p_{ij} \log q_{ij}$

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

$$-\sum_{i \neq j} p_{ij} \log q_{ij} = \sum_{i \neq j} p_{ij} \log(1 + \|y_i - y_j\|^2) + \log Z$$

Differentiate w.r.t. y_i and simplify:

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

Barnes-Hut approximation: Reduces $O(n^2)$ repulsive term to $O(n \log n)$ via space-partitioning trees (quadtree in 2D, octree in 3D).

The gradient has a physical interpretation: spring forces (attractive) vs. Coulomb repulsion.

Perplexity, Entropy, and Bandwidth Selection

Perplexity is defined via entropy of the conditional distribution:

$$\text{Perp}(P_i) = 2^{H(P_i)} \quad \text{where } H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

Finding σ_i: For each point i, find σ_i such that $\text{Perp}(P_i)$ equals the user-specified perplexity. Solved via **binary search** on σ_i.

Adaptive bandwidth:

- Dense regions → small σ_i (tight Gaussian)
- Sparse regions → large σ_i (wide Gaussian)
- Each point adapts to its local density

Why this matters: Without adaptive σ_i, sparse-region points would have no meaningful neighbors, and dense-region points would consider too many.

Binary search for σ_i typically converges in 20–50 iterations per point.

Kernel PCA

Idea: Apply PCA in a feature space $\phi(x)$ without explicitly computing ϕ.

Kernel matrix: $K_{ij} = \kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$
Eigendecomposition: Solve $K\alpha = n\lambda\alpha$ (dual formulation).

Common kernels:

- **Polynomial:** $\kappa(x, y) = (x^T y + c)^d$
- **RBF (Gaussian):** $\kappa(x, y) = \exp(-\gamma \|x - y\|^2)$
- **Sigmoid:** $\kappa(x, y) = \tanh(\alpha\, x^T y + c)$

Complexity: $O(n^2)$ for kernel matrix, $O(n^3)$ for eigendecomposition — prohibitive for large n. Approximations: Nystrom method, random Fourier features.

Kernel PCA bridges PCA and t-SNE: non-linear but with a well-defined mathematical framework.

Trustworthiness and Continuity Metrics

Trustworthiness $T(k)$ — are embedded neighbors true neighbors?

$$T(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in \mathcal{U}_k(i)} (r(i,j) - k)$$

where $\mathcal{U}_k(i)$ = points in k-NN of i in low-D but *not* in high-D, and $r(i,j)$ = rank of j w.r.t. i in high-D.

Continuity $C(k)$ — are true neighbors still embedded neighbors?

$$C(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in \mathcal{V}_k(i)} (\hat{r}(i,j) - k)$$

where $\mathcal{V}_k(i)$ = points in k-NN of i in high-D but *not* in low-D.

Interpretation: Both range $[0, 1]$; higher is better. Use $k = 10$–50. Compare across methods, perplexities, and random seeds.

Trustworthiness penalizes “false neighbors”; continuity penalizes “missing neighbors” in the embedding.

References and Further Reading

Advanced Theory:

- Jolliffe, I.T. & Cadima, J. (2016). Principal Component Analysis: A Review and Recent Developments. *Phil. Trans. R. Soc. A*, 374.
- Hinton, G. & Roweis, S. (2002). Stochastic Neighbor Embedding. *NeurIPS*.
- Kobak, D. & Berens, P. (2019). The Art of Using t-SNE for Single-Cell Transcriptomics. *Nature Communications*.

Finance Applications:

- Litterman, R. & Scheinkman, J. (1991). Common Factors Affecting Bond Returns.
- Alexander, C. (2008). *Market Risk Analysis, Vol. I*: Quantitative Methods in Finance. Wiley.
- Lopez de Prado, M. (2018). *Advances in Financial Machine Learning*. Wiley. Ch. 8: Feature Importance.

Lopez de Prado (2018) discusses PCA denoising for covariance matrices in portfolio optimization.