L05: PCA & t-SNE

Deep Dive: Theory, Implementation, and Applications

Methods and Algorithms

Spring 2026

Outline

Part 1: PCA Foundations

Principal Component Analysis (PCA)

- Find orthogonal directions of maximum variance
- Project data onto these directions
- Reduce dimensions while preserving information

Key Properties:

- Linear transformation
- Components are uncorrelated
- Partially reversible (lossy reconstruction when $k < p$)

PCA: one of the most fundamental tools in data science

Mathematical Foundation

Covariance Matrix:

$$\Sigma = \frac{1}{n-1} X_c^T X_c \quad \text{where } X_c = X - \bar{X} \text{ (mean-centered)}$$

Eigendecomposition:

$$\Sigma v = \lambda v$$

where v = eigenvector (principal direction), λ = eigenvalue (variance)

Projection:

$$Z = X_c W_k \quad \text{where } X_c \text{ is centered, } W_k = [v_1, \ldots, v_k]$$

Eigenvalues tell us how much variance each component captures

Why Eigenvectors? Optimality Proof

Find direction w maximizing variance:

$$\max_w w^T \Sigma w \quad \text{subject to } \|w\| = 1$$

Lagrangian:

$$L = w^T \Sigma w - \lambda(w^T w - 1)$$

First-order condition:

$$\nabla_w L = 2\Sigma w - 2\lambda w = 0$$

Result:

$$\Sigma w = \lambda w \quad \text{(eigenvalue equation)}$$

Maximum variance = largest eigenvalue λ_1

The constrained optimization proof shows eigenvectors are optimal

SVD–PCA Equivalence: The Computational Connection

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

1. The covariance matrix: $C = \frac{1}{n-1} X_c^\top X_c$
2. SVD of X_c: $X_c = U\Sigma V^\top$
3. Then: $X_c^\top X_c = V\Sigma^\top U^\top U\Sigma V^\top = V\Sigma^2 V^\top$
4. Therefore: $C = \frac{1}{n-1} V\Sigma^2 V^\top$
5. Since $CV = V \cdot \frac{\Sigma^2}{n-1}$, columns of V are eigenvectors of C

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

Computational advantage: SVD is numerically more stable than eigendecomposition of $X_c^\top X_c$ (avoids squaring condition number)

Practical implementations (scikit-learn, R) use SVD internally, not eigendecomposition.

Variance Explained

Proportion of Variance:

$$\text{Explained Variance Ratio}_i = \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j}$$

Cumulative Variance:

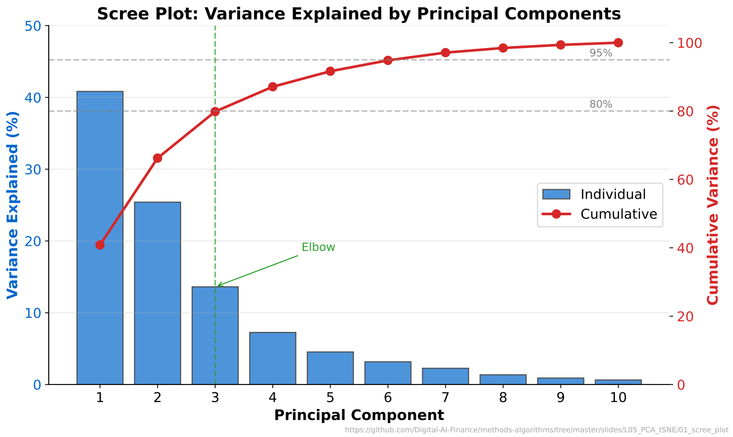
$$\text{Cumulative}_k = \sum_{i=1}^{k} \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j}$$

Rules of thumb for choosing k:

- Keep 80-95% of total variance
- Use scree plot "elbow" method
- Kaiser criterion: $\lambda > 1$ (valid for correlation matrix / standardized data only)

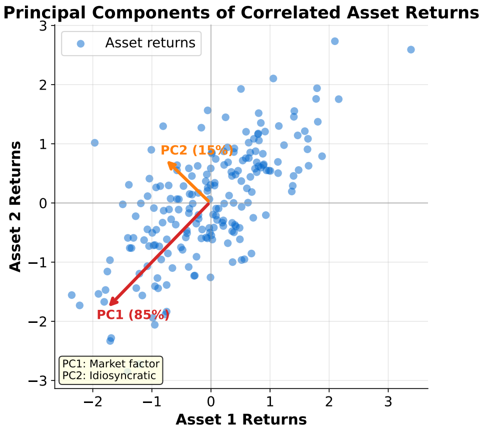
Balance dimensionality reduction with information preservation

Scree Plot



Look for the "elbow" where variance explained drops off

Principal Components Visualization



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

PC1 captures the dominant trend, PC2 the residual variation

Reconstruction

From k components back to original space:

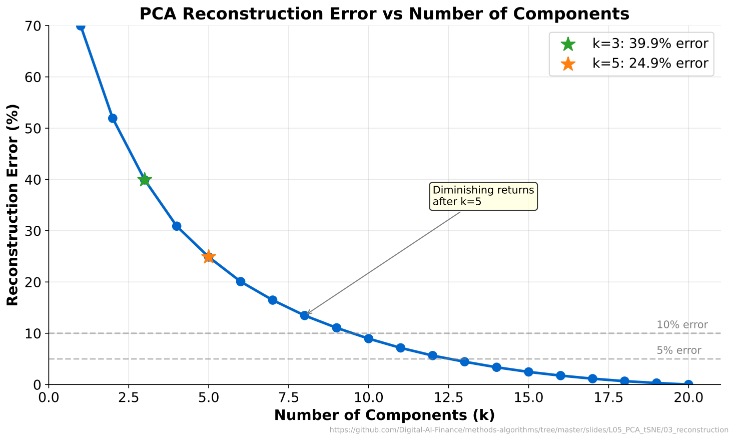
$$\hat{X} = ZW_k^T + \bar{X} \quad \text{(add mean back for original scale)}$$

Reconstruction Error:

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

Reconstruction error = sum of discarded eigenvalues

Reconstruction Error vs Components



Adding more components always reduces error (but diminishing returns)

Statistical Inference for PCA

Bootstrap Confidence Intervals:

- Resample data, recompute PCA, track loading stability

Parallel Analysis:

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

Cross-Validation for k:

- Split data, train PCA, test reconstruction on holdout

For t-SNE:

- Run multiple times with different seeds
- Unstable clusters may be artifacts

MSc-level: always quantify uncertainty in dimensionality reduction

Part 2: PCA in Finance

Portfolio Risk Decomposition:

- PC1 often represents "market factor"
- PC2-3 may capture sector/size factors
- Higher PCs: idiosyncratic risk

Example: 10-Stock Portfolio

- PC1 loadings $\approx [0.31, 0.32, 0.30, \ldots]$ — uniform (market factor)
- PC2: tech stocks positive, banks negative (sector rotation)
- First 3 PCs explain $\sim$75% of portfolio variance

Applications:

- Risk factor modeling
- Dimensionality reduction for trading signals
- Noise reduction in time series
- Feature extraction for ML models

PCA reveals latent structure in financial data

Finance Application: Yield Curve PCA

The Canonical Finance Example

- Yield curves decompose into ∼3 principal components
- PC1 = **Level** (parallel shift): explains ∼85% variance
- PC2 = **Slope** (steepening/flattening): explains ∼10% variance
- PC3 = **Curvature** (butterfly): explains ∼3% variance

Together explain 98%+ of yield curve movements

- Used daily in every bank's risk management system
- Foundation for interest rate hedging strategies

Level/slope/curvature: industry-standard yield curve decomposition

Yield Curve Factor Interpretation

Typical Loadings Pattern:

- **PC1 (Level):** Flat loadings across maturities—all rates move together
- **PC2 (Slope):** Negative short end, positive long end—curve steepens/flattens
- **PC3 (Curvature):** Negative at ends, positive in middle—butterfly trades

Trading Applications:

- Duration-neutral hedging (hedge PC1 exposure)
- Curve trades (exploit PC2 movements)
- Butterfly spreads (exploit PC3 movements)

PCA on interest rates is foundational fixed-income knowledge

PCA Limitations

When PCA Falls Short:

- Non-linear relationships (curved manifolds)
- Cluster structure not aligned with variance
- Discrete or categorical data
- Outliers heavily influence results

Solutions:

- Kernel PCA (implicit non-linear mapping via kernel trick)
- Robust PCA (outlier-resistant)
- t-SNE/UMAP (for visualization)

PCA assumes linear structure; Gaussian NOT required but PCA is optimal for Gaussian data

Part 3: t-SNE Introduction

t-Distributed Stochastic Neighbor Embedding

- Non-linear dimensionality reduction
- Optimized for visualization (2D/3D)
- Preserves local neighborhood structure

Key Idea:

- Convert distances to probabilities
- In high-D: Gaussian similarities
- In low-D: t-distribution similarities
- Minimize KL divergence between distributions

t-SNE: visualization method, NOT for preprocessing

t-SNE: Mathematical Formulation

High-dimensional similarity:

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

Symmetrize: $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$ (joint probability)
Low-dimensional similarity (t-distribution):

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

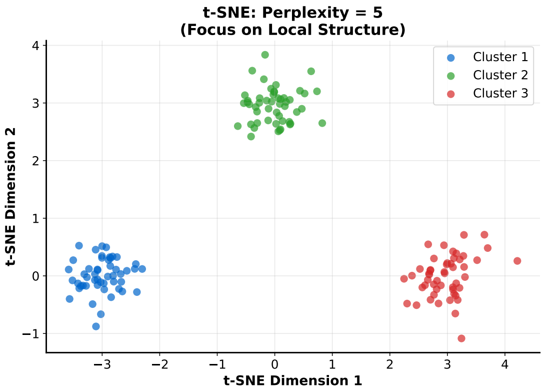
Objective: Minimize KL divergence

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

Note: KL is asymmetric—penalizes large p_{ij} with small q_{ij} more than reverse. This preserves local structure (nearby points stay nearby).

t-distribution solves the crowding problem: in high-D, moderate distances become small in low-D, causing points to collapse. Heavy tails allow dissimilar points to spread apart.

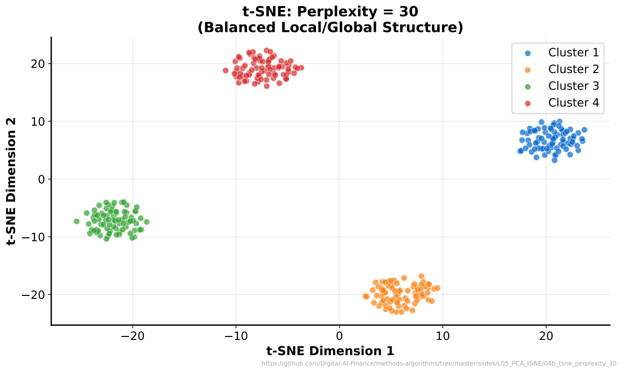
Perplexity: Low (Local Focus)



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

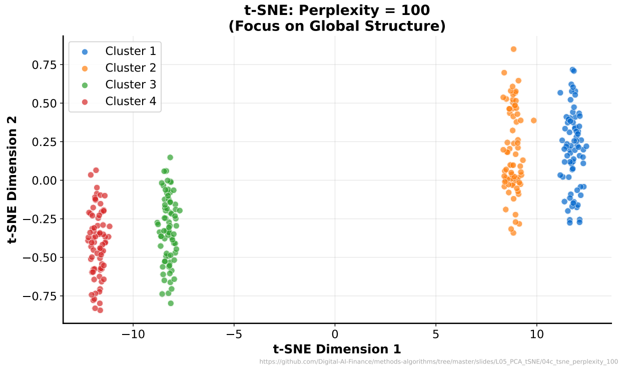
Low perplexity (5): tight clusters, focus on nearest neighbors only

Perplexity: Default (Balanced)



Default perplexity (30): balanced local and global structure

Perplexity: High (Global Focus)



High perplexity (100): more spread, clusters may merge

Perplexity Guidelines

Perplexity controls the balance between local and global structure:
Perplexity $= 2^H$ where H is entropy; roughly the effective number of neighbors considered.

- Low perplexity (5-10): Focus on very local structure
- Medium perplexity (30-50): Balanced (default)
- High perplexity (100+): More global structure

Guidelines:

- Should be smaller than number of points
- Larger datasets can use higher perplexity
- Run multiple perplexities to validate findings

Results can vary significantly with perplexity choice

t-SNE Caveats

Important Limitations:

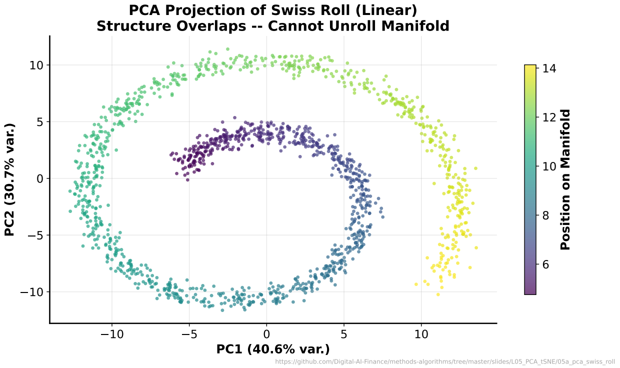
- Non-deterministic (run multiple times)
- Cluster sizes are not meaningful
- Distances between clusters are not meaningful
- Slow for large datasets (O(n^2); Barnes-Hut approximation gives O($n \log n$))

Best Practices:

- Use PCA first to reduce to 30-50 dims
- Run multiple times with different seeds
- Don't over-interpret cluster sizes/distances
- Use for exploration, not final conclusions

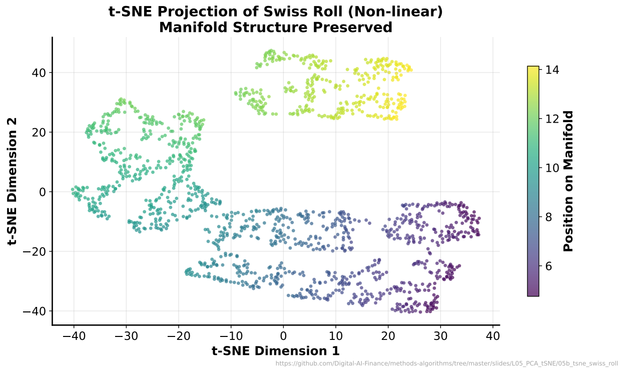
t-SNE shows IF clusters exist, not HOW they relate

Part 4: PCA on Non-linear Manifolds



PCA (linear) cannot unroll the Swiss roll - structure overlaps

t-SNE on Non-linear Manifolds



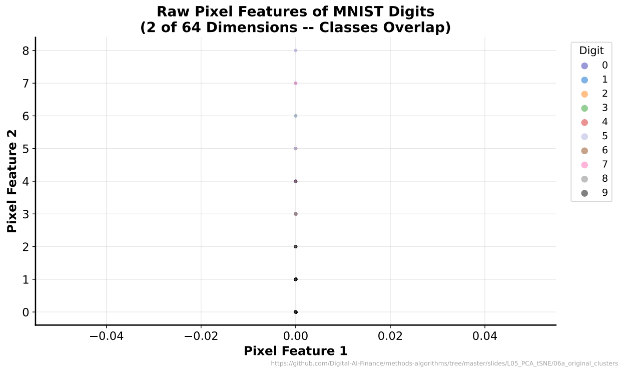
t-SNE (non-linear) successfully unrolls the manifold structure

Comparison Table

Aspect	**PCA**	**t-SNE**
Type	Linear	Non-linear
Speed	Fast $O(np^2)$	Slow $O(n^2)$
Deterministic	Yes	No
Preserves	Global variance	Local neighbors
Reversible	Lossy if $k < p$	No
Use for ML	Yes (preprocessing)	No
Visualization	Okay	Excellent

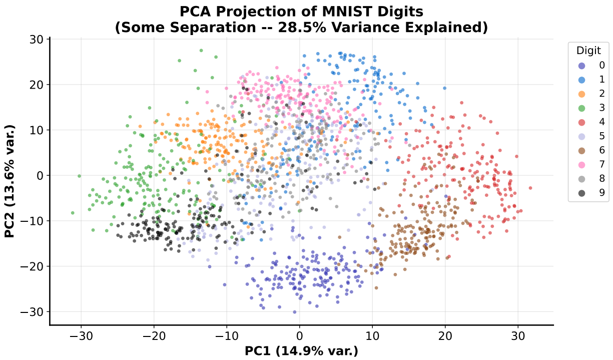
Use PCA for preprocessing, t-SNE for visualization only

Cluster Preservation: Original Data



MNIST digits (64 dims): classes overlap when viewed in raw pixel space

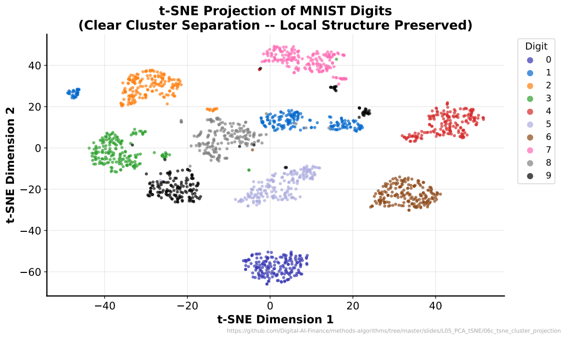
Cluster Preservation: PCA Projection



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

PCA finds max-variance directions – partial digit separation

Cluster Preservation: t-SNE Projection



t-SNE preserves local neighborhoods – clear digit cluster separation

When to Use Which

Use PCA When:

- Preprocessing for ML (reduce features)
- Linear relationships expected
- Need reversibility (reconstruction)
- Speed matters

Use t-SNE When:

- Visualizing high-dimensional data
- Looking for cluster structure
- Non-linear manifolds expected
- Exploratory analysis

Often use both: PCA first to 30-50 dims, then t-SNE for visualization

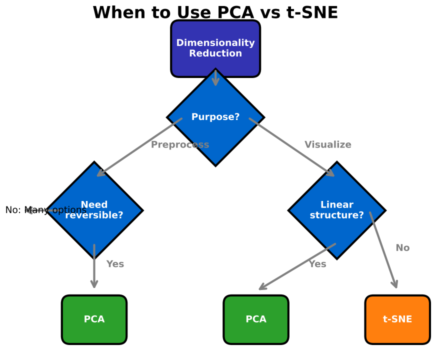
Hands-on Exercise

Open the Colab Notebook

- Exercise 1: Apply PCA to high-dimensional finance data
- Exercise 2: Visualize clusters with t-SNE
- Exercise 3: Compare PCA vs t-SNE for different datasets

Link: `https://colab.research.google.com/` See course materials

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

Consider purpose: preprocessing (PCA) vs visualization (t-SNE)

Part 5: Implementation

PCA in scikit-learn:

- `PCA(n_components=k)`: Keep k components
- `PCA(n_components=0.95)`: Keep 95% variance
- `pca.explained_variance_ratio_`: Variance per component
- `pca.inverse_transform()`: Reconstruct original
- Note: sklearn uses truncated SVD internally (more stable than eigendecomposition)

t-SNE in scikit-learn:

- `TSNE(n_components=2, perplexity=30)`
- Always normalize data first
- Consider PCA preprocessing for speed

Standardize data before PCA; normalize before t-SNE

UMAP: Modern Alternative

Uniform Manifold Approximation and Projection

- Faster than t-SNE
- Better preserves global structure
- Can embed new points (unlike t-SNE)
- Hyperparameters: n_neighbors, min_dist

When to use UMAP:

- Large datasets (faster than t-SNE)
- Need to embed new data points
- Want more preserved global structure

UMAP increasingly popular; see McInnes et al. (2018) for comparison

Summary

PCA:

- Linear, fast, reversible
- Use for preprocessing and feature extraction
- Choose k by variance explained or elbow

t-SNE:

- Non-linear, slow, visualization-only
- Excellent for exploring cluster structure
- Don't interpret distances or sizes literally

Common Pipeline: Standardize $\rightarrow$ PCA (30-50) $\rightarrow$ t-SNE (2D)

Next: Embeddings and Reinforcement Learning

References

Textbooks:

- James et al. (2021). *ISLR*, Chapter 12: Unsupervised Learning
- Hastie et al. (2009). *ESL*, Chapter 14: Unsupervised Learning

Original Papers:

- Pearson (1901). On Lines and Planes of Closest Fit
- van der Maaten & Hinton (2008). Visualizing Data using t-SNE
- McInnes et al. (2018). UMAP

Documentation:

- scikit-learn: `sklearn.decomposition.PCA`
- scikit-learn: `sklearn.manifold.TSNE`

t-SNE paper: one of the most influential visualization papers