Dimensionality Reduction Techniques

A Comprehensive Guide for Single-Cell RNA-seq Analysis

PCA for scRNA-seq

Linear dimensionality reduction technique that identifies directions of maximum variance in high-dimensional data

t-SNE Principles

Non-linear technique that preserves local structure through probability distributions

UMAP Advantages

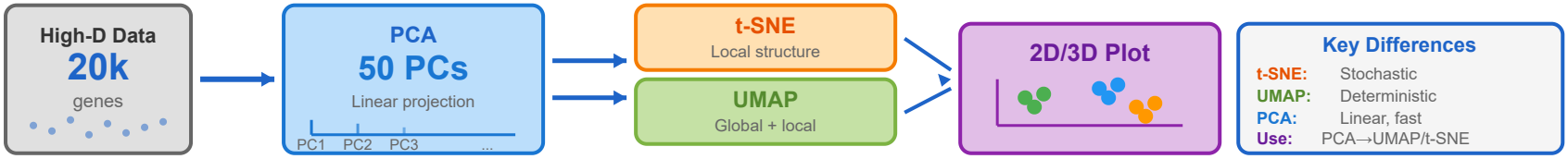
Fast manifold learning technique preserving both local and global data structure

Diffusion Maps

Captures continuous trajectories and developmental processes in biological data

Parameter Selection

Critical parameters like perplexity and n_neighbors significantly affect visualization results



PCA Principal Component Analysis (PCA)

Overview

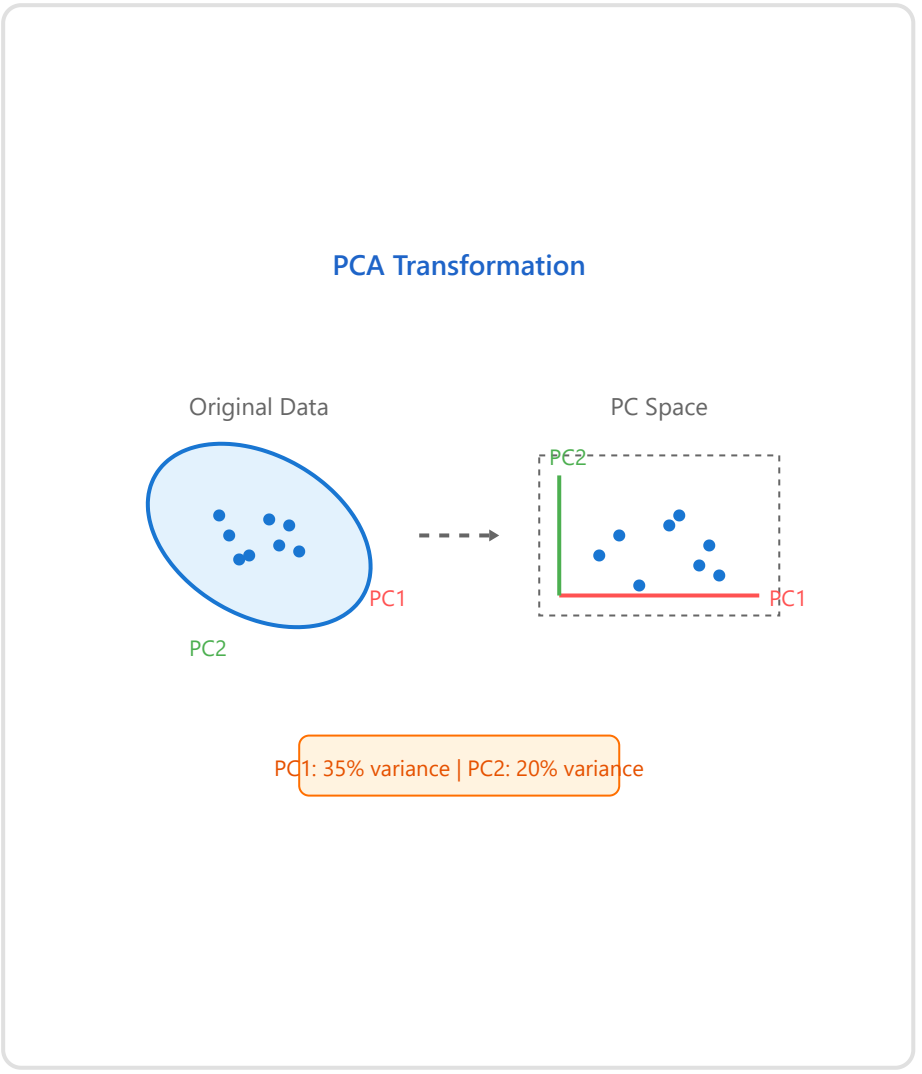
PCA is a linear dimensionality reduction technique that identifies orthogonal axes (principal components) along which the data shows maximum variance. In scRNA-seq analysis, PCA is typically the first step after normalization and feature selection, reducing thousands of genes to 10-50 principal components while preserving most biological variation.

Mathematical Foundation

PCA performs eigendecomposition of the covariance matrix or singular value decomposition (SVD) of the data matrix. Each PC is a linear combination of original features, with coefficients (loadings) indicating each gene's contribution to that component.

Key Parameters

- n_components: Number of PCs to compute (typically 10-50)
- svd_solver: Algorithm choice ('auto', 'full', 'arpack', 'randomized')
- whiten: Whether to normalize PC scores



Advantages

✓ Computationally efficient for large datasets
✓ Mathematically well-understood

Limitations

✗ Assumes linear relationships
✗ May miss non-linear patterns

✓ Preserves global structure
✓ Deterministic results

✗ Sensitive to outliers
✗ Not ideal for visualization

```python
# Python example using Scanpy
import scanpy as sc

# Perform PCA on normalized data
sc.pp.pca(adata, n_comps=50, svd_solver='arpack')

# Visualize variance ratio
sc.pl.pca_variance_ratio(adata, n_pcs=50)

# Plot PCA results
sc.pl.pca(adata, color='cell_type', components=['1,2', '2,3'])
```

tSNE t-Distributed Stochastic Neighbor Embedding (t-SNE)

Overview

t-SNE is a non-linear dimensionality reduction technique that models pairwise similarities between data points in high-dimensional space and iteratively adjusts their representation in low-dimensional space to preserve these similarities. It excels at revealing local structure and finding clusters in complex datasets.

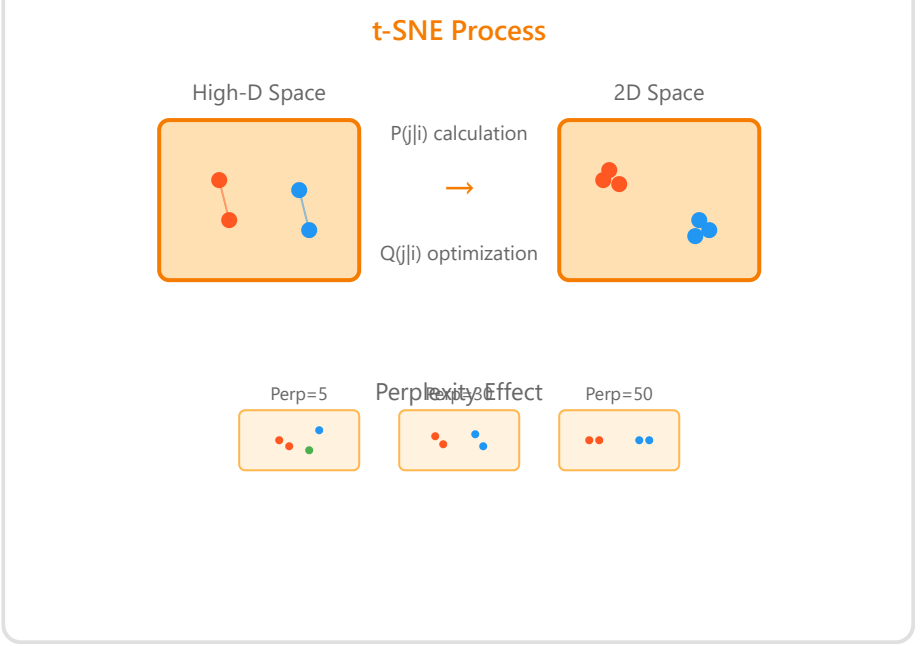
Algorithm Process

t-SNE converts high-dimensional Euclidean distances into conditional probabilities representing similarities. It uses a Student

t-distribution in the low-dimensional space to allow dissimilar objects to be modeled far apart, addressing the "crowding problem" of traditional SNE.

Critical Parameters

- perplexity: Balance between local and global aspects (5-50)
- learning_rate: Step size for gradient descent (10-1000)
- n_iter: Number of iterations (250-1000)
- early_exaggeration: Factor for spacing clusters (4-12)



Advantages

✓ Excellent for visualizing clusters
✓ Preserves local structure well
✓ Reveals complex patterns
✓ Good for exploratory analysis

Limitations

✗ Computationally expensive
✗ Non-deterministic (stochastic)
✗ Distorts global structure
✗ Cannot embed new data

```python
# Python example using Scanpy
import scanpy as sc

# Run t-SNE on PCA results
sc.tl.tsne(adata,
    perplexity=30,
    learning_rate=200,
    n_iter=1000,
    random_state=42)

# Visualize results
sc.pl.tsne(adata, color=['cell_type', 'n_genes'])
```

Uniform Manifold Approximation and Projection (UMAP)

Overview

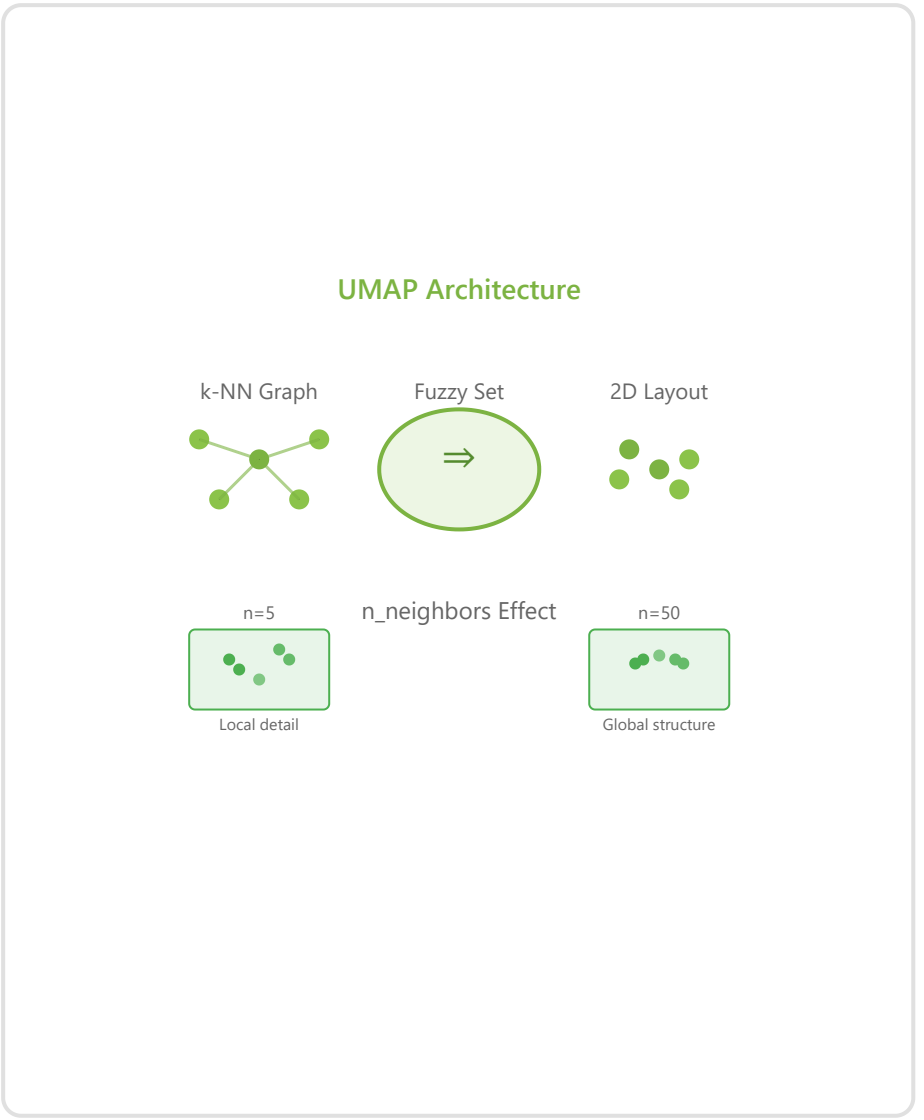
UMAP is a manifold learning technique based on Riemannian geometry and algebraic topology. It constructs a high-dimensional graph representation of the data and optimizes a low-dimensional graph to be as structurally similar as possible, preserving both local and global structure better than t-SNE.

Theoretical Foundation

UMAP assumes data is uniformly distributed on a Riemannian manifold and models the manifold with a fuzzy topological structure. The algorithm uses a novel fuzzy simplicial set representation and optimizes the cross-entropy between high and low-dimensional representations.

Important Parameters

- n_neighbors: Size of local neighborhood (5-50)
- min_dist: Minimum distance between points (0.0-1.0)
- metric: Distance metric ('euclidean', 'manhattan', etc.)
- n_components: Output dimensions (typically 2 or 3)



Advantages

✓ Faster than t-SNE

Limitations

✗ Newer, less established method

✓ Preserves global structure better
✓ More deterministic results
✓ Can handle larger datasets
✓ Supports various distance metrics

✗ Can create artificial connections
✗ Sensitive to hyperparameters
✗ Interpretation can be challenging

```python
# Python example using Scanpy
import scanpy as sc

# Compute neighborhood graph
sc.pp.neighbors(adata, n_neighbors=15, n_pcs=50)

# Run UMAP
sc.tl.umap(adata,
    min_dist=0.3,
    spread=1.0,
    n_components=2,
    random_state=42)

# Visualize with different parameters
sc.pl.umap(adata, color=['leiden', 'n_counts'], legend_loc='on data')
```

DM Diffusion Maps

Overview

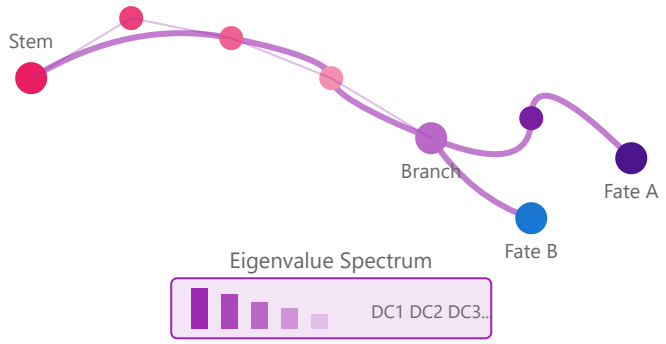
Diffusion Maps is a non-linear dimensionality reduction technique that models data as lying on a manifold and uses a diffusion process to discover its underlying geometry. It's particularly powerful for capturing continuous trajectories and branching

processes in biological systems, making it ideal for developmental studies.

Mathematical Principle

The method constructs a Markov chain on the data, where transition probabilities reflect local geometry. Eigenvectors of the transition matrix provide coordinates that respect the manifold's intrinsic geometry, with diffusion distance capturing connectivity along the manifold rather than Euclidean distance.

Diffusion Process



Key Parameters

- n_components: Number of diffusion components
- knn: Number of nearest neighbors for graph
- decay: Rate of kernel decay (bandwidth)
- t: Diffusion time parameter

Advantages

✓ Captures continuous processes well
✓ Robust to noise
✓ Preserves branching structure
✓ Good for trajectory inference
✓ Respects data geometry

Limitations

✗ Computationally intensive
✗ Parameter selection crucial
✗ Less intuitive visualization
✗ May oversmooth data

```
# Python example using Scanpy/Palantir
import scanpy as sc
import palantir

# Compute diffusion maps
sc.tl.diffmap(adata, n_comps=10)

# Run Palantir for trajectory analysis
```

```
dm_res = palantir.utils.run_diffusion_maps(adata.obsm['X_pca'], n_components=5)

# Compute pseudotime
pr_res = palantir.core.run_palantir(
    dm_res['EigenVectors'],
    start_cell,
    num_waypoints=500)

# Visualize diffusion components
sc.pl.diffmap(adata, color='pseudotime', components=['1,2', '2,3'])
```

Parameter Selection Guidelines

Overview

Proper parameter selection is crucial for meaningful dimensionality reduction results. Parameters control the balance between local and global structure preservation, computational efficiency, and biological interpretability. Understanding how parameters affect results helps avoid artifacts and reveals true biological patterns.

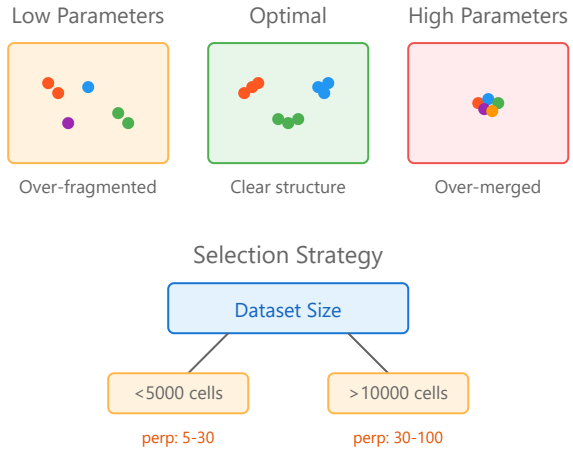
General Principles

Start with default parameters and systematically vary them to understand their effects. Consider your biological question: local structure (cell types) vs global structure (trajectories). Dataset size and sparsity influence optimal parameter choices. Always validate results using known markers and biological knowledge.

Critical Decisions

- Number of PCs: Elbow plot, technical variance
- Perplexity (t-SNE): 5-50, scale with N^0.5
- n_neighbors (UMAP): 5-50, affects connectivity
- min_dist (UMAP): 0.0-1.0, cluster tightness

Parameter Impact Comparison



Best Practices

✓ Test multiple parameter sets
✓ Use biological validation
✓ Consider computational resources
✓ Document parameter choices
✓ Check stability across runs

Common Pitfalls

✗ Using default parameters blindly
✗ Over-interpreting distances
✗ Ignoring batch effects
✗ Not checking multiple seeds
✗ Forcing biological interpretation

```
# Parameter optimization example
import scanpy as sc
import numpy as np

# Determine optimal number of PCs
sc.pp.pca(adata, n_comps=100)
sc.pl.pca_variance_ratio(adata, n_pcs=100, log=True)

# Test different perplexity values for t-SNE
perplexities = [5, 15, 30, 50]
for perp in perplexities:
    sc.tl.tsne(adata, perplexity=perp, use_rep='X_pca')
    adata.obsm[f'X_tsne_perp{perp}'] = adata.obsm['X_tsne'].copy()

# Grid search for UMAP parameters
```

```
n_neighbors_list = [5, 15, 30, 50]
min_dist_list = [0.0, 0.1, 0.3, 0.5]

for nn in n_neighbors_list:
    for md in min_dist_list:
        sc.pp.neighbors(adata, n_neighbors=nn)
        sc.tl.umap(adata, min_dist=md)
        # Evaluate clustering quality
        sc.tl.leiden(adata)
        # Calculate silhouette score or other metrics
```

Method Comparison Matrix

Method	Type	Speed	Global Structure	Local Structure	Deterministic	Best Use Case
PCA	Linear	Very Fast	Excellent	Poor	Yes	Initial reduction, QC
t-SNE	Non-linear	Slow	Poor	Excellent	No	Cluster visualization
UMAP	Non-linear	Fast	Good	Excellent	Mostly	General visualization
Diffusion Maps	Non-linear	Medium	Good	Good	Yes	Trajectory analysis
Force Atlas 2	Graph-based	Medium	Fair	Good	No	Network visualization

Key Takeaways

- **Visualization ≠ Analysis:** Use these methods for exploration, not quantitative conclusions

- **Pipeline approach:** PCA → Clustering → UMAP/t-SNE for visualization
- **Validate findings:** Always confirm patterns with known markers and biology
- **Parameter sensitivity:** Test multiple settings and report those used