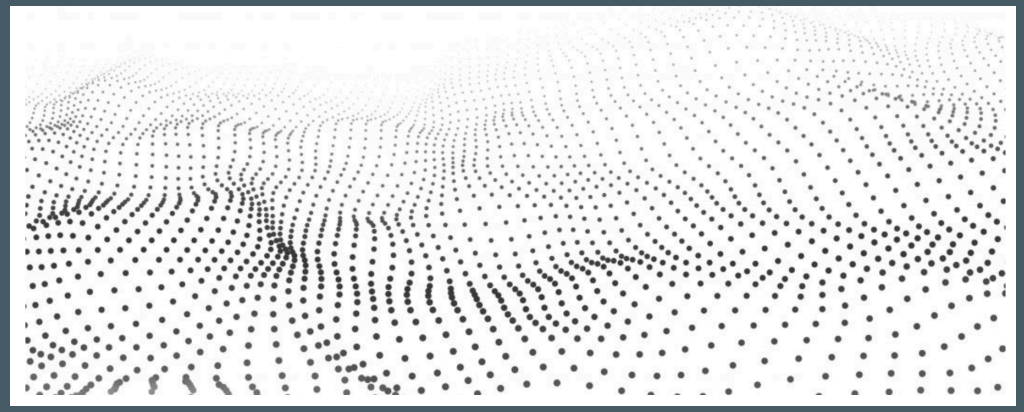
Dimension Reduction

Data as a Point Cloud

A dataset with **n samples** and **m features** can be interpreted as **n points in m-dimensional space**.



Curse of Dimensionality

High-dimensional geometry is unintuitive:

- In 2D: A random point in a 1x1 square has a 0.4% chance of being < 0.001 units from the edge.
- In 10,000D: That probability is > 99.999999%.
- Mean distance between two points:
  - In 3D cube: ≈ 0.66
  - In 1,000,000D cube: ≈ 408.25

Distance Concentration in High Dimensions

- In low dimensions (e.g. 3D), distances between random points vary significantly.
- In high dimensions (e.g. 1,000,000D), almost all distances are close to the **mean**.

This means:

> As dimension increases, the **relative difference** between the shortest and longest distances between points becomes negligible.

Curse of Dimensionality: Why It Matters

- **Sparsity**: Data becomes sparse in high-dimensional space.
  - Most volume is near the boundary.
  - Models see few meaningful neighbors.
- **Distances Become Unreliable**
  - All points become equally far.
  - Undermines k-NN, clustering, SVMs, etc.
- **Overfitting Risk**
  - More features = more degrees of freedom.
  - High chance of fitting noise, especially when $n \ll m$.

Why Reduce Dimensions?

- Simplifies models → better generalization
- Faster training, less memory usage
- Enables 2D/3D visualization of data structure
- Helps isolate signal from noise

> “Dimension reduction is feature engineering guided by geometry.”

Curse of Dimensionality: Rule Of Thumb

Rule of thumb: For non-parametric models, the number of required samples grows **exponentially** with the number of dimensions.

Examples of non-parametric models:

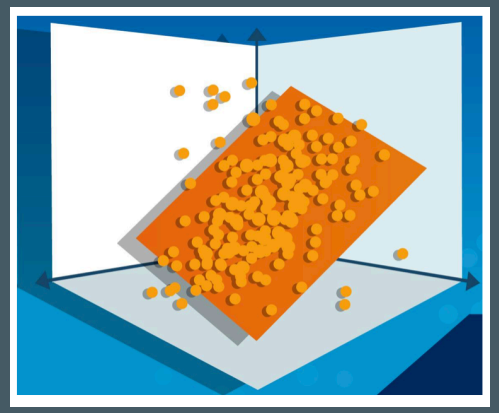
- k-Nearest Neighbors (kNN)
- Support Vector Machines (SVM)

Do the Data Lie on a Subspace?

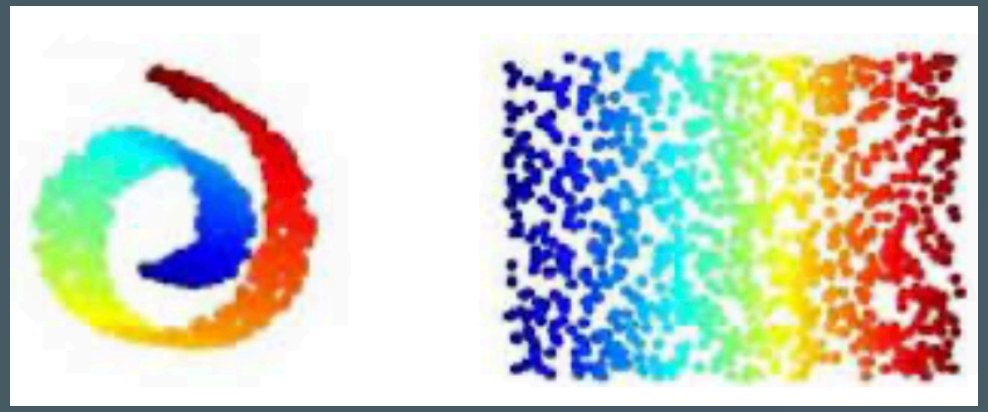
- Linear
- Curved

Dimension Reduction Approaches

- **Linear**: Projection onto a subspace (e.g., PCA)



- **Curved**: Manifold Learning (e.g., UMAP, Isomap)



PCA: Projection onto a Subspace

To find a good subspace, we use **Principal Component Analysis (PCA)**.

Key components:

- Covariance Matrix
- Change of Basis
- Singular Value Decomposition (SVD)

Covariance Matrix

For one feature x which is centered (mean 0), then:

$$\mathrm{Var}(x) = x^T x$$

Given two features x and y, both centered:

$$\mathrm{Cov}(x, y) = x^T y$$

Given a full matrix of features X which is centered:

$$\mathrm{Cov}(X) = X^T X$$

- Entry (i, j): Covariance between features i and j
- Entry (i, i): Variance of feature i
- Result is symmetric $m \times m$ matrix

Exercise: Covariance Matrix

- Load the 8x8 MNIST dataset (starter notebook provided)
- Compute the covariance matrix of the data

Basis Change

Besides the standard basis, there are many others.

If you're unsure, review: YouTube: Change of Basis

PCA Goal

Find an **orthonormal basis change** P such that P applied to X^T has diagonal covariance matrix.

In Formulas, set $Y^T = PX^T$. We want:

$$Y^TY = D$$

- D is diagonal: Maximize variances, minimize covariances
- Off-diagonal terms represent redundancy

Example: Basis Change

Given 17 samples from $\mathbb{R}^3$:

- $X^T \in \mathbb{R}^{3\times 17}$
- $P \in \mathbb{R}^{3\times 3}$

Then transformed data:

$$(P \cdot X^T)^T \in \mathbb{R}^{17\times 3}$$

Singular Value Decomposition (SVD)

If $X^T X$ is symmetric, then:

$$X^T X v_i = \lambda_i v_i$$

Here the λ_i are Eigenvalues, v_i are Eigenvectors. They exist, since X is symmetric!

Let:

$$X^T X = V D V^T, \quad D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$$

$$u_i = \frac{1}{\sqrt{\lambda_i}} X v_i$$

SVD – Matrix Form

Stack the v_i into a matrix V: $V = [v_1, \ldots, v_n]$.

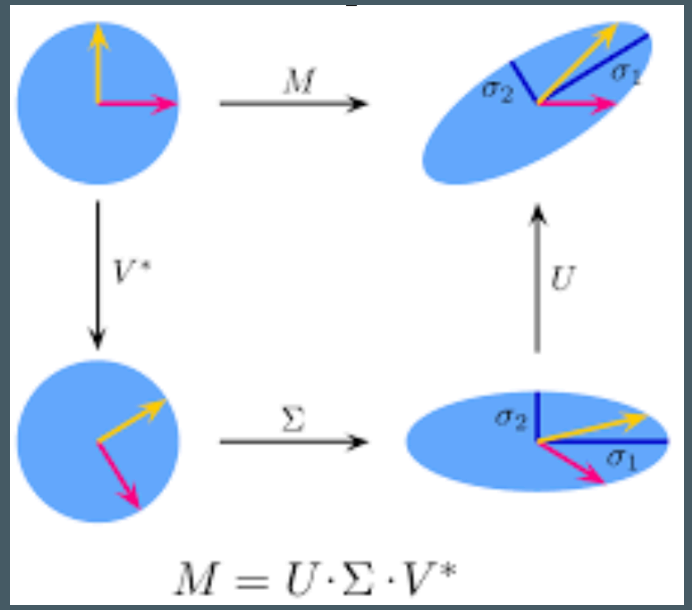
Stack the u_i into a matrix U: $U = [u_1, \ldots, u_n]$.

Assemble the $\sqrt{\lambda_i}$ into a diagonal matrix Σ, filling up with zeroes if necessary.

Then:

$$X = U\Sigma V^T$$

SVD geometrically



PCA via SVD

The base change we wanted is V^T! Using this base change, set $Y^T = V^T X^T$. Our goal was $Y^T Y = D$. We can easily check this now:

$$(V^T X^T) XV = V^T X^T XV = V^T VDV^T V = D$$

Exercise: SVD

- Use `np.linalg.svd` to compute the basis change for MNIST
- Center the matrix beforehand
- Bonus: Reconstruct the matrix using returned values (`np.allclose`)

Dimensionality Reduction with PCA

Keep only the first k columns of V to reduce dimensions:

$$X_p^T = [v_1, \ldots, v_k]^T \cdot X^T$$

We thus project onto the linear subspace spanned by the first k basis vectors.

Exercise: Projection

- Project MNIST onto first two principal components
- Remember: The base change is $V^T \cdot X^T$. To convert back to tidy format, we have to transpose again: $(V^T X^T)^T = X \cdot V$.

Comparison with sklearn.decomposition.PCA

- Use Scikit-learn to repeat the PCA projection
- Compare results with your implementation

Explained Variance

The variance of the data has not changed by the base change:

$$\sum_i (X^T X)_{ii} = \sum_i ((V^T X^T) X V)_{ii} = \sum_i D_{ii}$$

Define the **explained variance** of component i:

$$\frac{\lambda_i}{\sum_j \lambda_j}$$

This gives a good estimate on how many components to project onto. Take so many components so that you account for e.g. 90% of the variance.

Exercise: Explained Variance

- Use `pca.explained_variance_ratio_`
- How many components to capture 95% variance?

Reconstruction Error

Remember that our projection was:

$$X_p = X \cdot V_k$$

Multiplying this equation from the right with V_k^T, we arrive at our reconstruction:

$$X_r = X_p \cdot V_k^T$$

Exercise: Reconstruction

- Reconstruct MNIST from PCA with 95% variance
- Use `pca.inverse_transform`
- Visualize the first 2 reconstructed images

Kernel PCA

- The kernel trick works for PCA just as for other methods.
- Use kernel trick: Linear in feature space = nonlinear in input space
- Kernel trick: Don’t need the feature space explicitly, it is only used implicitly

Exercise: Kernel PCA

- Apply kernel PCA to Swiss Roll dataset
- Use RBF and Sigmoid kernels
- Experiment with γ values

Random Projection

- Instead of maximizing variance, try preserving distances
- This works by using a random choice of subspace in the smaller dimensional target! This is counter-intuitive.
- The Johnson–Lindenstrauss Lemma guarantees the result is pretty good after a sufficient amount of tries.

Random Projection – Theory

For $0 < \epsilon < 1$ and n points in $\mathbb{R}^d$, there exists a projection $p : \mathbb{R}^d \to \mathbb{R}^k$:

$$(1-\epsilon)\|x_i - x_j\|^2 \leq \|p(x_i) - p(x_j)\|^2 \leq (1+\epsilon)\|x_i - x_j\|^2$$

for all i, j, with $k > \frac{4\log(n)}{\epsilon^2/2 - \epsilon^3/3}$

So there always exists a radom projection preserving distances. Only open question: How many radom choices till I find it?

Random Projection - Implementation

Just use brute force:

- Choose k random vectors in $\mathbb{R}^d$.
- If they are not linear independent, choose again.
- If linear independent, project onto subspace
- If Projection is not good, start again.

Exercise: Random Projection

- Use `sklearn.random_projection.GaussianRandomProjection`
- Project MNIST to 10D
- What if no dimension is given?
- Try reconstructing

Random Projection – No Brute Force

- Construct matrix $R \in \mathbb{R}^{d \times k}$ with 0/1 entries (coin flip)
- Use the projection:

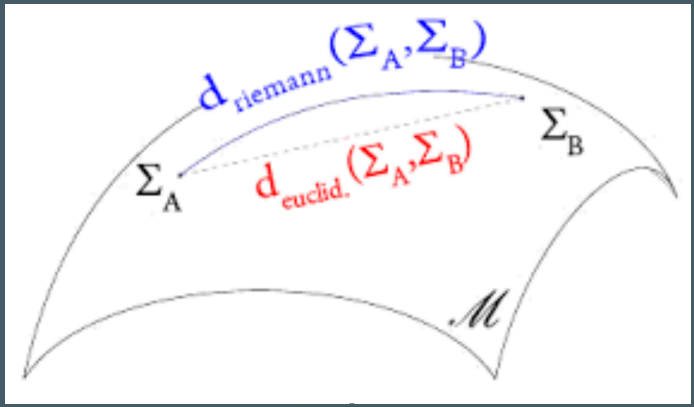
$$\frac{1}{\sqrt{k}} \cdot R \cdot X$$

With a sufficiently high probability, the result has the desired properties.

Curved Data Now

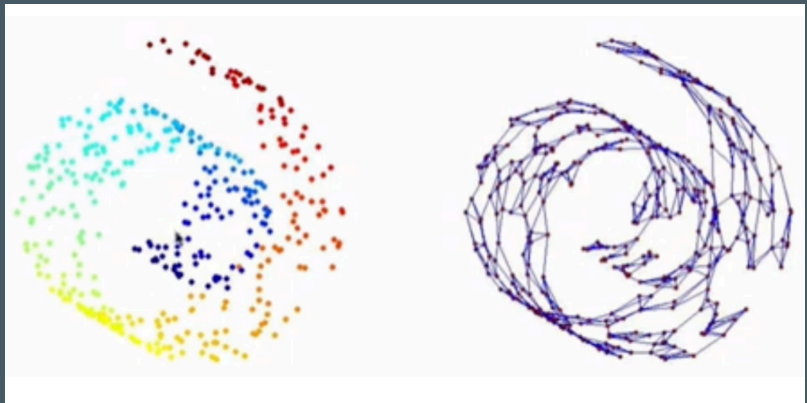
First Approach: Curved Data

- Use geodesic distances for large gaps
- Use Euclidean for small gaps
- Find low-dimensional embeddings preserving these distances



Isomap

- Build a k-nearest neighbor graph
- Use shortest path distances
- Matrix of pairwise distances D



Isomap – Goal

- The recorded distances are collected in a large matrix:

$$D = \begin{pmatrix} d_{11} & d_{12} & \dots & d_{1m} \\ \dots & & & \\ \dots & & & \\ d_{m1} & d_{m2} & \dots & d_{mm} \end{pmatrix}$$

- Then find $x_1, \dots, x_m$ in a space of lower dimension such that:

$$d(x_i, x_j) = D_{ij}$$

- This algorith is called Multi-Dimensional Scaling

Exercise: Isomap

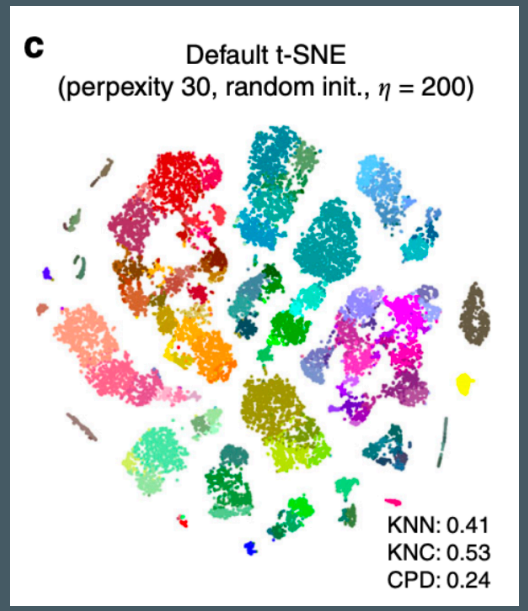
- Apply Isomap to Swiss Roll
- At what value of `n_neighbors` does it fail?

Other Manifold Learning Methods

- Locally Linear Embedding
- UMAP
- Laplacian Eigenmaps
- ...

Second Approach: Visual Clustering

- Identify clusters
- Good for visualization, not preprocessing



t-SNE

- Pronounced tee-Snee
- Embed high-dimensional data $x_i \in \mathbb{R}^d$ into $\mathbb{R}^2$
- Surround each Point x_i with a probability function. Then Define $p_{j|i}$: probability that x_j belongs to same cluster as x_i.
- Warning: This is not symmetric and strongly depends on your choice of parameters!
- Define a similarity function $q_{j|i}$ in 2D with t-distribution
- Now, try to make the two distributions as similar as possible.

t-SNE Formulas

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

- The parameters σ_i depend on the point x_i. Densely populated areas get a small σ, empty areas get large σ_i.
- Rough definition: Choose σ such that the perplexity of the density function fits to your chosen parameters

t-SNE Formulas

$$q_{j|i} = \frac{1}{1 + \|y_i - y_j\|^2} \sum_{k \neq i} \frac{1}{1 + \|y_i - y_k\|^2}$$

- Modelled with t-distribution, no parameters involved

t-SNE Heuristic

- Large $q_{i|j}$, small $p_{i|j}$: heavy penalty
- Small $q_{i|j}$, large $p_{i|j}$: light penalty
- Heuristically, all points have repelling forces between them, but you also introduce attracting forces between near points

Exercise: t-SNE

- Run t-SNE on first 500 MNIST samples
- Vary `perplexity` and visualize

t-SNE: Evaluation

Fun and beautiful plots
Danger of over-tuning and misinterpretation: there are way too many parameters

Usually, one tunes so long till one sees what was expected anyway. No real insight.

Cluster size and distances don'Pt have real meaning

Distances don't have real meaning.

Use carefully for exploration only