PCA: k Components

- Given data $\{\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}\} \in \mathbb{R}^d$, number of components k.
- Compute covariance matrix C, top $k \leq d$ eigenvectors $\vec{u}^{(1)}$, $\vec{u}^{(2)}, \ldots, \vec{u}^{(k)}$.
- For any vector $\vec{x} \in \mathbb{R}$, its new representation in $\mathbb{R}^k$ is $\vec{z} = (z_1, z_2, \ldots z_k)^T$, where:

$$
\left\{
\begin{aligned}
z_1 &= \vec{x} \cdot \vec{u}^{(1)} \\
z_2 &= \vec{x} \cdot \vec{u}^{(2)} \\
&\vdots \\
z_k &= \vec{x} \cdot \vec{u}^{(k)}
\end{aligned}
\right.
$$

Matrix Formulation

- Let X be the **data matrix** (n rows, d columns)
- Let U be matrix of the k eigenvectors as columns (d rows, k columns)
- The new representation: $Z = XU$

DSC 140B
Representation Learning

Lecture 12 | Part 1

Reconstructions

Reconstructing Points

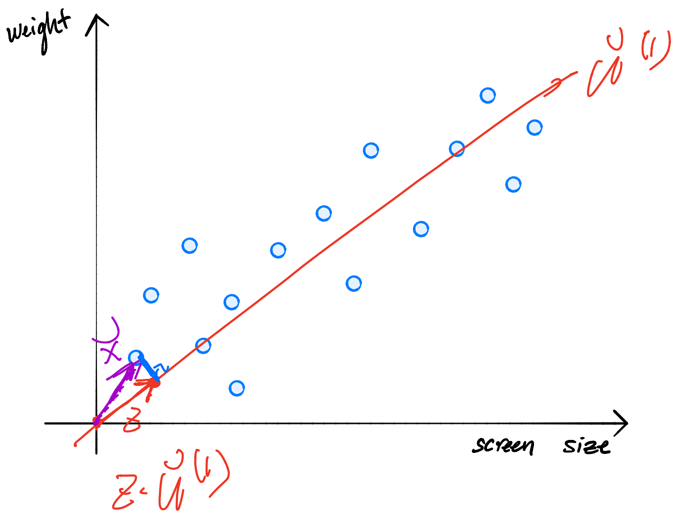
- PCA helps us reduce dimensionality from $\mathbb{R}^d \to R^k$
- Suppose we have the "new" representation in $\mathbb{R}^k$.
- Can we "go back" to $\mathbb{R}^d$?
- And why would we want to?

Back to $\mathbb{R}^d$

- Suppose new representation of $\vec{x}$ is z.

- $z = \vec{x} \cdot \vec{u}^{(1)}$

- Idea: $\vec{x} \approx z\vec{u}^{(1)}$



Reconstructions

- Given a "new" representation of $\vec{x}$, $\vec{z} = (z_1, \ldots, z_k) \in \mathbb{R}^k$
- And top k eigenvectors, $\vec{u}^{(1)}, \ldots, \vec{u}^{(k)}$
- The **reconstruction** of $\vec{x}$ is

$$z_1 \vec{u}^{(1)} + z_2 \vec{u}^{(2)} + \ldots + z_k \vec{u}^{(k)} = U\vec{z}$$

Reconstruction Error

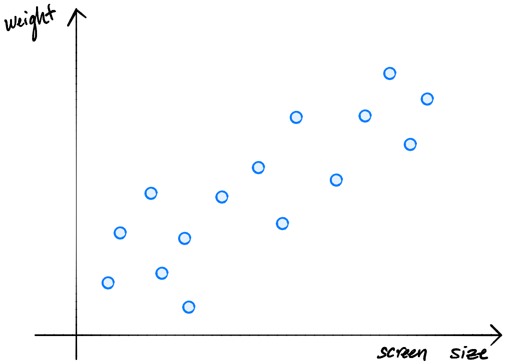
- The reconstruction *approximates* the original point, $\vec{x}$.

- The **reconstruction error** for a single point, $\vec{x}$:

$$\|\vec{x} - U\vec{z}\|^2$$

- Total reconstruction error:

$$\sum_{i=1}^{n} \|\vec{x}^{(i)} - U\vec{z}^{(i)}\|^2$$



DSC 140B

Representation Learning

Lecture 12 | Part 2

Interpreting PCA

Three Interpretations

- What is PCA doing?
- Three interpretations:
  1. Mazimizing variance
  2. Finding the best reconstruction
  3. Decorrelation

Recall: Matrix Formulation

$Z = XU$

- Given data matrix X.
- Compute new data matrix $Z = XU$.
- PCA: choose U to be matrix of eigenvectors of C.
- For now: suppose U can be anything – but columns should be orthonormal
  - Orthonormal = "not redundant"

View #1: Maximizing Variance

- This was the view we used to derive PCA

- Define the **total variance** to be the sum of the variances of each column of Z.

  $Z = X \cdot U$

- Claim: Choosing U to be top eigenvectors of C maximizes the total variance among all choices of orthonormal U.

Main Idea

PCA maximizes the total variance of the new data. I.e., chooses the most “interesting” new features which are not redundant.

View #2: Minimizing Reconstruction Error

supervised

unsupervised

> self supervised.

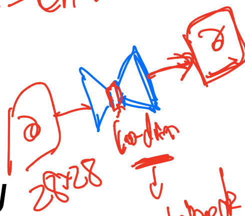
CoTB

auto-encoder

- Recall: total reconstruction error $NN_\theta(\vec{x}^{(i)})$

$$\sum_{i=1}^{n} \|\vec{x}^{(i)} - U\vec{z}^{(i)}\|^2$$

- Goal: minimize total reconstruction error.
- Claim: Choosing U to be top eigenvectors of C minimizes reconstruction error among all choices of orthonormal U



AE. bottleneck

$k < d$.

tradeoff

$k \downarrow$

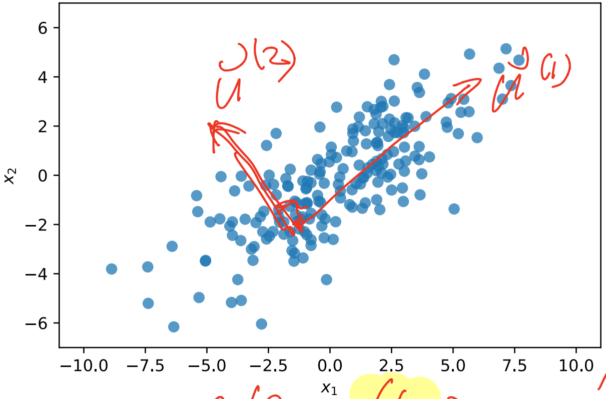
Main Idea

PCA minimizes the reconstruction error. It is the "best" projection of points onto a linear subspace of dimensionality k. When $k = d$, the reconstruction error is zero.

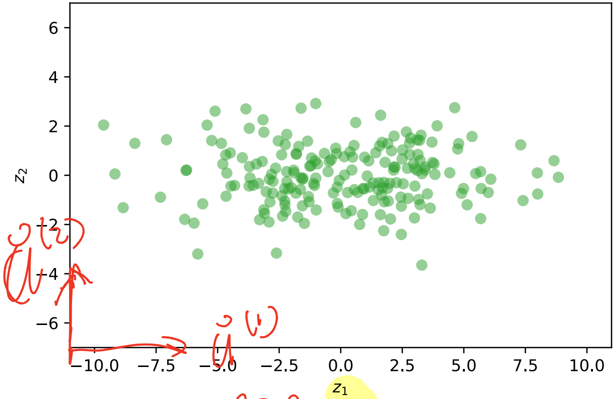
View #3: Decorrelation

- PCA has the effect of "decorrelating" the features.

x_2



x_1



$$C = \begin{pmatrix} 10 & 4 \\ 4 & 5 \end{pmatrix} \qquad C' = \begin{pmatrix} 20 & 0 \\ 0 & 10 \end{pmatrix}$$

Main Idea

PCA learns a new representation by rotating the data into a basis where the features are uncorrelated (not redundant). That is: the natural basis vectors are the principal directions (eigenvectors of the covariance matrix). PCA changes the basis to this natural basis.

DSC 140B
Representation Learning

Lecture 12 | Part 3

PCA in Practice

representation learning

PCA in Practice

- PCA is often used in **preprocessing** before classifier is trained, etc.
- Must choose number of dimensions, k. 8

hyper parameter

- One way: cross-validation.
- Another way: the elbow method.

variance in direction $\vec{u}$

$\vec{u}^{(1)}$ λ_1

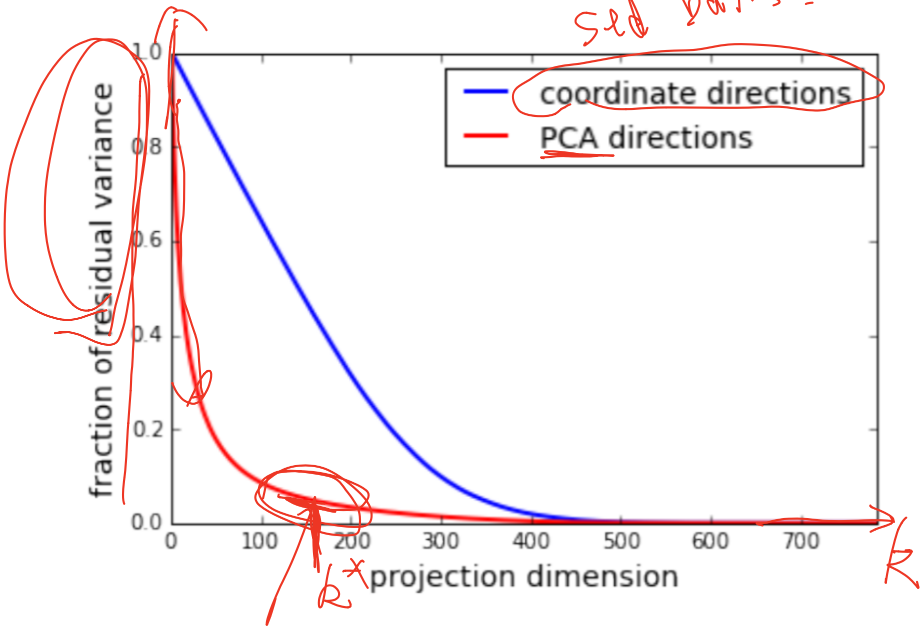
$\vec{u}^T C \vec{u} \longrightarrow \vec{u}^{(1)} C \vec{u}^{(1)} = \vec{u}^{(1)} \cdot \lambda_1 \vec{u}^{(1)} = \lambda_1$

Total Variance

- The **total variance** is the sum of the eigenvalues of the covariance matrix.
- Or, alternatively, sum of variances in each orthogonal basis direction.

natural basis direction

std basis.



elbow

$$\sum_{i=1}^{d} \lambda_i \quad C$$

k

$k=1$ $\sum_{i=1}^{d} \lambda_i$

$k \uparrow$ $\sum_{i=k+1}^{d} \lambda_i$

① centering $\vec{x}_i - \vec{\mu}$

② standardization $\dfrac{x_i^{(j)} - \mu^{(j)}}{s^{(j)}}$

Caution

kg g km m

x_1 1 1000 dominate variance

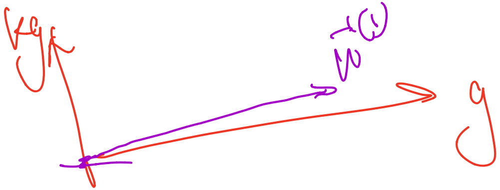
x_2 10 100 $s^{(2)}$

- PCA's assumption: variance is interesting

$s^{(j)}$: std deviation of feature j

$s^{(j)} = \sqrt{\sigma^{(j)}}$

- PCA is totally unsupervised
- The direction most meaningful for classification may not have large variance!



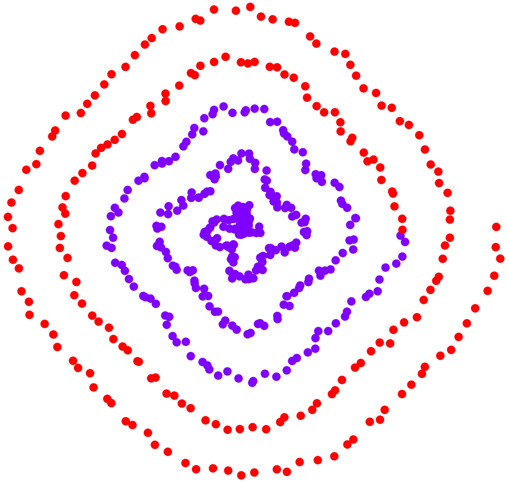
DSC 140B
Representation Learning

Lecture 12 | Part 4

Nonlinear Dimensionality Reduction

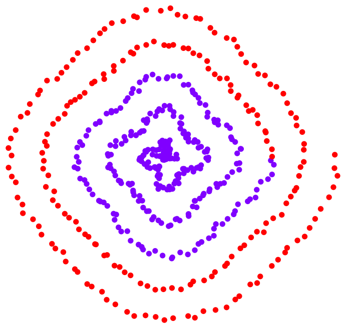
Scenario

- You want to train a classifier on this data.
- It would be easier if we could “unroll” the spiral.
- Data seems to be one-dimensional, even though in two dimensions.
- Dimensionality reduction?



PCA?

- Does PCA work here?
- Try projecting onto one principal component.



No

PCA?

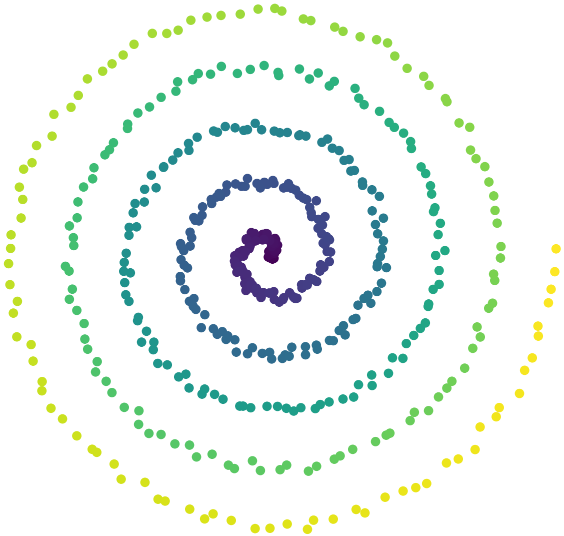
- PCA simply “rotates” the data.
- No amount of rotation will “unroll” the spiral.
- We need a fundamentally different approach that works for non-linear patterns.

Today

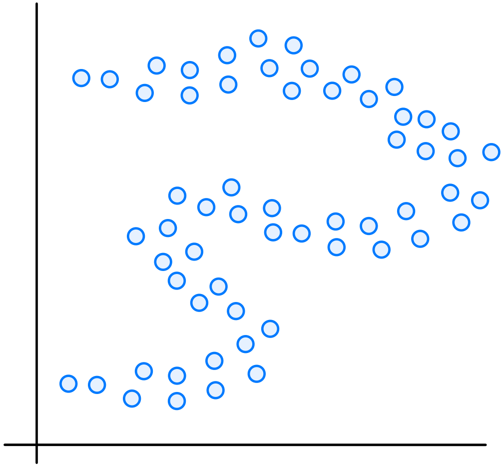
- Non-linear dimensionality reduction via **spectral embeddings**.

Rethinking Dimensionality

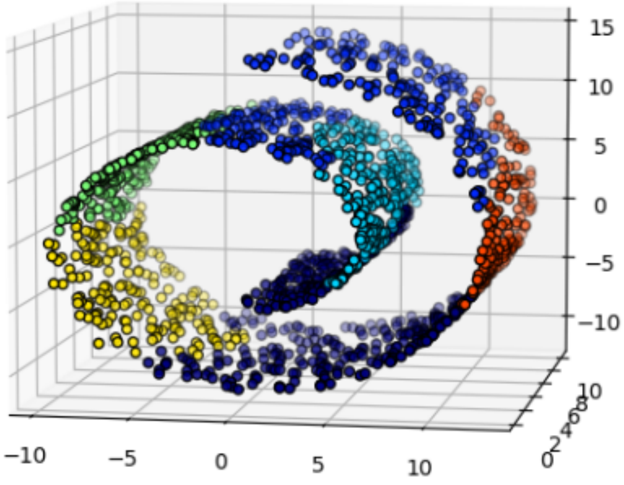
- Each point is an (x, y) coordinate in two dimensional space
- But the structure is one-dimensional
- Could (roughly) locate point using one number: distance from end.



Rethinking Dimensionality



Rethinking Dimensionality

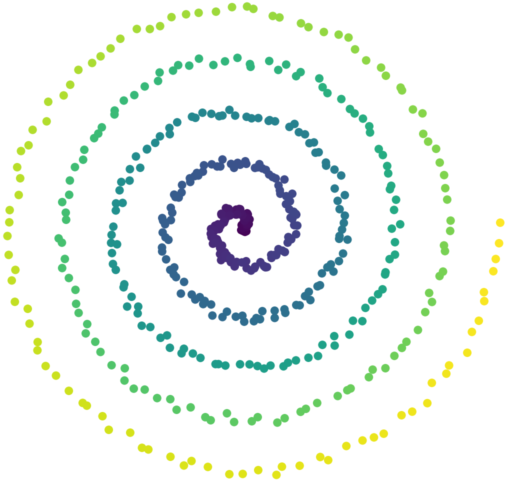


Rethinking Dimensionality

- Informally: data expressed with d dimensions, but its *really* confined to k-dimensional region
- This region is called a **manifold**
- d is the **ambient** dimension
- k is the **intrinsic** dimension

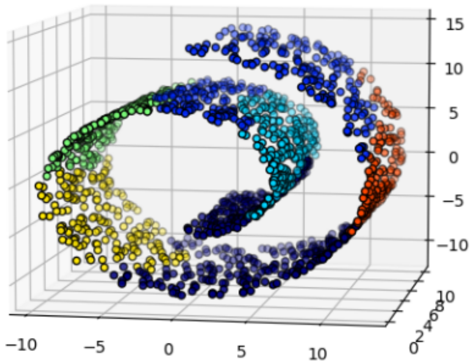
Example

- Ambient dimension: 2
- Intrinsic dimension: 1



Example

- Ambient dimension: 3
- Intrinsic dimension: 2



Example

- Ambient dimension:
- Intrinsic dimension:

Manifold Learning

- **Given**: data in high dimensions
- **Recover**: the low-dimensional manifold

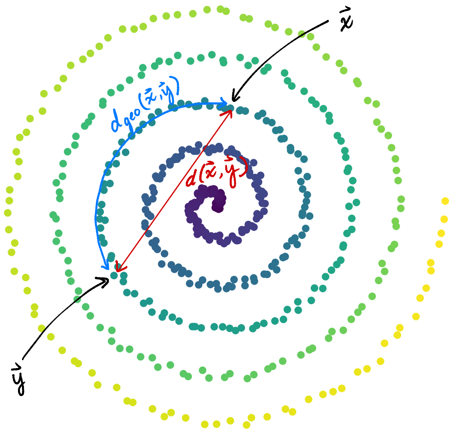
Types of Manifolds

- Manifolds can be linear
  - E.g., linear subpaces – hyperplanes
  - Learned by PCA

- Can also be non-linear (locally linear)
  - Example: the spiral data
  - Learned by **Laplacian eigenmaps**, among others

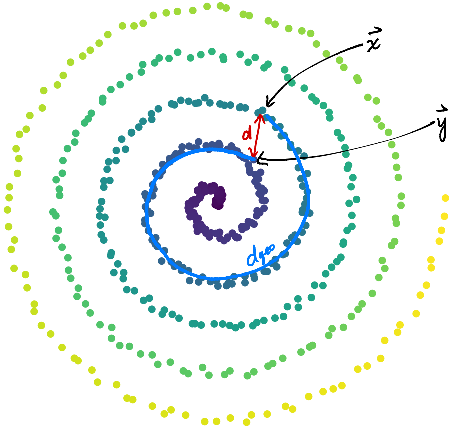
Euclidean vs. Geodesic Distances

- **Euclidean distance**: the “straight-line” distance
- **Geodesic distance**: the distance along the manifold



Euclidean vs. Geodesic Distances

- **Euclidean distance**: the “straight-line” distance
- **Geodesic distance**: the distance along the manifold



Euclidean vs. Geodesic Distances

- If data is close to a linear manifold, geodesic ≈ Euclidean

- Otherwise, can be very different

Non-Linear Dimensionality Reduction

- **Goal**: Map points in $\mathbb{R}^d$ to $\mathbb{R}^k$
- **Such that**: if $\vec{x}$ and $\vec{y}$ are close in **geodesic** distance in $\mathbb{R}^d$, they are close in **Euclidean** distance in $\mathbb{R}^k$

Embeddings