

DSC 140B

Representation Learning

Lecture 08 | Part 1

Interpreting PCA

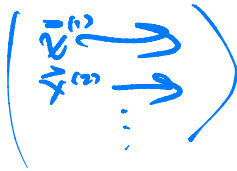
Three Interpretations

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

$U \vec{x}$

Recall: Matrix Formulation

- ▶ 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”



change of basis matrix

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

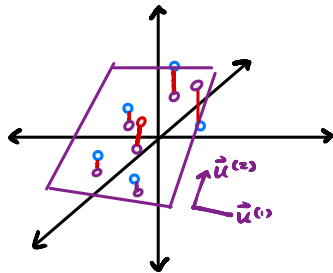
Main Idea

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

- ▶ Recall: total reconstruction error

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



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

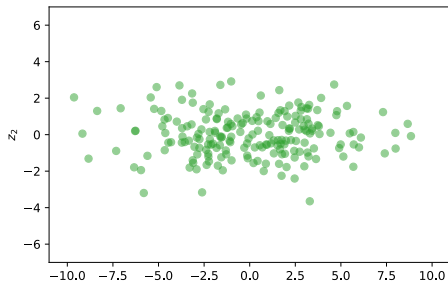
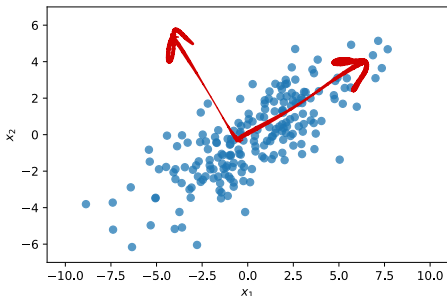
Main Idea

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.

$$(AB)^T = B^T A^T$$

View #3: Decorrelation

- ▶ PCA has the effect of “decorrelating” the features.



$$C_z = \frac{1}{n} Z^T Z = \frac{1}{n} (XU)^T XU = \frac{1}{n} U^T X^T X U = \underbrace{U^T C U}_{\text{diagonal}}$$
$$Z = XU$$

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 08 | Part 2

PCA in Practice

PCA in Practice

- ▶ PCA is often used in **preprocessing** before classifier is trained, etc.
- ▶ Must choose number of dimensions, k .
- ▶ One way: cross-validation.
- ▶ Another way: the elbow method.

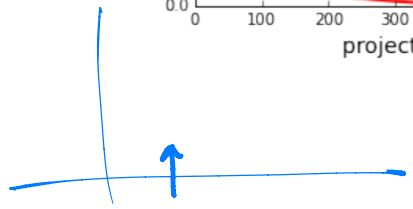
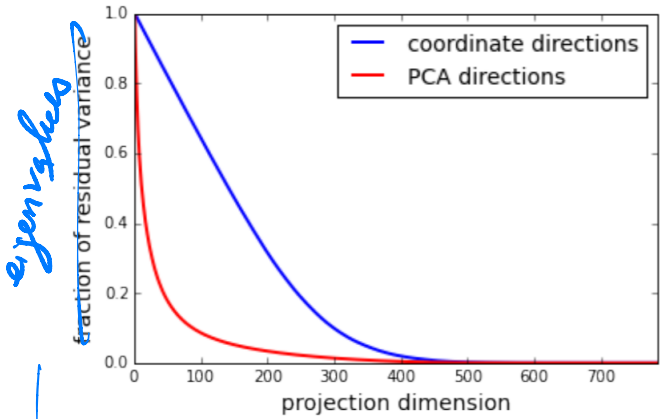
$\hat{u}^T C \hat{u} = \text{variance in direction of } \hat{u}.$

$$\vec{u}^{(i)} \lambda_i$$

Total Variance

$$\begin{aligned} & \vec{u}^{(i)T} C \vec{u}^{(i)} \\ &= \vec{u}^{(i)T} \lambda_i \vec{u}^{(i)} = \lambda_i \end{aligned}$$

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



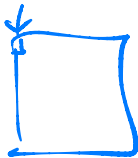
Standardize the features?

Km	Km
1570	.72
830	1.3
⋮	2.1

Caution

pixel 1	pixel 2
255	1
130	5
70	4

- ▶ PCA's assumption: variance is interesting
- ▶ PCA is totally unsupervised
- ▶ The direction most meaningful for classification may not have large variance!



DSC 140B

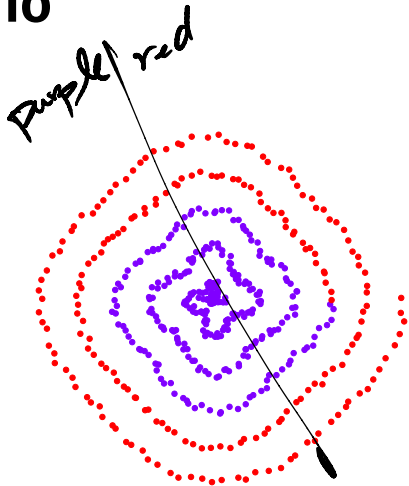
Representation Learning

Lecture 08 | Part 3

Nonlinear Dimensionality Reduction

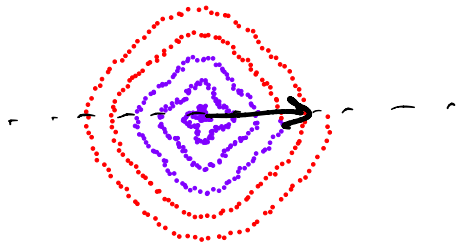
Scenario

- ▶ 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?

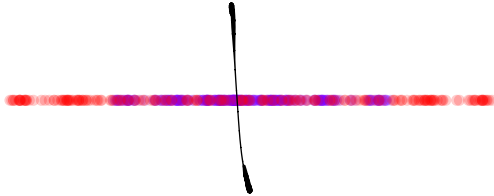


PCA?

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



No



PCA?

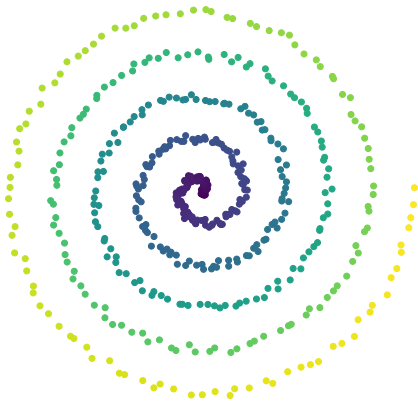
- ▶ PCA simply “rotates” the data.
- ▶ No amount of rotation will “unroll” the spiral.
- ▶ We need a fundamentally different approach that works for non-linear patterns.

Today

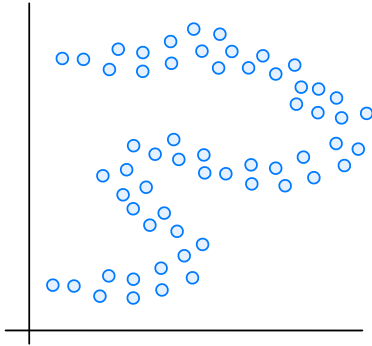
- ▶ Non-linear dimensionality reduction via **spectral embeddings**.

Rethinking Dimensionality

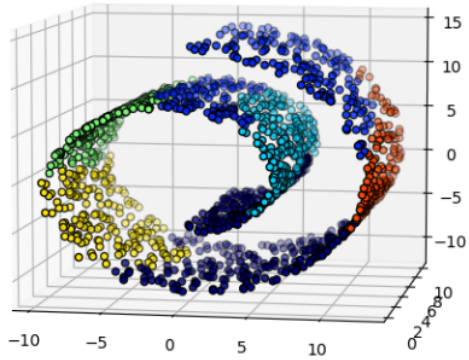
- ▶ Each point is an (x, y) coordinate in two dimensional space
- ▶ But the structure is one-dimensional
- ▶ Could (roughly) locate point using one number: distance from end.



Rethinking Dimensionality



Rethinking Dimensionality

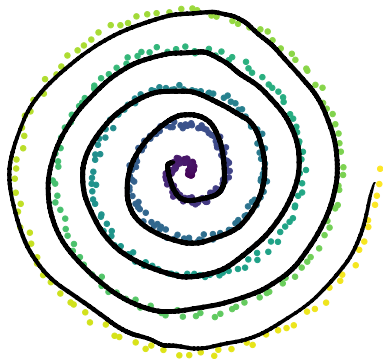


Rethinking Dimensionality

- ▶ 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

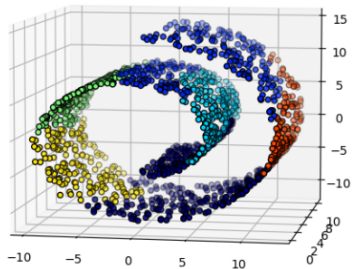
Example

- ▶ Ambient dimension: 2
- ▶ Intrinsic dimension: 1



Example

- ▶ Ambient dimension: 3
- ▶ Intrinsic dimension: 2



Example

- ▶ Ambient dimension: 3
- ▶ Intrinsic dimension: 2



Manifold Learning

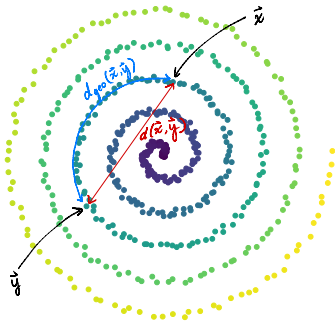
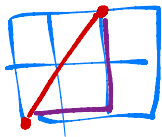
- ▶ **Given:** data in high dimensions
- ▶ **Recover:** the low-dimensional manifold

Types of Manifolds

- ▶ Manifolds can be linear
 - ▶ E.g., linear subspaces – hyperplanes
 - ▶ Learned by PCA
- ▶ Can also be non-linear (locally linear)
 - ▶ Example: the spiral data
 - ▶ Learned by **Laplacian eigenmaps**, among others

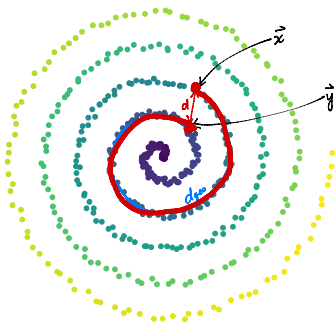
Euclidean vs. Geodesic Distances

- ▶ **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



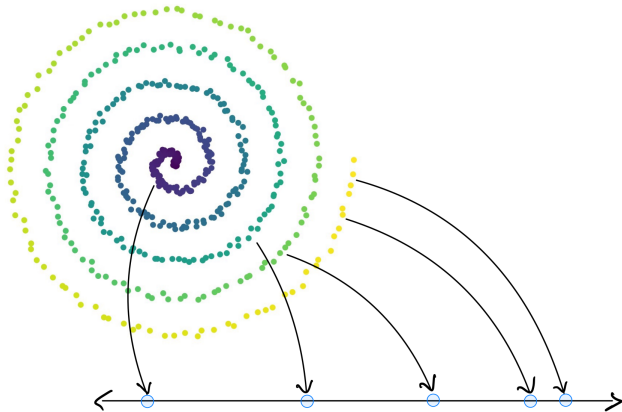
Euclidean vs. Geodesic Distances

- ▶ If data is close to a linear manifold, geodesic \approx 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



DSC 140B

Representation Learning

Lecture 08 | Part 4

Embedding Similarities

Similar Netflix Users

- ▶ Suppose you are a data scientist at Netflix
- ▶ You're given an $n \times n$ **similarity matrix** W of users
 - ▶ entry (i, j) tells you how *similar* user i and user j are
 - ▶ 1 means “very similar”, 0 means “not at all”
- ▶ Goal: visualize to find patterns

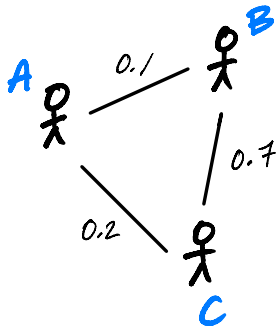
Idea

- ▶ We like scatter plots. Can we make one?
- ▶ Users are **not** vectors / points!
- ▶ They are nodes in a **similarity graph**

Similarity Graphs

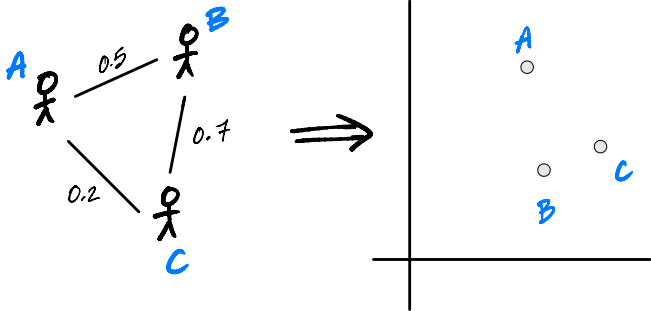
- ▶ Similarity matrices can be thought of as weighted graphs, and *vice versa*.

$$\begin{matrix} & \begin{matrix} A & B & C \end{matrix} \\ \begin{matrix} A \\ B \\ C \end{matrix} & \begin{pmatrix} 1 & 0.1 & 0.2 \\ 0.1 & 1 & 0.7 \\ 0.2 & 0.7 & 1 \end{pmatrix} \end{matrix}$$



Goal

- ▶ **Embed** nodes of a similarity graph as points.
- ▶ Similar nodes should map to nearby points.



Today

- ▶ We will design a graph embedding approach:
 - ▶ **Spectral embeddings** via **Laplacian eigenmaps**

More Formally

- ▶ **Given:**
 - ▶ A **similarity graph** with n nodes
 - ▶ a number of dimensions, k
- ▶ **Compute:** an **embedding** of the n points into \mathbb{R}^k so that similar objects are placed nearby

To Start

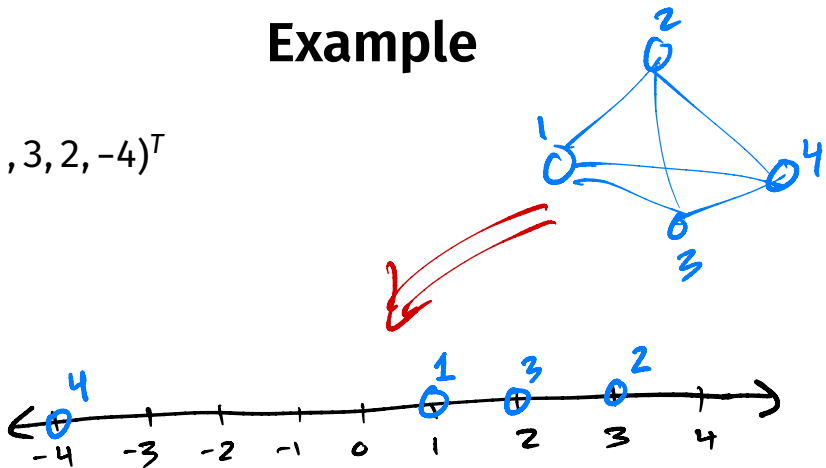
- ▶ **Given:**
 - ▶ A **similarity graph** with n nodes
- ▶ **Compute:** an **embedding** of the n points into \mathbb{R}^1 so that similar objects are placed nearby

Vectors as Embeddings into \mathbb{R}^1

- ▶ Suppose we have n nodes (objects) to embed
- ▶ Assume they are numbered $1, 2, \dots, n$
- ▶ Let $f_1, f_2, \dots, f_n \in \mathbb{R}$ be the embeddings
- ▶ We can pack them all into a vector: \vec{f} .
- ▶ Goal: find a good set of embeddings, \vec{f} .

Example

$$\vec{f} = (1, 3, 2, -4)^T$$

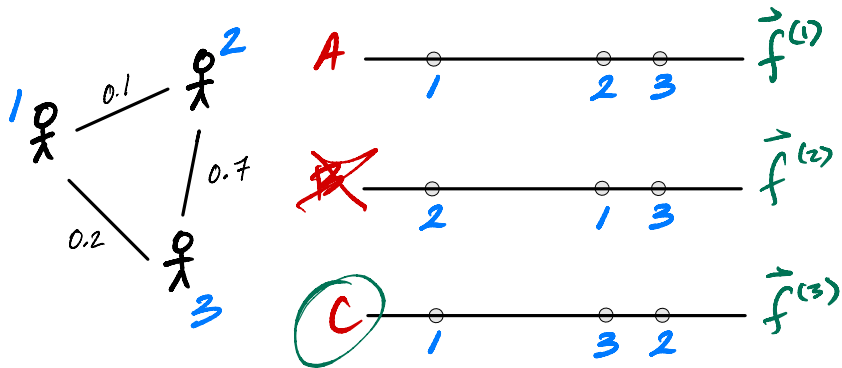


An Optimization Problem

- ▶ We'll turn it into an optimization problem:
- ▶ **Step 1:** Design a cost function quantifying how good a particular embedding \vec{f} is
- ▶ **Step 2:** Minimize the cost

Example

- Which is the best embedding?



Cost Function for Embeddings

- ▶ Idea: cost is low if similar points are close
- ▶ Here is one approach:

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} (f_i - f_j)^2$$

- ▶ where w_{ij} is the weight between i and j .

Interpreting the Cost

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} (f_i - f_j)^2$$

- ▶ If $w_{ij} \approx 0$, that pair can be placed very far apart without increasing cost
- ▶ If $w_{ij} \approx 1$, the pair should be placed close together in order to have small cost.

Exercise

Do you see a problem with the cost function?

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} (f_i - f_j)^2$$

Hint: what embedding \vec{f} minimizes it?

Problem

- ▶ The cost is **always** minimized by taking $\vec{f} = 0$.
- ▶ This is a “**trivial**” solution. Not useful.
- ▶ **Fix:** require $\|\vec{f}\| = 1$
 - ▶ Really, any number would work. 1 is convenient.

Exercise

Do you see **another** problem with the cost function, even if we require \vec{f} to be a unit vector?

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} (f_i - f_j)^2$$

Hint: what other choice of \vec{f} will **always** make this zero?

Problem

- ▶ The cost is **always** minimized by taking $\vec{f} = \frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T$.
- ▶ This is a “**trivial**” solution. Again, not useful.
- ▶ **Fix:** require \vec{f} to be orthogonal to $(1, 1, \dots, 1)^T$.
 - ▶ Written: $\vec{f} \perp (1, 1, \dots, 1)^T$
 - ▶ Ensures that solution is not close to trivial solution
 - ▶ Might seem strange, but it will work!

The New Optimization Problem

- ▶ **Given:** an $n \times n$ similarity matrix W
- ▶ **Compute:** embedding vector \vec{f} minimizing

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} (f_i - f_j)^2$$

subject to $\|\vec{f}\| = 1$ and $\vec{f} \perp (1, 1, \dots, 1)^T$

How?

- ▶ This looks difficult.
- ▶ Let's write it in matrix form.
- ▶ We'll see that it is actually (hopefully) familiar.

How?

- ▶ This looks difficult.
- ▶ Let's write it in matrix form.
- ▶ We'll see that it is actually (hopefully) familiar.