ENGR-E 511; ENGR-E 399

Machine Learning for Signal Processing

Module 08:

Nonlinear Methods

Minje Kim

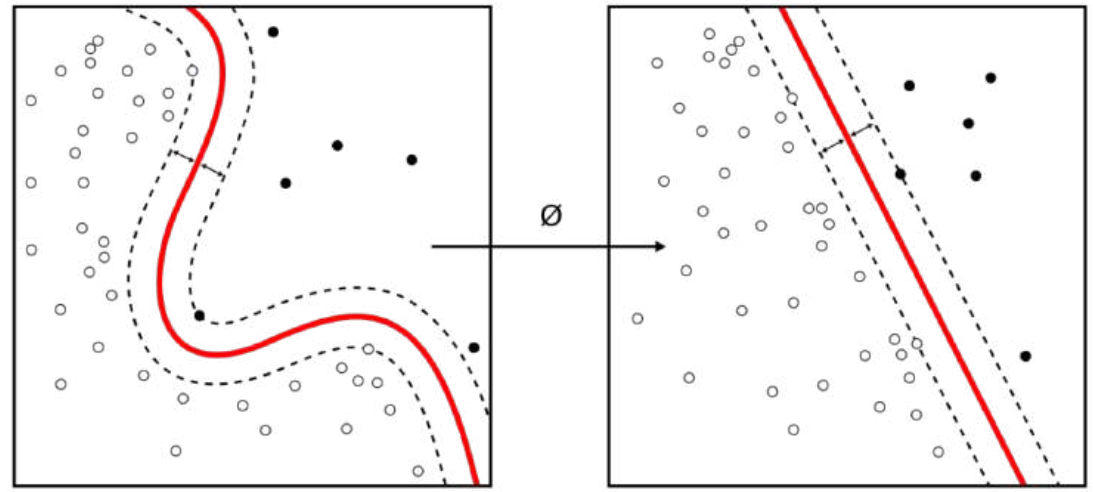
Department of Intelligent Systems Engineering

Email: minje@indiana.edu

Website: http://minjekim.com

Research Group: http://saige.sice.indiana.edu

Meeting Request: http://doodle.com/minje



https://en.wikipedia.org/wiki/Kernel_method

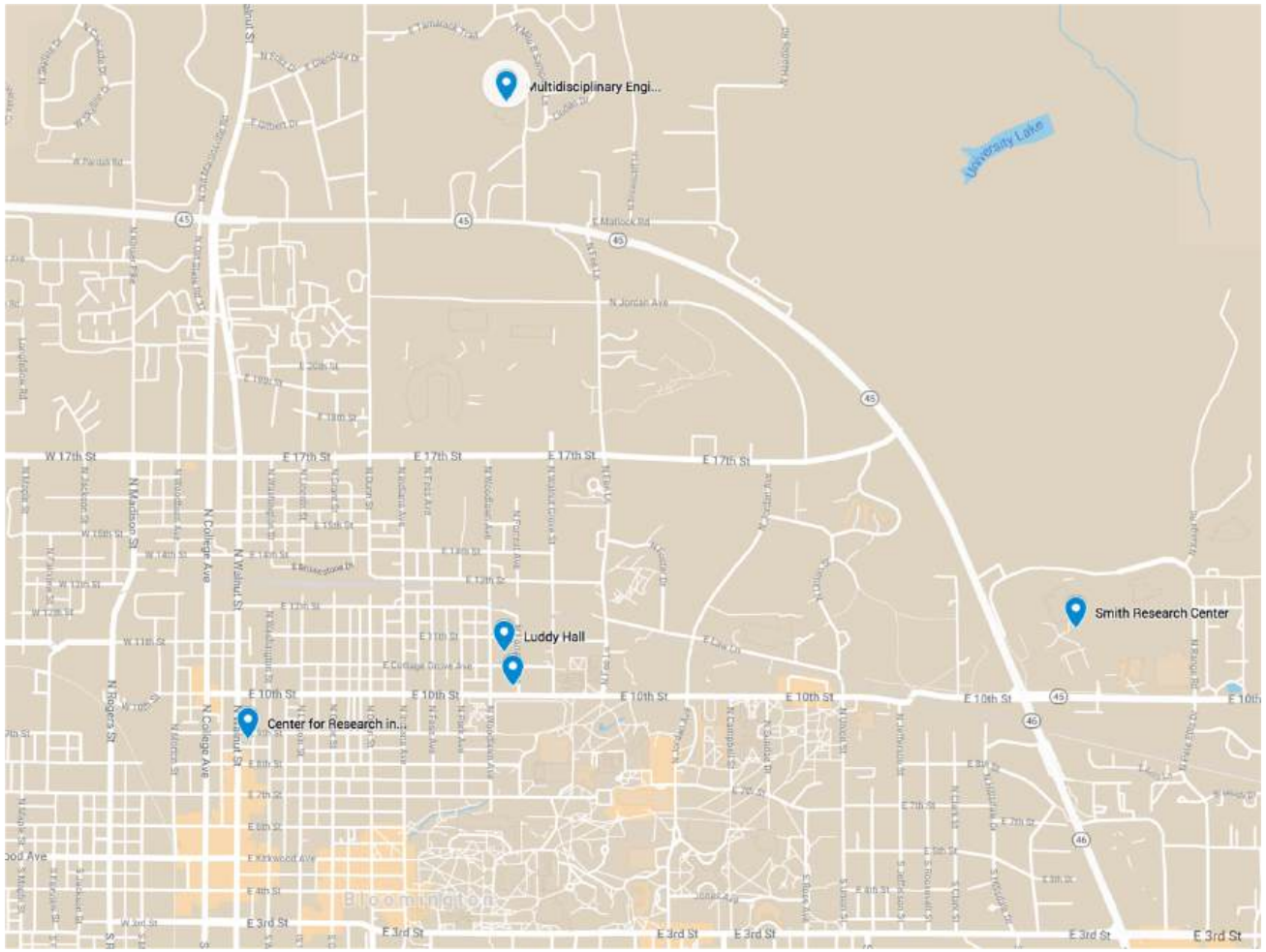
INDIANA UNIVERSITY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

A Motivating (Linear) Example

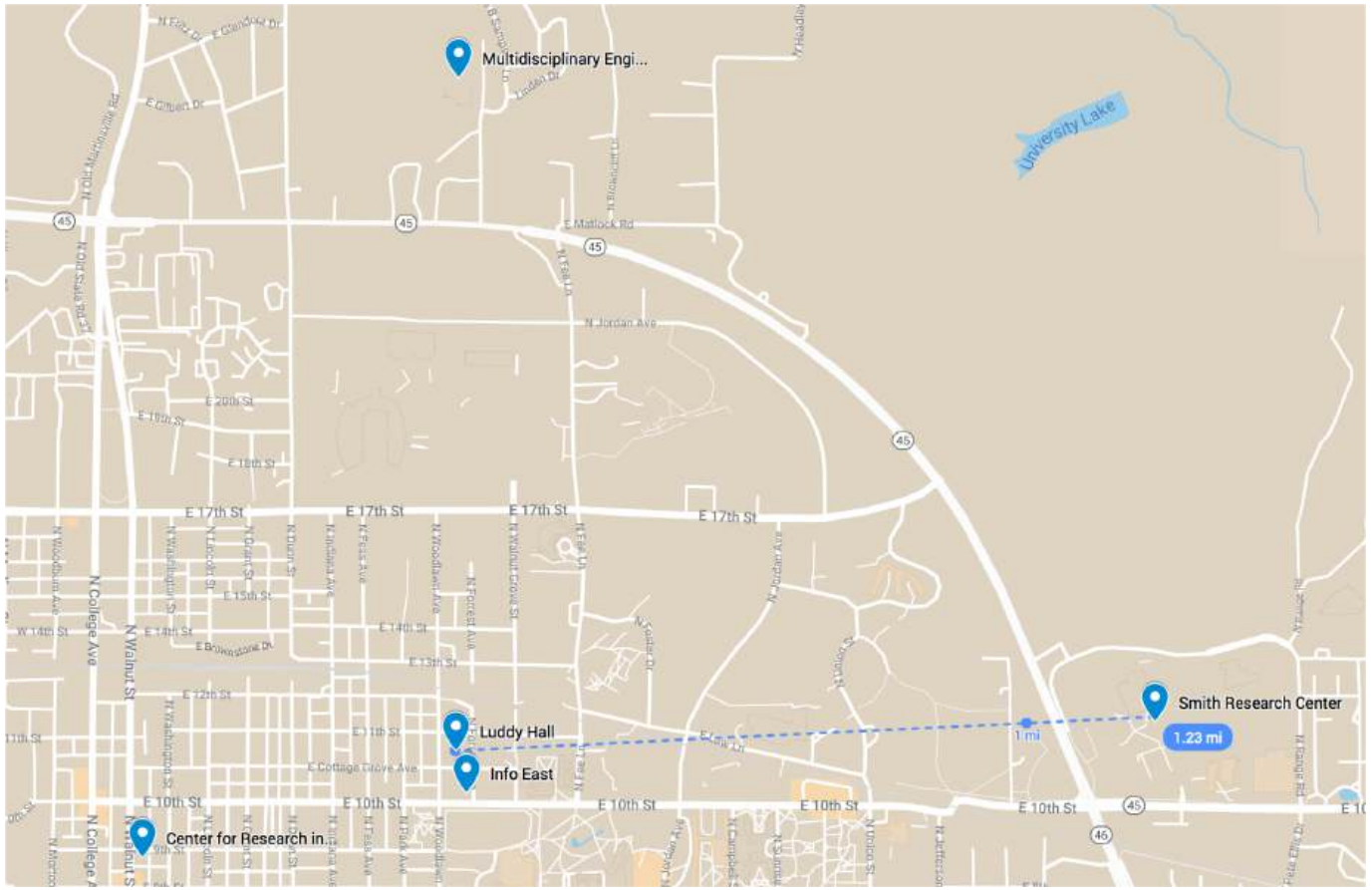
- SICE is dispersed

- I'm finally moved to Luddy Hall
  - Yet, SICE is dispersed
- What if I don't know their locations but their pairwise distance?
  - Can I still recover the map?



A Motivating (Linear) Example

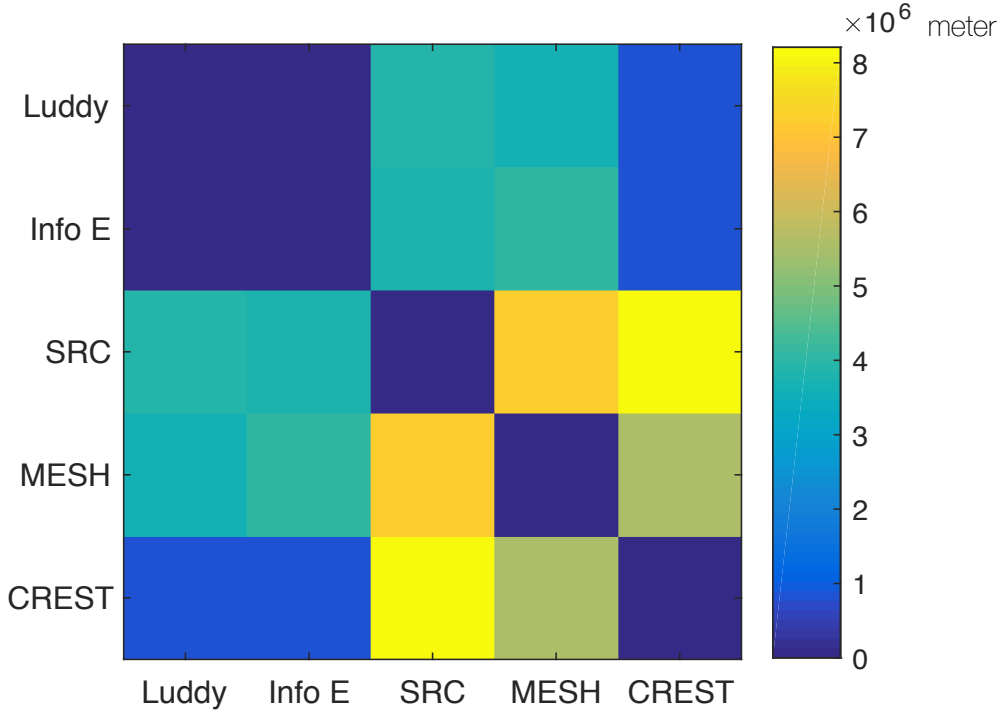
- SICE is dispersed

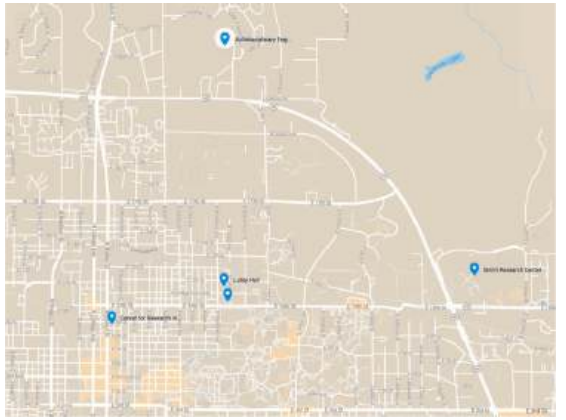


A Motivating (Linear) Example

- SICE is dispersed

- From now on I pretend like I don't know their locations
- Instead, I prepare a pairwise distance matrix





- From this, I want to recover their locations

MultiDimensional Scaling (MDS)

- A linear model

- The (squared) Euclidean pairwise distance matrix

$$\boldsymbol{M}_{ij} = ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top (\boldsymbol{x}_i - \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_i + \boldsymbol{x}_j^\top \boldsymbol{x}_j - 2\boldsymbol{x}_i^\top \boldsymbol{x}_j$$

  - If it were not for the quadratic terms, we can say $\boldsymbol{W} \approx \boldsymbol{y}_i^\top \boldsymbol{y}_j$ ← Inner product between ESTIMATED locations (similarity)

- So, the goal of MDS is to transform $\boldsymbol{M}$ into $\boldsymbol{W}$ by eliminating $\boldsymbol{x}_i^\top \boldsymbol{x}_i + \boldsymbol{x}_j^\top \boldsymbol{x}_j$
  - Suppose the average of $\boldsymbol{M}$ along the row

$$\overline{\boldsymbol{M}} = \frac{1}{N}\sum_j \boldsymbol{M}_{ij} = (N\boldsymbol{x}_i^\top \boldsymbol{x}_i + \sum_j \boldsymbol{x}_j^\top \boldsymbol{x}_j - 2\boldsymbol{x}_i^\top \sum_j \boldsymbol{x}_j)/N$$

  - Then, let's subtract it from $\boldsymbol{M}$

$$\tilde{\boldsymbol{M}}_{ij} = \boldsymbol{M}_{ij} - \overline{\boldsymbol{M}}_i = \boldsymbol{x}_j^\top \boldsymbol{x}_j - \frac{1}{N}\sum_j \boldsymbol{x}_j^\top \boldsymbol{x}_j - 2\boldsymbol{x}_i^\top \boldsymbol{x}_j + 2\boldsymbol{x}_i^\top \bar{\boldsymbol{x}}$$

  - Suppose the average of $\tilde{\boldsymbol{M}}$ along the column

$$|\tilde{\boldsymbol{M}}_{ij} = \boldsymbol{x}_j^\top \boldsymbol{x}_j - \frac{1}{N}\sum_j \boldsymbol{x}_j^\top \boldsymbol{x}_j - 2\bar{\boldsymbol{x}}^\top \boldsymbol{x}_j + 2\bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}}$$

  - Then, let's subtract it from $\tilde{\boldsymbol{M}}$

$$\boldsymbol{W}_{ij} = \tilde{\boldsymbol{M}}_{ij} - |\tilde{\boldsymbol{M}}_j = -2\boldsymbol{x}_i^\top \boldsymbol{x}_j + 2\boldsymbol{x}_i^\top \bar{\boldsymbol{x}} + 2\bar{\boldsymbol{x}}^\top \boldsymbol{x}_j - 2\bar{\boldsymbol{x}}^\top \bar{\boldsymbol{x}} = -2(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top (\boldsymbol{x}_j - \bar{\boldsymbol{x}})$$

MultiDimensional Scaling (MDS)

- A linear model

- We start from the distance matrix M
  - Subtract the average of rows and columns sequentially
- As a result we get a matrix W
  - Inner products of centered data samples (let's discard -2 from the previous slide)

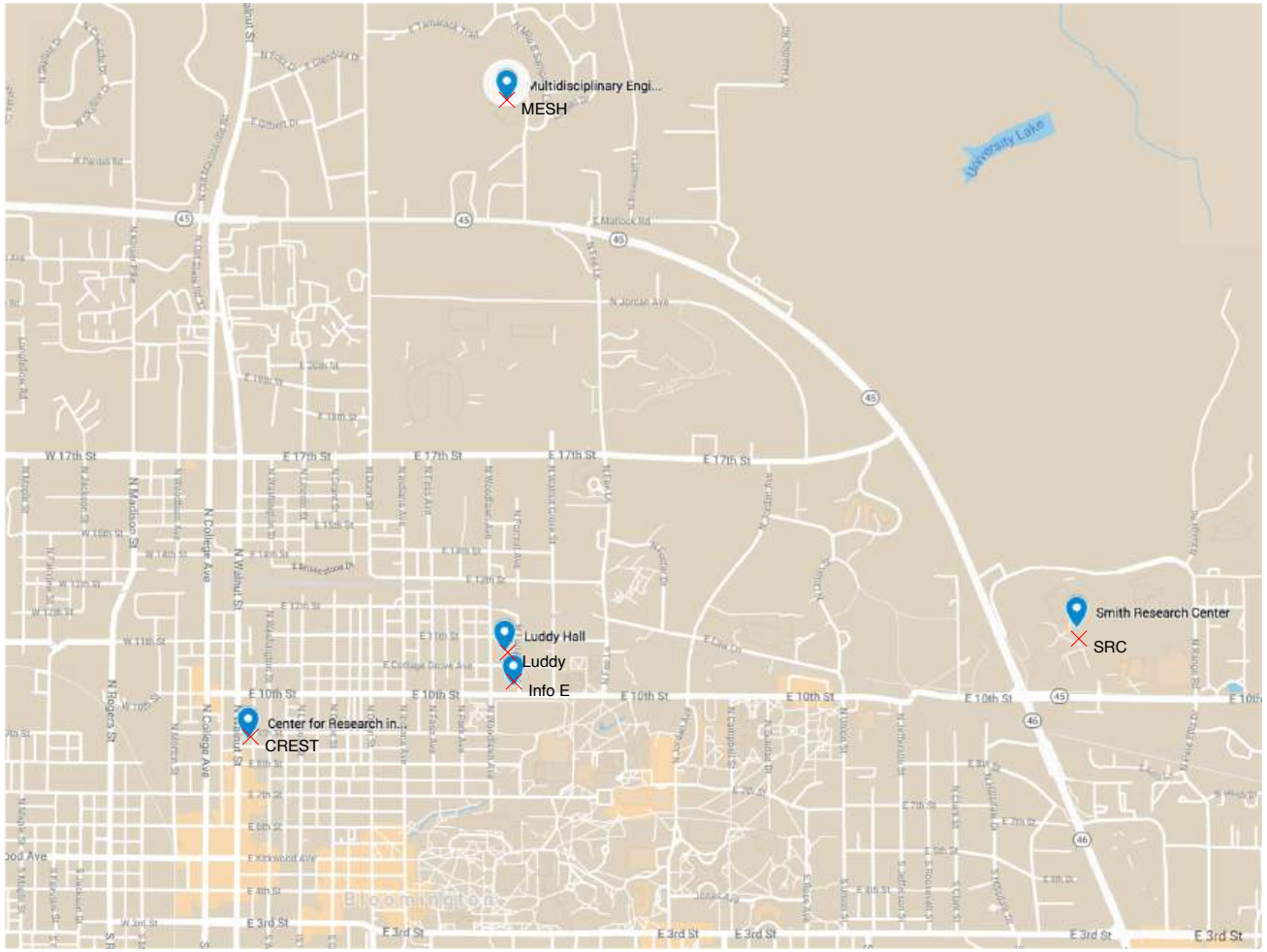
$$W_{ij} = (x_i - \bar{x})^\top (x_j - \bar{x}) \qquad W = Y^\top Y \qquad Y = X - \bar{x}$$

- So what?
  - W is a symmetric positive definite matrix
  - If we can decompose it into two factors, we're done. Any ideas?
  - Eigendecomposition! $W = UVU^\top = \left(UV^{\frac{1}{2}}\right)\left(V^{\frac{1}{2}}U^\top\right)$
- **The recipe for MDS**
  - Convert your pairwise distance matrix M into a positive definite matrix W
  - Do eigendecomposition on the matrix W
  - First few vectors of $UV^{\frac{1}{2}}$ holds recovers the locations (coordinates)

MDS on SICE Buildings

- Using two eigenvectors

- Note that there are rotation and shift ambiguity in MDS solutions

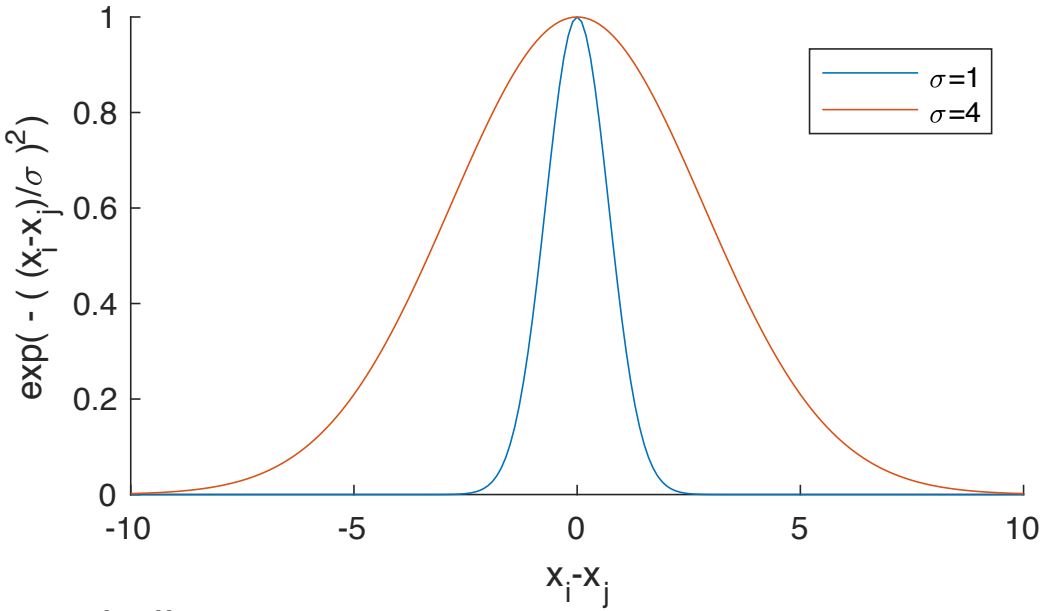


Euclidean Distance?

- MDS and beyond

- Inner products can define some kind of affinity
- How about the other types of affinity?
- A popular choice:

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{\sigma^2}\right)$$

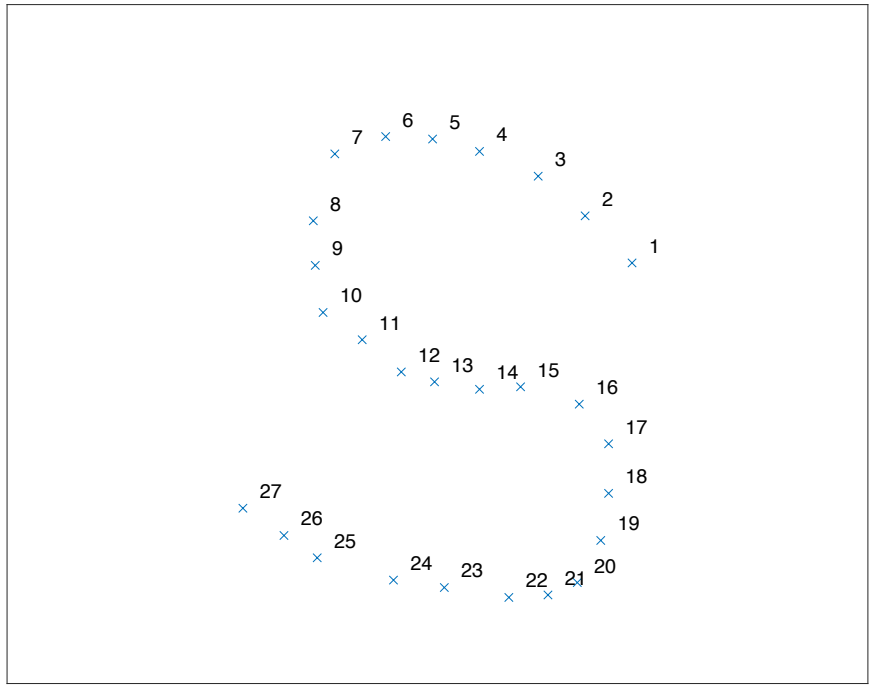


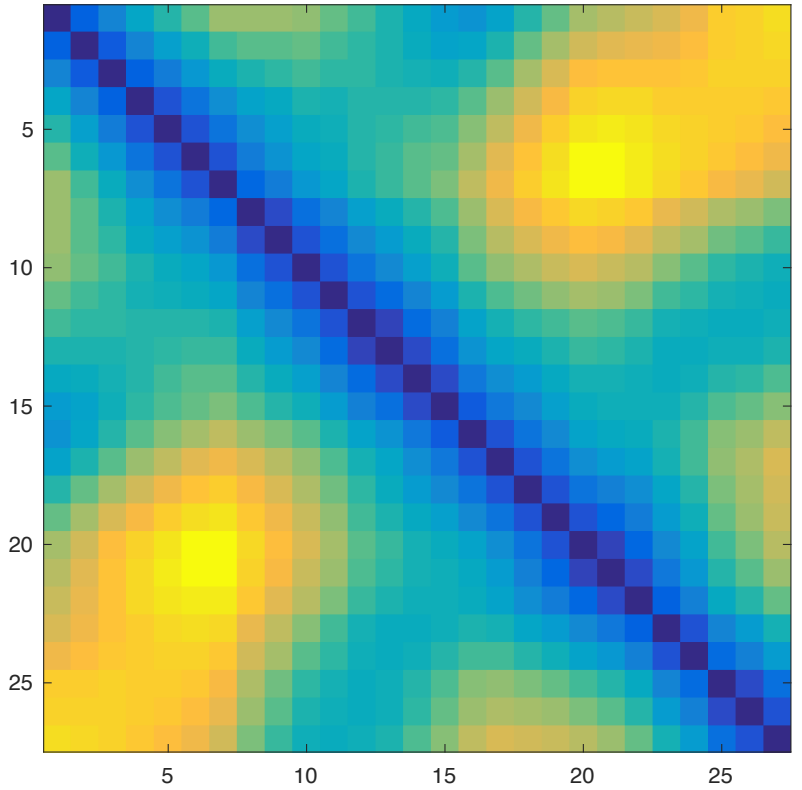
- If σ is small, closer data points become more similar
  - It weighs more on the closer relationships while suppressing farther ones
  - We call it **Radial Basis Function (RBF)**

Euclidean Distance?

- MDS and beyond

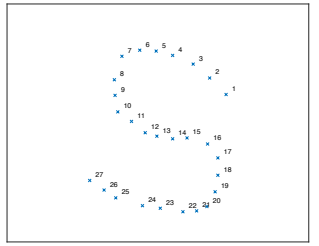
- An “S” shape and the pairwise Euclidean distance matrix



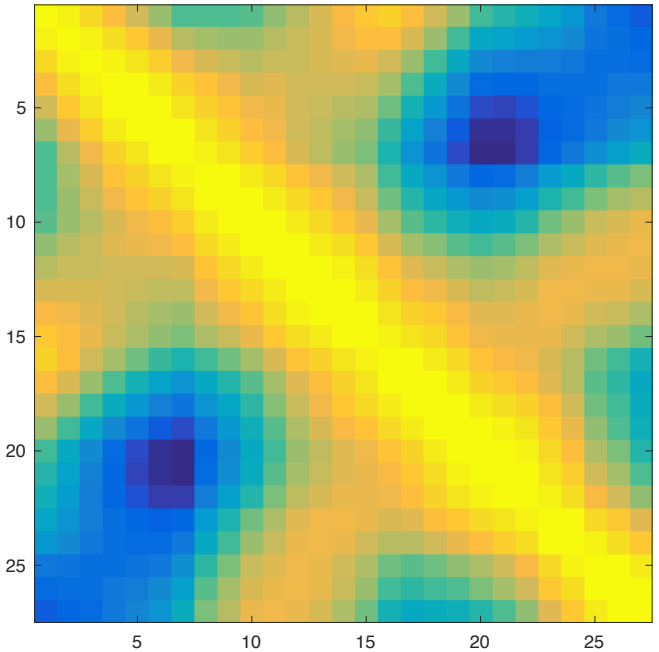


Euclidean Distance?

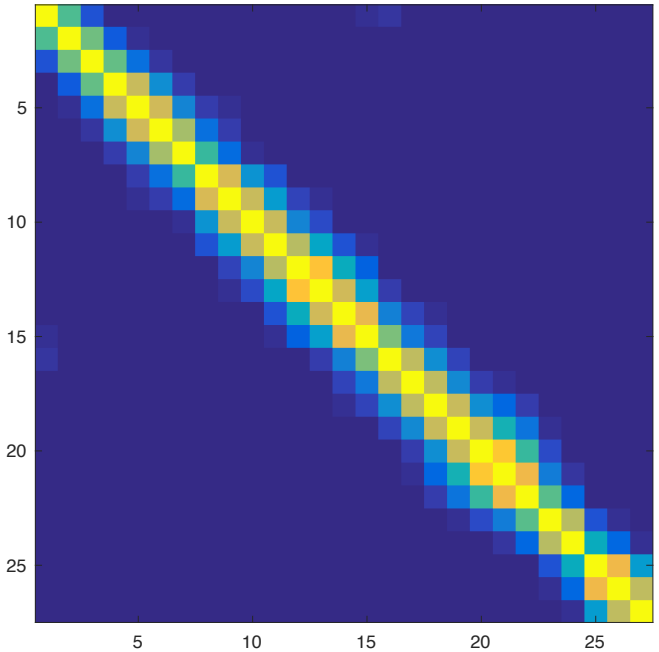
- MDS and beyond



- The affinity matrix with RBF kernels



$\sigma = 1$



$\sigma = 0.1$

- The narrower kernel preserves the local affinity better
  - While largely ignoring the global (linear) structure

Nonlinear Similarity

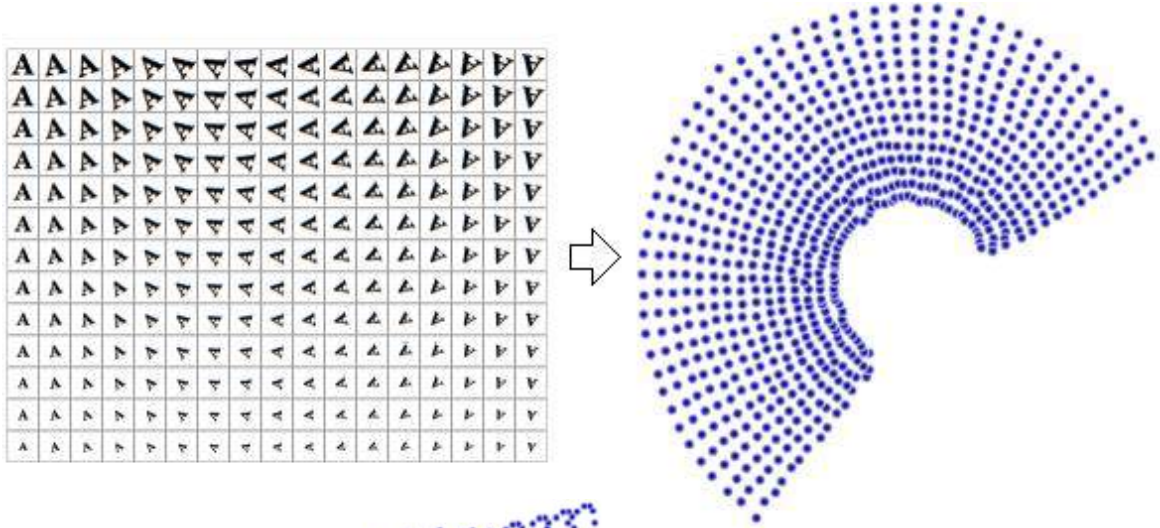
- When do we care about this?

- MDS can recover the map from a kind of affinity matrix
  - We can control the dimensionality of the map using eigenvalues
    - e.g. The buildings are actually lying in a 3D space, not 2D
  - MDS is a linear technique and its affinity matrix is inner product
    - Perfect reconstruction is possible if we know the rank
- From this, what we can say is
  - We can recover the lower dim representation from the affinity matrix
    - Not from the data points
  - We can use whatever funky affinity matrix
    - If we don't care about the perfect reconstruction
- So, when do we not care about the perfect reconstruction?
  - If the data points are too noisy and high dimensional
    - Noisy AND high dimensional?

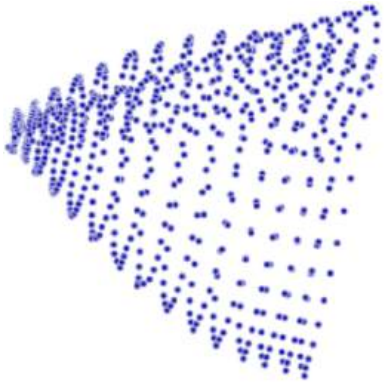
Intrinsic Dimensionality

- Moving A's

- How many dimensions do we need?



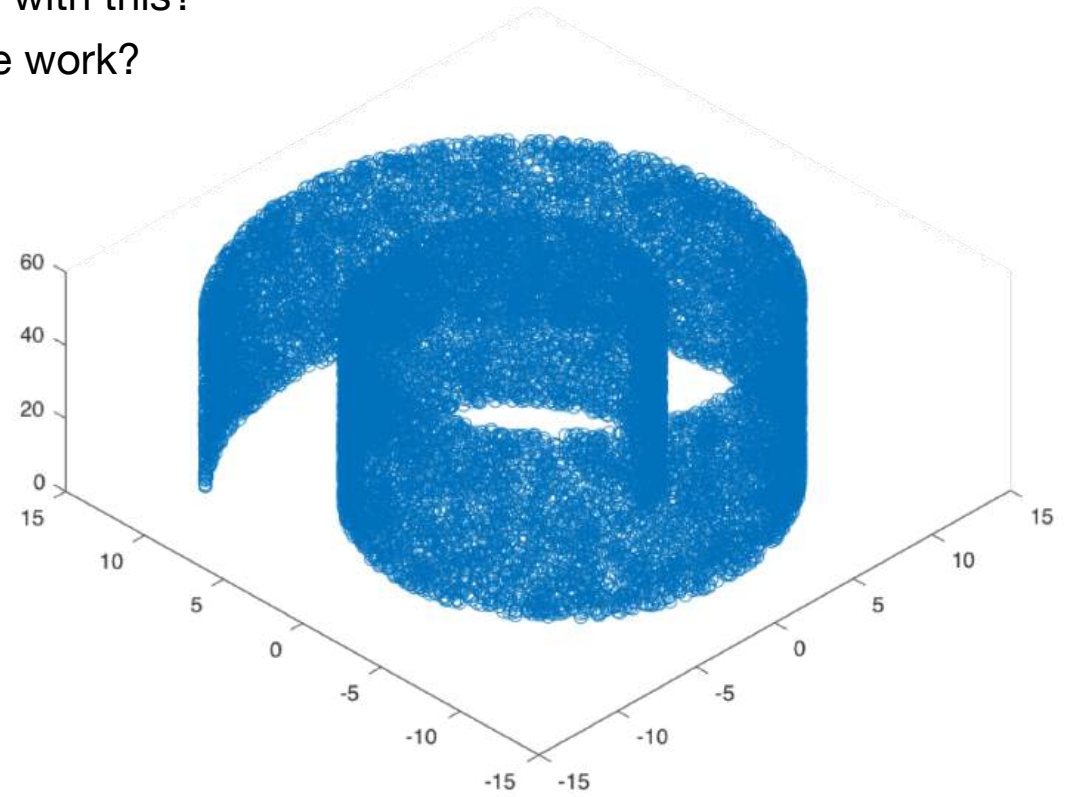
- PCA doesn't do it well



Manifold Learning

- Nonlinear dimension reduction: the Swiss roll example

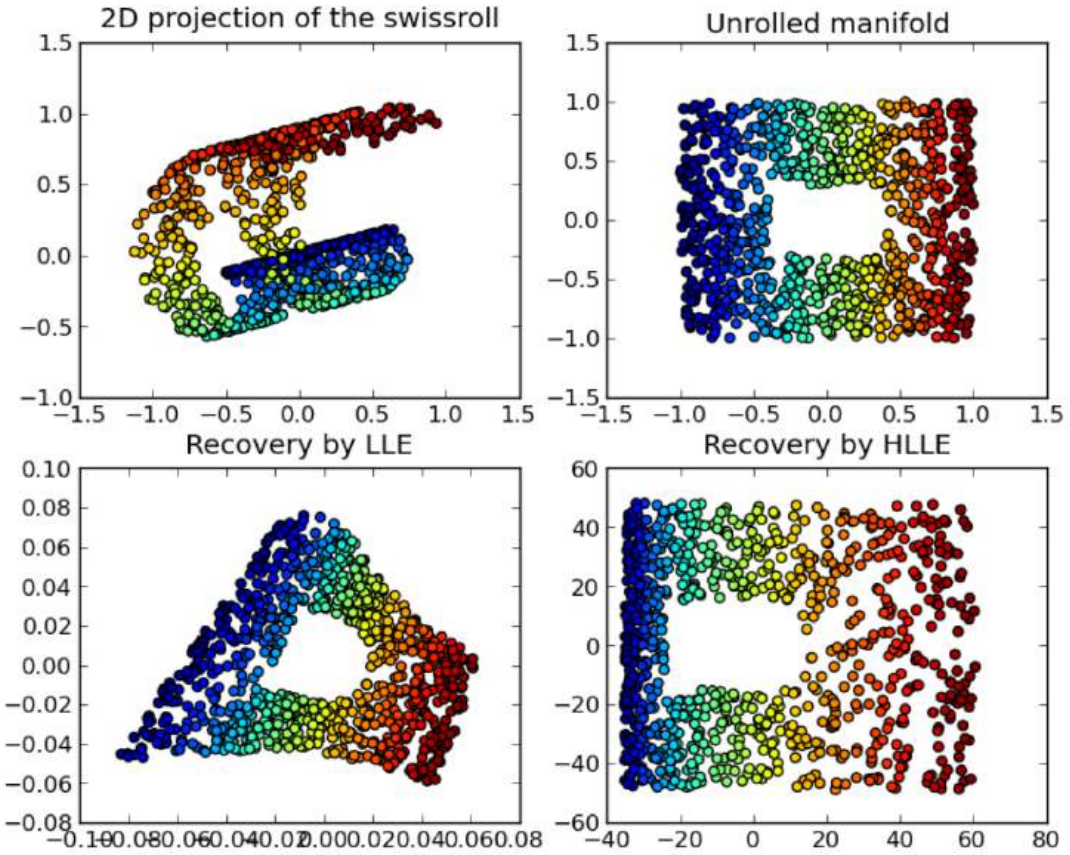
- The Swiss roll example
  - What do you want to do with this?
  - Would a linear technique work?



Manifold Learning

- Nonlinear dimension reduction: the Swiss roll example

- Unrolling needs a nonlinear projection



https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction

Laplacian Eigenmap

- Eigendecomposition on the RBF kernel

- First, we build an affinity matrix using the RBF kernel

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{\sigma^2}\right)$$

  - With a small σ for the locality
  - In practice, we skip the non-neighboring pairs $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$
    - If $||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 > \tau$
    - Or, if $\boldsymbol{x}_j \notin \mathcal{N}_i$

- The objective function $\underset{\boldsymbol{Y}}{\arg\min} \sum_{i,j} \underbrace{(\boldsymbol{y}_i - \boldsymbol{y}_j)^\top (\boldsymbol{y}_i - \boldsymbol{y}_j)}_{\text{Distance after projection}} \underbrace{\boldsymbol{W}_{ij}}_{\text{Original affinity } \mathcal{K}}$

  - It will be minimal if originally similar pairs are also similar after the projection

- Why not the direct decomposition on $\boldsymbol{W}$ (as in MDS)?
  - Because perfect reconstruction might not be possible

Laplacian Eigenmap

- Eigendecomposition on the RBF kernel

- The objective function $\arg\min_{\boldsymbol{Y}} \sum_{i,j} (\boldsymbol{y}_i - \boldsymbol{y}_j)^\top (\boldsymbol{y}_i - \boldsymbol{y}_j) \boldsymbol{W}_{ij}$
- Optimization

$$\begin{aligned}\mathcal{E} &= \sum_{i,j} (\boldsymbol{y}_i^\top \boldsymbol{y}_i - 2\boldsymbol{y}_j^\top \boldsymbol{y}_i + \boldsymbol{y}_j^\top \boldsymbol{y}_j) \boldsymbol{W}_{ij} \\ &= \sum_i \boldsymbol{y}_i^\top \boldsymbol{y}_i \sum_j \boldsymbol{W}_{ij} + \sum_j \boldsymbol{y}_j^\top \boldsymbol{y}_j \sum_i \boldsymbol{W}_{ij} - 2 \sum_{i,j} \boldsymbol{y}_j^\top \boldsymbol{y}_i \boldsymbol{W}_{ij} \\ &= 2(\boldsymbol{Y}\boldsymbol{D}\boldsymbol{Y}^\top - \boldsymbol{Y}\boldsymbol{W}\boldsymbol{Y}^\top) = 2\boldsymbol{Y}\boldsymbol{L}\boldsymbol{Y}^\top\end{aligned}$$

$\boldsymbol{D}_{ii} = \sum_j \boldsymbol{W}_{ij}$ — A diagonal matrix; A vector

$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$

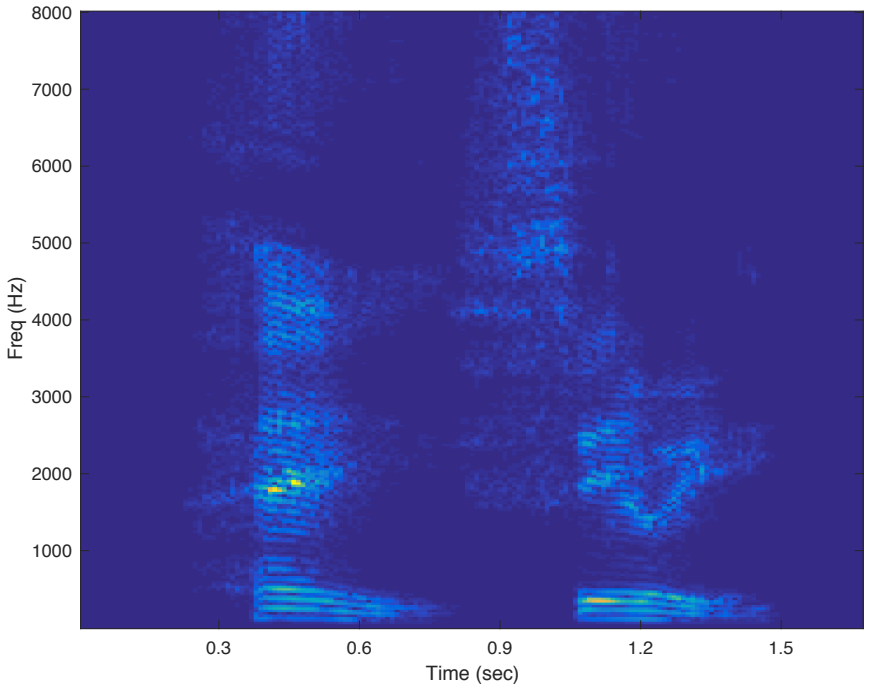
- To avoid trivial solution $\boldsymbol{Y} = \boldsymbol{0}$ we specify the norm of the solution

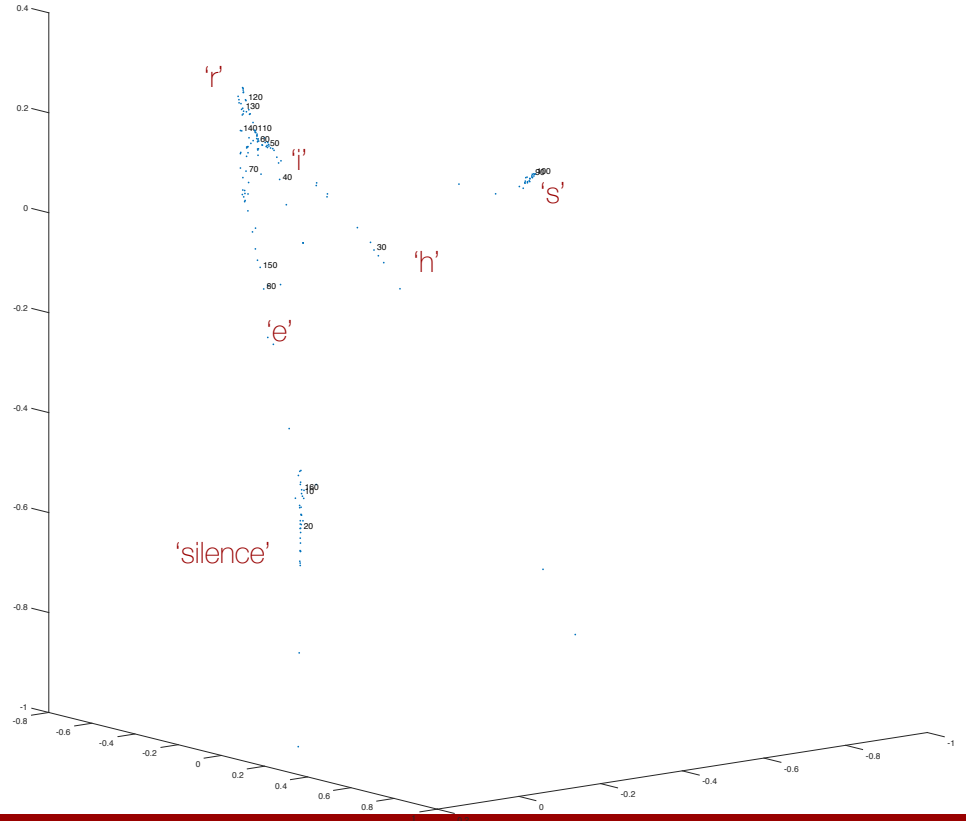
$$\arg\min_{\boldsymbol{Y}} \boldsymbol{Y}\boldsymbol{L}\boldsymbol{Y}^\top \quad \text{s.t. } \boldsymbol{Y}\boldsymbol{D}\boldsymbol{Y}^\top = \boldsymbol{I}$$

- Or, $\arg\min_{\tilde{\boldsymbol{Y}}} \tilde{\boldsymbol{Y}}\tilde{\boldsymbol{L}}\tilde{\boldsymbol{Y}}^\top \quad \text{s.t. } \tilde{\boldsymbol{Y}}\tilde{\boldsymbol{Y}}^\top = \boldsymbol{I} \qquad \tilde{\boldsymbol{L}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{D}^{-\frac{1}{2}} \qquad \tilde{\boldsymbol{Y}} = \boldsymbol{Y}\boldsymbol{D}^{\frac{1}{2}}$
  - Ring a bell?
    - **Eigendecomposition on the Laplacian matrix** $\tilde{\boldsymbol{L}}$

Laplacian Eigenmaps

- On “Hey, Siri”

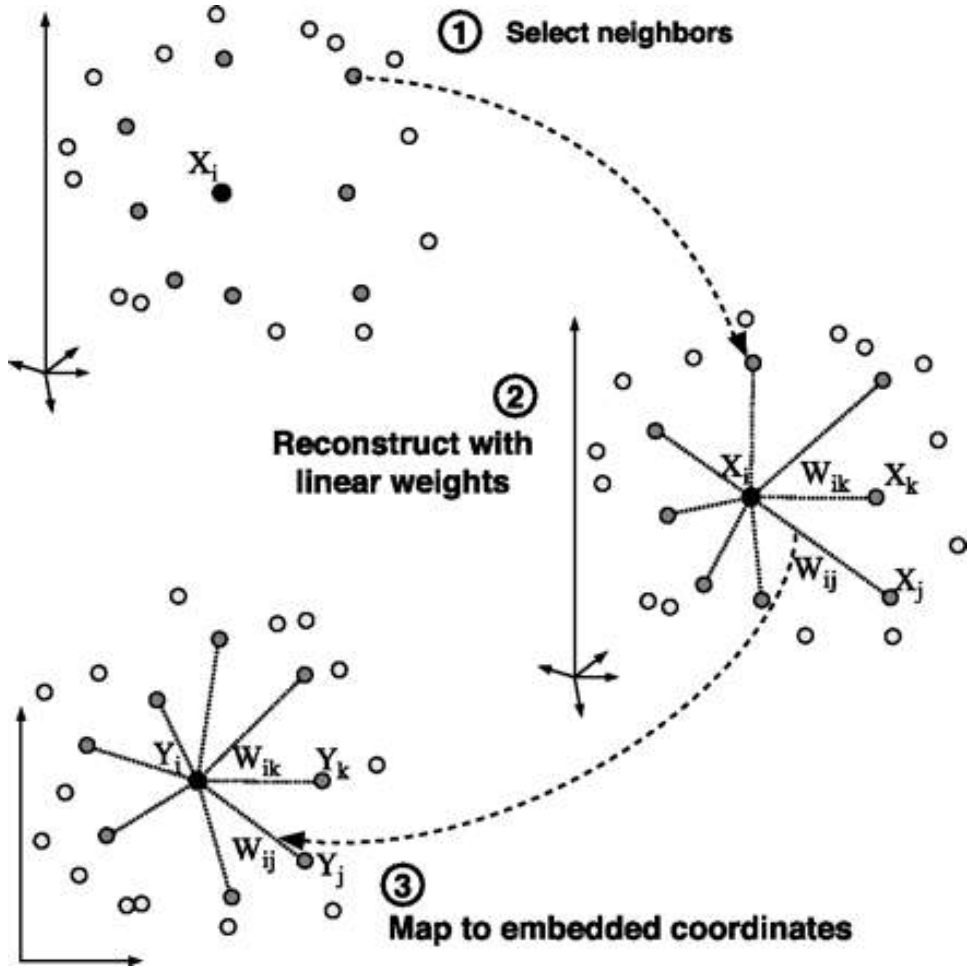




Locally Linear Embedding (LLE)

- The other manifold learning techniques

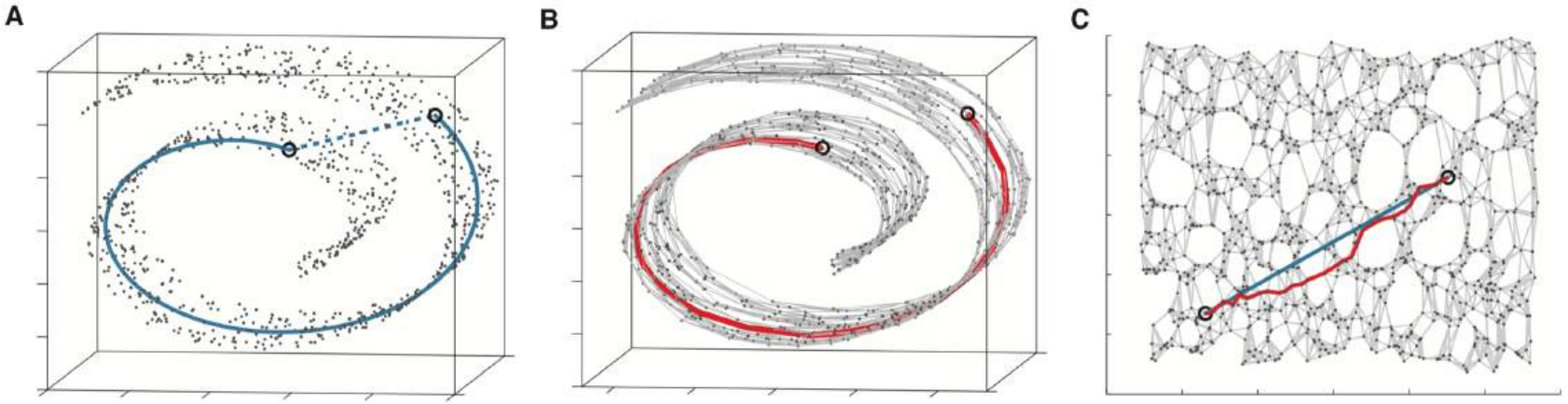
- For each data points, find a set of nearest neighbors
- Construct a local convex combination
- Find the lower dimensional version that preserves these local structures
- Yet another eigendecomposition problem



https://www.cs.nyu.edu/~roweis/lle/algorithm.html

ISOMAP

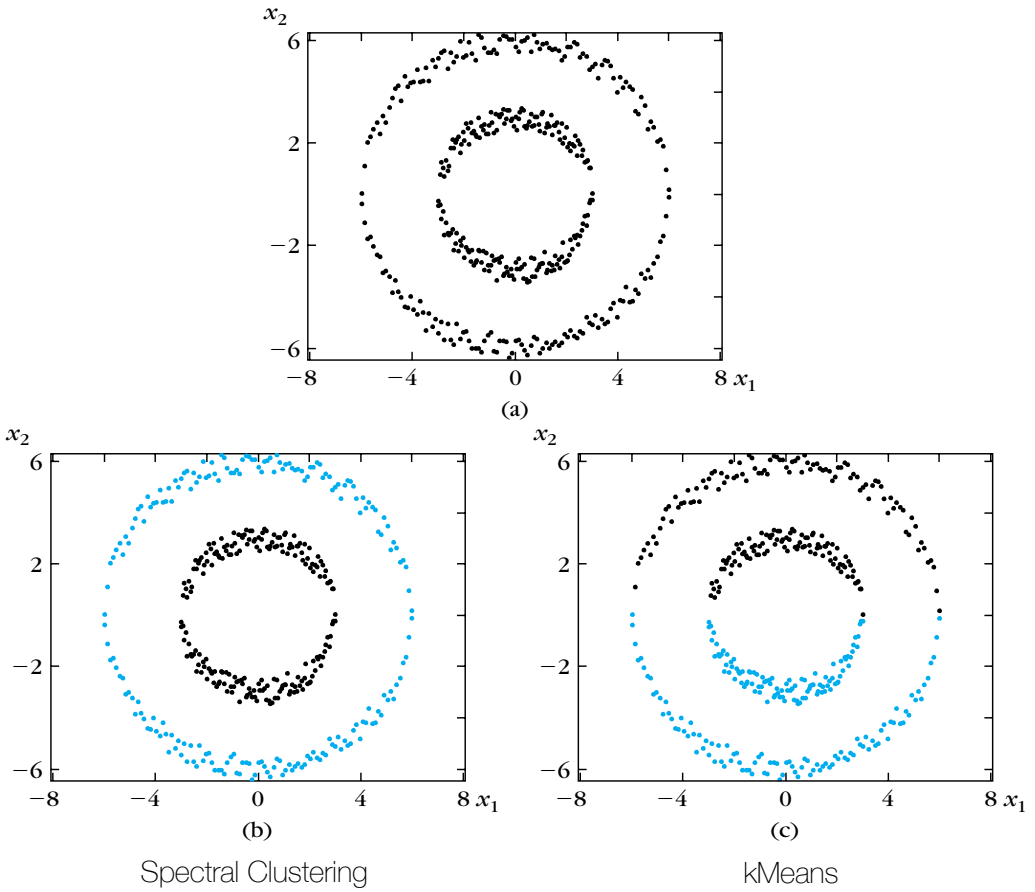
- The other manifold learning techniques



- For each data points, find a set of nearest neighbors
- Compute pairwise shortest paths (figure B)
  - Using Djikstra's algorithm
  - Which approximates the geodesic distance (figure A)
- You built a nonlinear distance matrix. Do MDS on this.

Spectral Clustering

- Nonlinear clustering



Eigendecomposition on the Kernel Matrix

- What is it good for?

- The techniques we learned:
  - We can find a lower dimensional feature space from the affinity matrix, not from the data
    - Eigenvectors correspond to the coefficients after the projection, not the projection themselves
  - The projection can be nonlinear, too
    - And we don't have an access to the projection

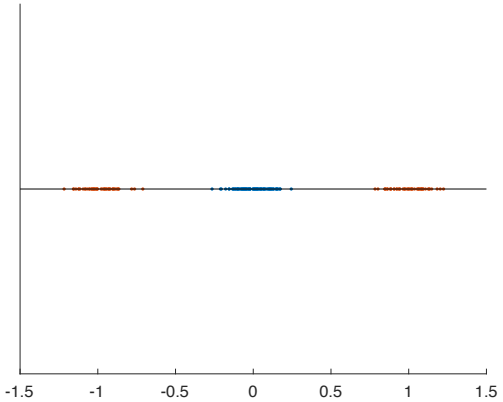
- What's so special about those kernel matrices?
  - We only saw RBF kernel, but there are other options

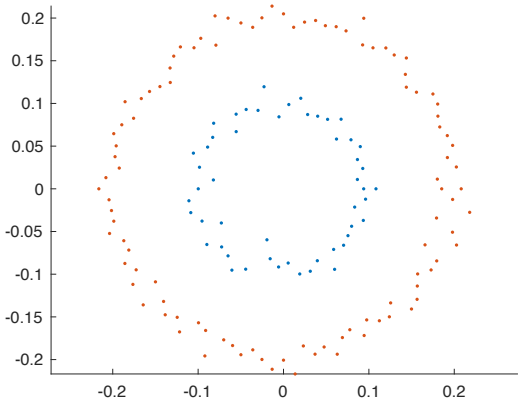
- In general…
  - If we use the kernel tricks we can avoid learning direct nonlinear transforms
  - Kernel tricks still let us discover nonlinear structure from the data
  - We call this kind of algorithms **kernel methods**

Nonlinear Transformation

- For linear separability

- A problem not linearly separable

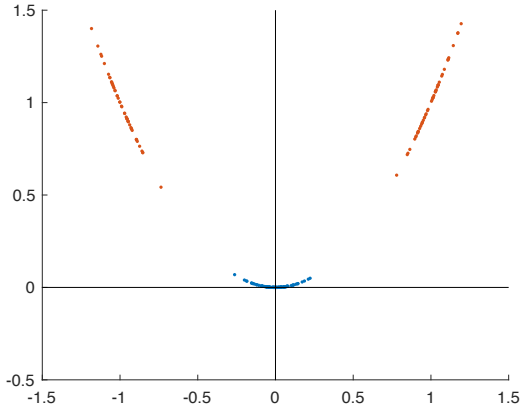


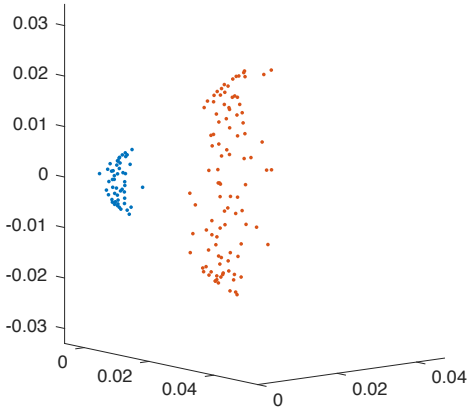


$$\phi(x_1) = [x_1, x_1^2]^\top$$

$$\phi(x_1, x_2) = [x_1^2, x_2^2, x_1 x_2]^\top$$

- After the transform

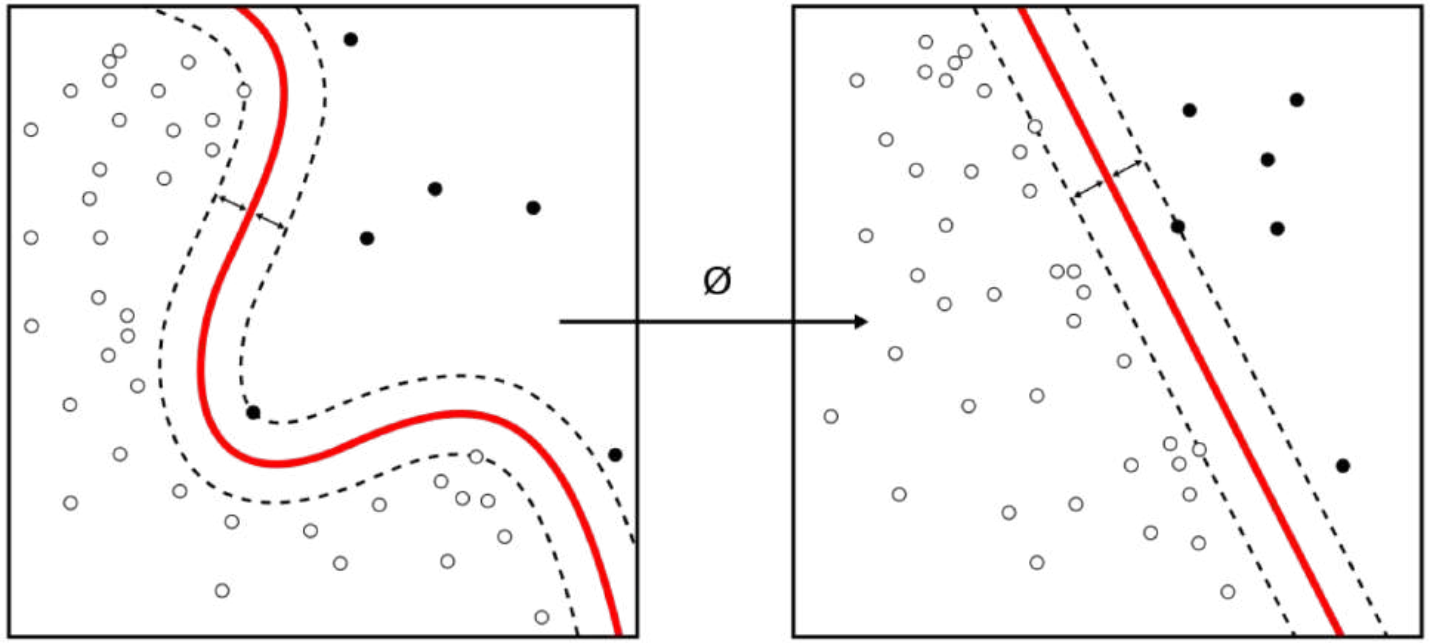




Nonlinear Transformation

- For linear separability

- Nonlinearly separable functions might be able to be solved linearly
  - In the nonlinearly found feature space



- If you increase the dimensionality during the nonlinear transform, you hope that you can solve it linearly

Nonlinear Transformation

- For linear separability

- It's difficult to find out this magical nonlinear transform function
  - Because it's problem-specific
- What we know is that the problem becomes easier to solve if we blow up the dimensionality
- Then, apply an arbitrary transform function to increase the dimensionality?
  - I don't want to do that
  - It may make the problem complicated with the high dimensionality
- Is there any way to work with the high dimensional data without explicitly knowing their values?
  - Kernels!
  - The kernel matrix can ***imply*** the relationship in the ***transformed feature space***

Radian Basis Function (RBF)

- An infinite dimensionality

- We've seen the use of RBF kernels in the previous examples
  - What's so special about it? (you know, we still don't know the mapping function)

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{\sigma^2}\right)$$

- The dimensionality of the feature space of the RBF kernel is **infinite**
  - So, using the RBF kernel implies that the transform increased the dimensionality a lot

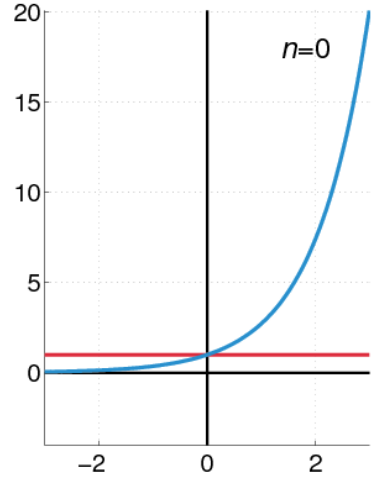
- In other words $\boldsymbol{z}_i = \phi(\boldsymbol{x}_i) \qquad \boldsymbol{z}_j = \phi(\boldsymbol{x}_j)$ ← We don't explicitly know the mapping

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{z}_i^\top \boldsymbol{z}_j$$ ← Kernel is the inner products of the features

$$\boldsymbol{z}_i, \boldsymbol{z}_j \in \mathbb{R}^\infty$$ ← For RBF the feature space is with an infinite dimensionality

Radian Basis Function (RBF)

- An infinite dimensionality



- Let's prove it for a simple case
  - x_i is scalar
  - $\sigma = 1$
- Then the kernel matrix can be defined as follows

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-(x_i - x_j)^2\right) = \exp\left(-x_i^2 - x_j^2 + 2x_i x_j\right)$$

$$= \exp\left(-x_i^2\right) \exp\left(-x_j^2\right) \exp\left(2x_i x_j\right)$$

Taylor series $$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

$$= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Therefore $$\exp(x) \approx \sum_{n=0}^{\infty} \frac{\exp(0)}{n!}(x-0)^n = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-x_i^2\right) \exp\left(-x_j^2\right) \sum_{n=0}^{\infty} \frac{2^n x_i^n x_j^n}{n!}$$

Radian Basis Function (RBF)

- An infinite dimensionality

- An inner product between infinite-dimensional vectors

$$\sum_{n=0}^{\infty} \frac{2^n x_i^n x_j^n}{n!} = \left[\frac{\sqrt{(2^0)}x_i^0}{\sqrt{0!}}, \frac{\sqrt{(2^1)}x_i^1}{\sqrt{1!}}, \frac{\sqrt{(2^2)}x_i^2}{\sqrt{2!}}, \cdots, \frac{\sqrt{(2^\infty)}x_i^\infty}{\sqrt{\infty!}} \right] \begin{bmatrix} \frac{\sqrt{(2^0)}x_j^0}{\sqrt{0!}} \\ \frac{\sqrt{(2^1)}x_j^1}{\sqrt{1!}} \\ \frac{\sqrt{(2^2)}x_j^2}{\sqrt{2!}} \\ \cdots \\ \frac{\sqrt{(2^\infty)}x_j^\infty}{\sqrt{\infty!}} \end{bmatrix}$$

- Eventually, the RBF kernel is an inner product between infinite dimensional features

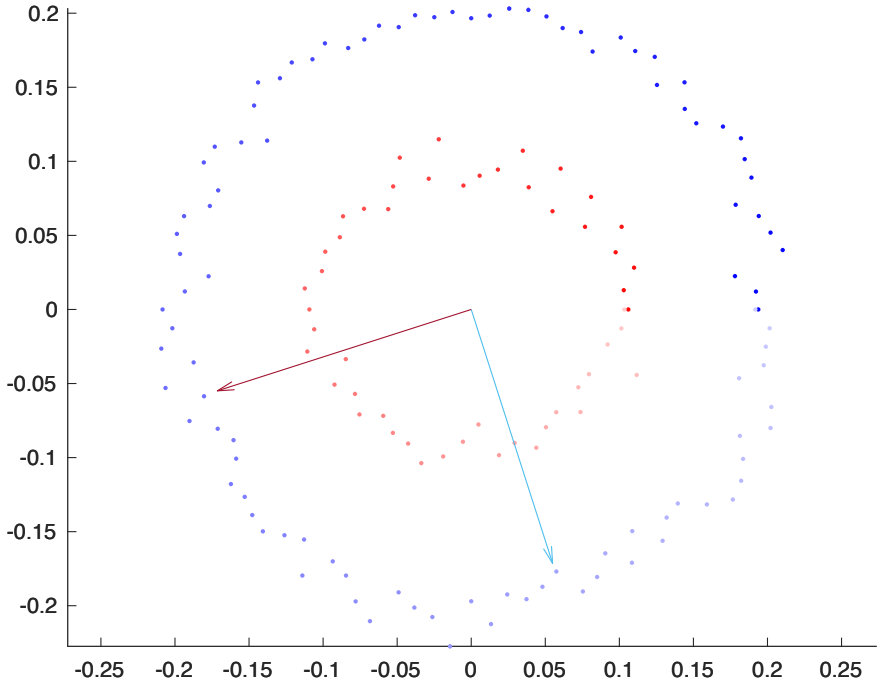
$$\mathcal{K}(x_i, x_j) = \exp\left(-x_i^2\right) \exp\left(-x_j^2\right) \sum_{n=0}^{\infty} \frac{2^n x_i^n x_j^n}{n!}$$

$$\mathcal{K}(x_i, x_j) = \left[\exp(-x_i^2) \frac{\sqrt{(2^0)}x_i^0}{\sqrt{0!}}, \exp(-x_i^2) \frac{\sqrt{(2^1)}x_i^1}{\sqrt{1!}}, \exp(-x_i^2) \frac{\sqrt{(2^2)}x_i^2}{\sqrt{2!}}, \cdots, \exp(-x_i^2) \frac{\sqrt{(2^\infty)}x_i^\infty}{\sqrt{\infty!}} \right] \begin{bmatrix} \exp(-x_j^2) \frac{\sqrt{(2^0)}x_j^0}{\sqrt{0!}} \\ \exp(-x_j^2) \frac{\sqrt{(2^1)}x_j^1}{\sqrt{1!}} \\ \exp(-x_j^2) \frac{\sqrt{(2^2)}x_j^2}{\sqrt{2!}} \\ \cdots \\ \exp(-x_j^2) \frac{\sqrt{(2^\infty)}x_j^\infty}{\sqrt{\infty!}} \end{bmatrix}$$

Kernel PCA

- Concentric circles

- What would be the principal components of these data samples?

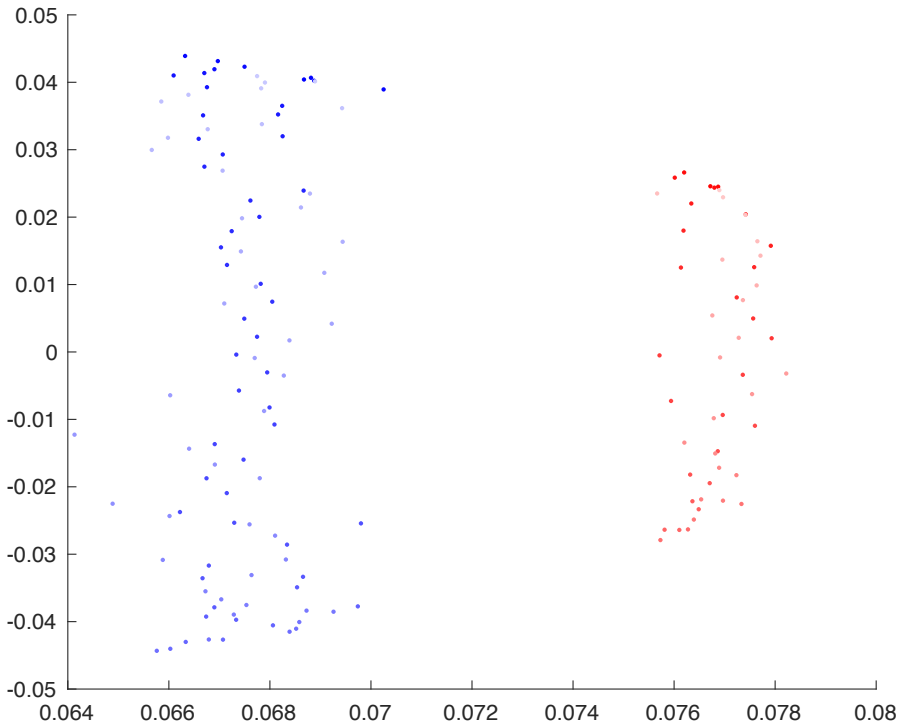


- An ordinary PCA doesn't give a satisfactory result

Kernel PCA

- Concentric circles

- On the other hand, kernel PCA can find out the (nonlinear) principal components



Kernel PCA

- Eigendecomposition with the kernel trick

- Suppose that we observe a dataset with a nonlinear structure
- Suppose that there is a nonlinear transform that corresponds to it

$$\boldsymbol{z}_i = \phi(\boldsymbol{x}_i)$$

  - Although we don't know what it exactly would be
- We assume that $\boldsymbol{z}_i$ is higher dimensional
  - And we assume the nonlinear structure turned into a linear structure
- Now we want to do PCA on $\boldsymbol{z}_i$, not on $\boldsymbol{x}_i$
  - Because $\boldsymbol{z}_i$ is with a linear structure now, although high dimensional
- PCA on the nonlinear transformed data starts from a cov matrix

$$\boldsymbol{C} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{z}_i \boldsymbol{z}_i^\top = \frac{1}{N}\sum_{i=1}^{N} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^\top$$

- But we don't exactly know the mapping function $\phi(\boldsymbol{x}_i)$
  - What can we do?

Kernel PCA

- Eigendecomposition with the kernel trick

- The definition of eigendecomposition gives us a hint

$$\boldsymbol{C}\boldsymbol{v} = \lambda\boldsymbol{v}$$

What's their dimensionality?

$$\left\{\frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^\top\right\}\boldsymbol{v} = \lambda\boldsymbol{v}$$

$$\frac{1}{N}\boldsymbol{Z}\boldsymbol{Z}^\top\boldsymbol{v} = \lambda\boldsymbol{v}$$

This is high dimensional, but a vector

$$\boldsymbol{Z}\boldsymbol{\alpha} = \boldsymbol{v}$$

Plug in

- Eventually,

$$\frac{1}{N}\boldsymbol{Z}\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\alpha} = \lambda\boldsymbol{Z}\boldsymbol{\alpha}$$

$$\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\alpha} = N\lambda\boldsymbol{\alpha}$$

The kernel matrix, not a covariance matrix

- We have a new eigendecomposition problem, on the Kernel matrix of the nonlinear transformed data

Kernel PCA

- Eigendecomposition with the kernel trick

- How do we recover the coefficients after the projection?
  - We want to do $\boldsymbol{y} = \boldsymbol{v}^\top \boldsymbol{Z}$, but we know neither $\boldsymbol{Z}$ nor $\boldsymbol{v}$
- With the kernel trick, we can do this, because $\boldsymbol{y} = \boldsymbol{v}^\top \boldsymbol{Z} = \boldsymbol{\alpha}^\top \boldsymbol{Z}^\top \boldsymbol{Z} = \boldsymbol{\alpha}^\top \mathcal{K}$
- But we never know the actual projection vector $\boldsymbol{Z}\boldsymbol{\alpha} = \boldsymbol{v}$
- Another thing to note
  - The eigendecomposition on the kernel matrix $\mathcal{K} = \boldsymbol{Z}^\top \boldsymbol{Z}$ will yield orthonormal eigenvectors $\boldsymbol{\alpha}$
  - But it doesn't guarantee the normalization of $\boldsymbol{v}$
  - So, we need to normalize $\boldsymbol{\alpha}$ further to make sure this: $\boldsymbol{v}^\top \boldsymbol{v} = \boldsymbol{\alpha}^\top \boldsymbol{Z}^\top \boldsymbol{Z}\boldsymbol{\alpha} = 1$
- One more thing to note: since the transform is nonlinear, the high dimensional features are not guaranteed to be centered

Kernel PCA

- The recipe

- Choose your kernel function
  - e.g. RBF
- Calculate the kernel matrix $\mathcal{K}$
  - Using the kernel function, not from the nonlinear transform
  - Center the kernel matrix
- Do eigendecomposition on $\mathcal{K}$
- Choose a few eigenvectors $\boldsymbol{\alpha}$
  - This will determine the dimensionality of this nonlinear dimensionality reduction algorithm
  - Normalize them further
- Compute the coefficients after the nonlinear dim reduction $\boldsymbol{y} = \boldsymbol{\alpha}^\top \mathcal{K}$

- **Note that this is exactly like MDS except we construct the kernel from the RBF function, not from an inner product**

Summary of the Kernel Methods

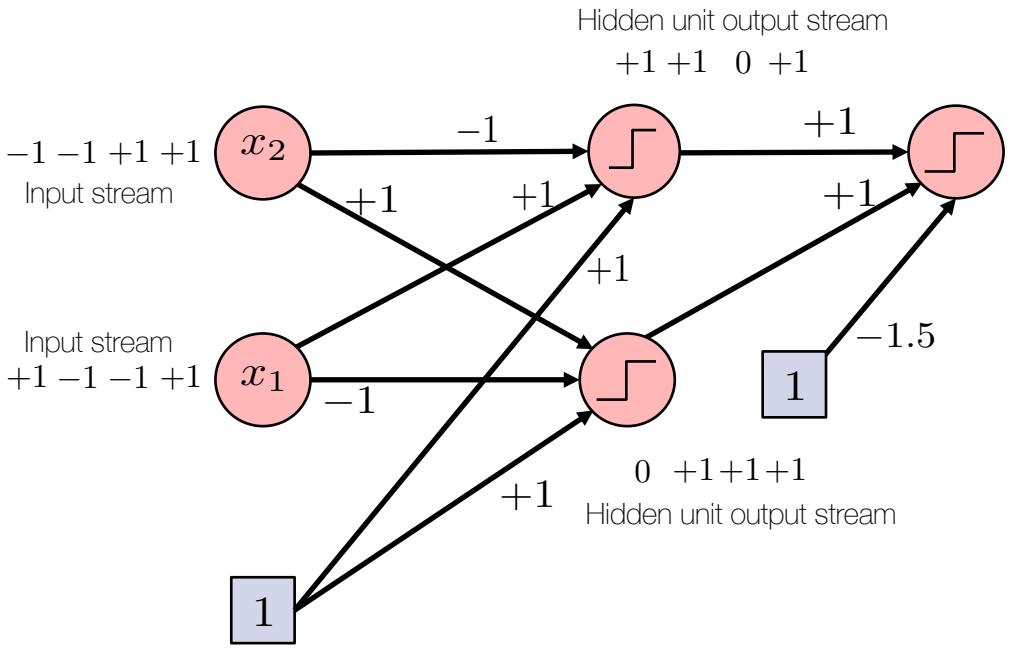
- Pros and cons

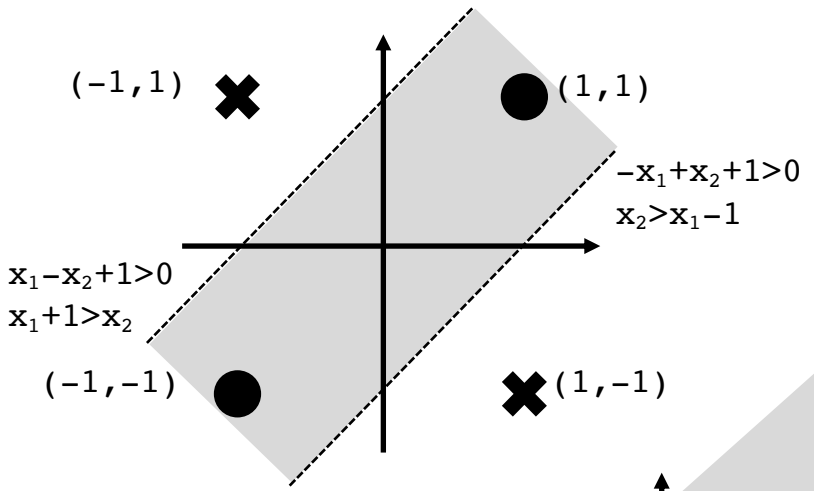
- Pros
  - We can do a nonlinear analysis of a data set
  - We don't have to know the nonlinear feature transform

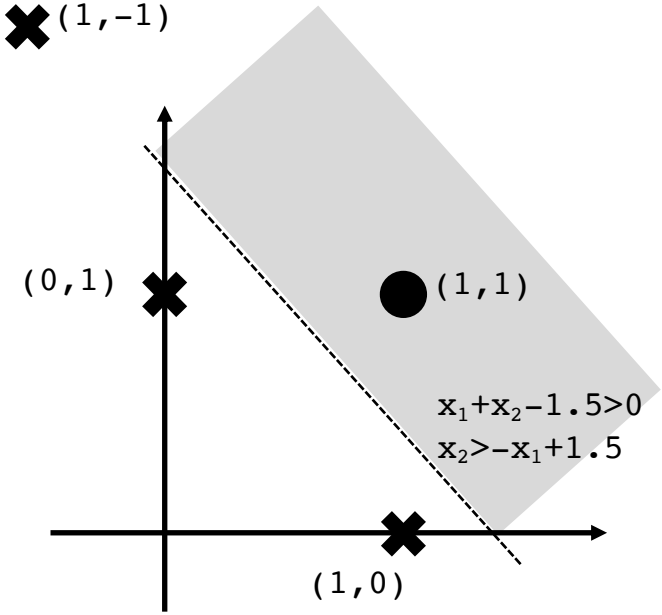
- Cons
  - We don't exactly know the nonlinear feature transform function
  - If there are too many data samples the size of the kernel matrix is too large to handle
  - Since we don't have the access to the mapping function, handling new data points is not straightforward
    - Though we can still do an out-of-samples extension
    - Basically the kernel matrix needs to grow

Neural Networks (Revisited)

- Non-linear transform using NN

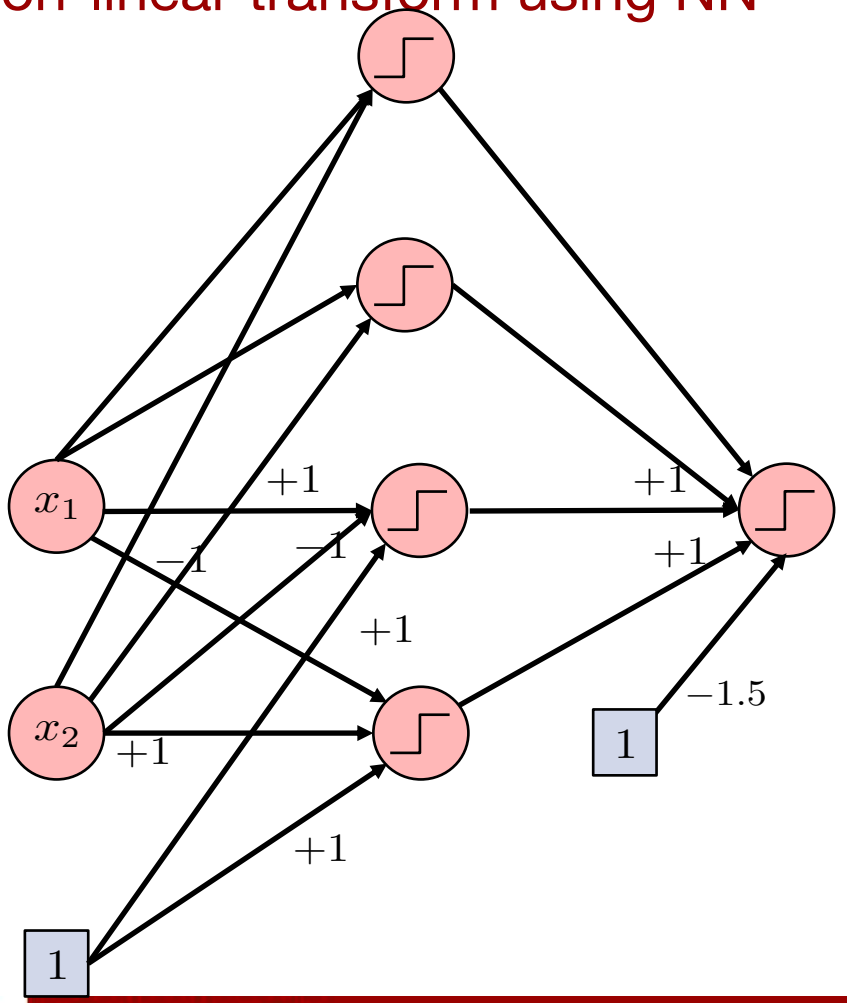


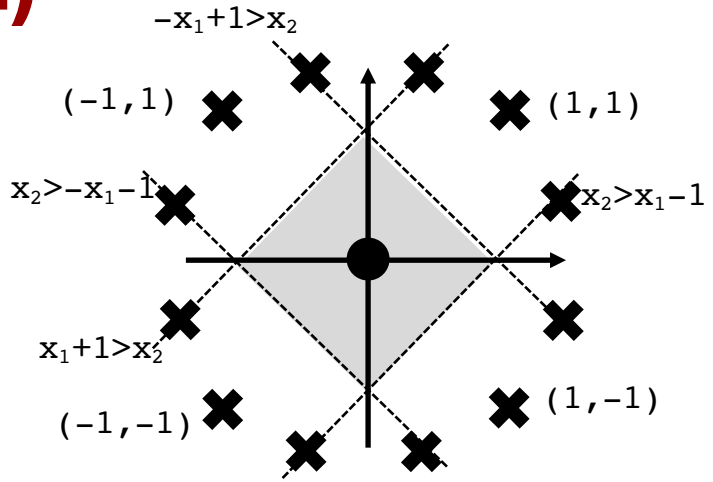


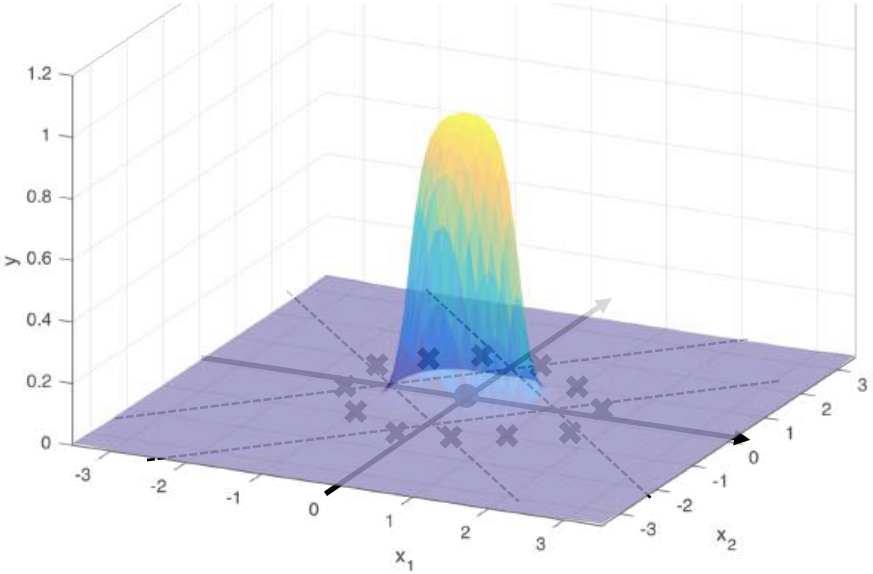


Neural Networks (Revisited)

- Non-linear transform using NN







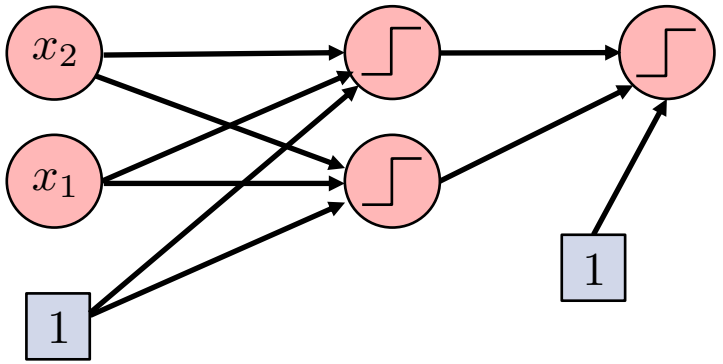
Multilayer Perceptron

- Backpropagation

- First we randomly initialize the weights and forwardpropagate
  - Layer 1

$$\begin{bmatrix} z_2^{(1)} \\ z_1^{(1)} \end{bmatrix} = \begin{bmatrix} A_{22}^{(1)} & A_{21}^{(1)} & A_{20}^{(1)} \\ A_{12}^{(1)} & A_{11}^{(1)} & A_{10}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_2^{(1)} \\ x_1^{(1)} \\ 1 \end{bmatrix} \qquad \boldsymbol{z}^{(1)} = \boldsymbol{A}^{(1)}\boldsymbol{x}^{(1)}$$

$$\begin{bmatrix} x_2^{(2)} \\ x_1^{(2)} \end{bmatrix} = g\left(\begin{bmatrix} z_2^{(1)} \\ z_1^{(1)} \end{bmatrix}\right) \qquad \boldsymbol{x}^{(2)} = g(\boldsymbol{z}^{(1)})$$



  - Final layer

$$z_1^{(2)} = \begin{bmatrix} A_{12}^{(2)} & A_{11}^{(2)} & A_{10}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_2^{(2)} \\ x_1^{(2)} \\ 1 \end{bmatrix} \qquad z^{(2)} = \boldsymbol{A}^{(2)}\boldsymbol{x}^{(2)}$$

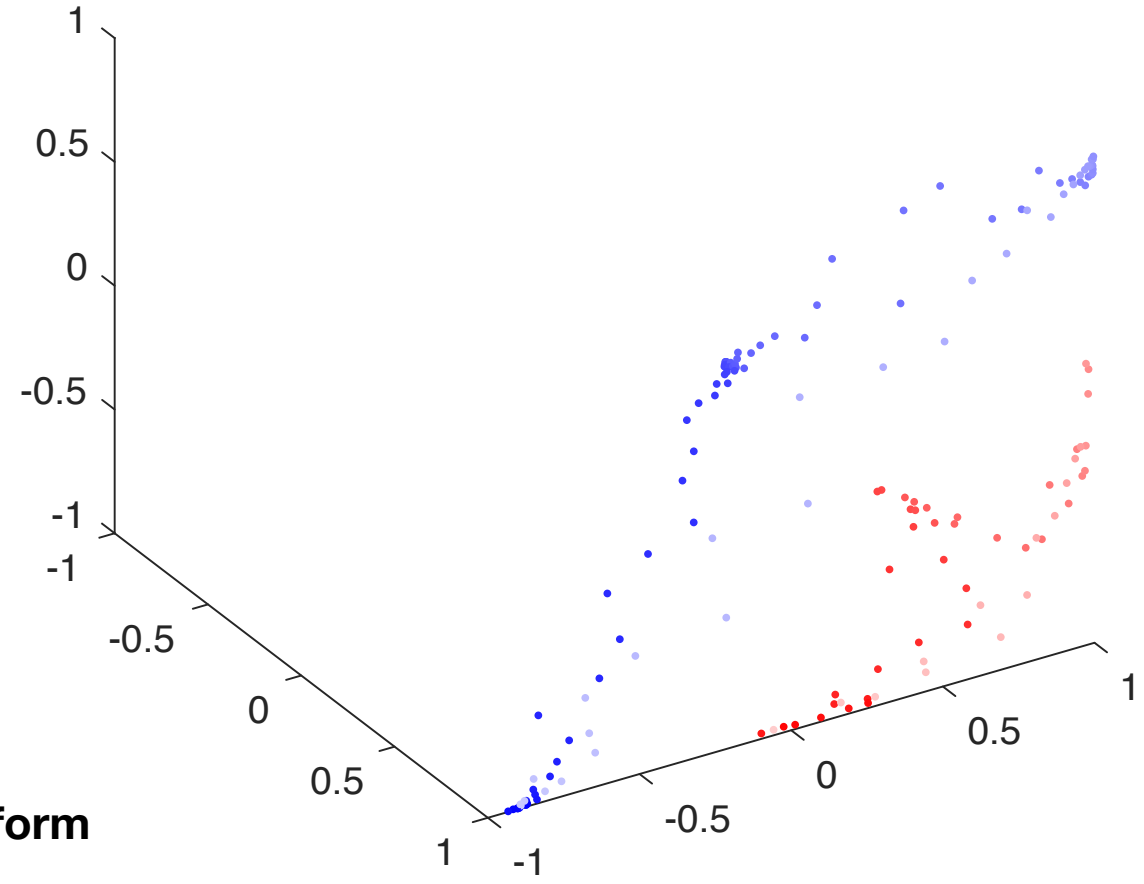
$$\hat{y} = g(z^{(2)})$$

- We can think of
  - The first layer as a nonlinear feature transform procedure
  - The final layer as the actual mapping function approximation layer (e.g. classification)
  - So what is the variable that represents the features?

Neural Networks (Revisited)

- Non-linear transform using NN

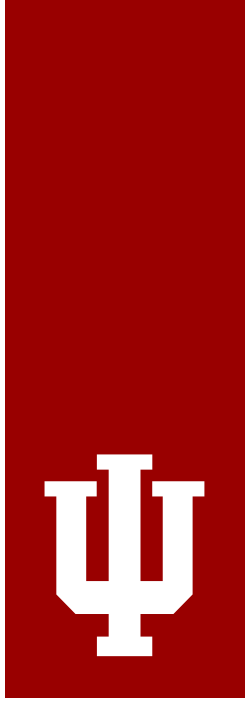
- The network takes the concentric circles as the input
- Plot of $x^{(2)}$
  - The output of the first layer
  - There are three hidden units



- They are now linearly separable
- **NN can do some nonlinear feature transform**

Reading

- Chapter 6.7, 15.2.4, 6.3



Thank You!