Lecture 8

DJM

13 November 2018

Motivation

Overview

Representation learning is the idea that performance of ML methods is highly dependent on the choice of representation

For this reason, much of ML is geared towards transforming the data into the relevant features and then using these as inputs

This idea is as old as statistics itself, really,

(e.g. Pearson (1901), where PCA was first introduced)

However, the idea is constantly revisited in a variety of fields and contexts

Commonly, these learned representations capture 'low level' information like overall shape types

Other sharp features, such as images, aren't captured

It is possible to quantify this intuition for PCA at least

PCA

Principal components analysis (PCA) is an (unsupervised) dimension reduction technique

It solves various equivalent optimization problems

(Maximize variance, minimize L_2 distortions, find closest subspace of a given rank,...)

At its core, we are finding linear combinations of the original (centered) covariates

$$Z_{ij} = \alpha_j^\top X_i$$

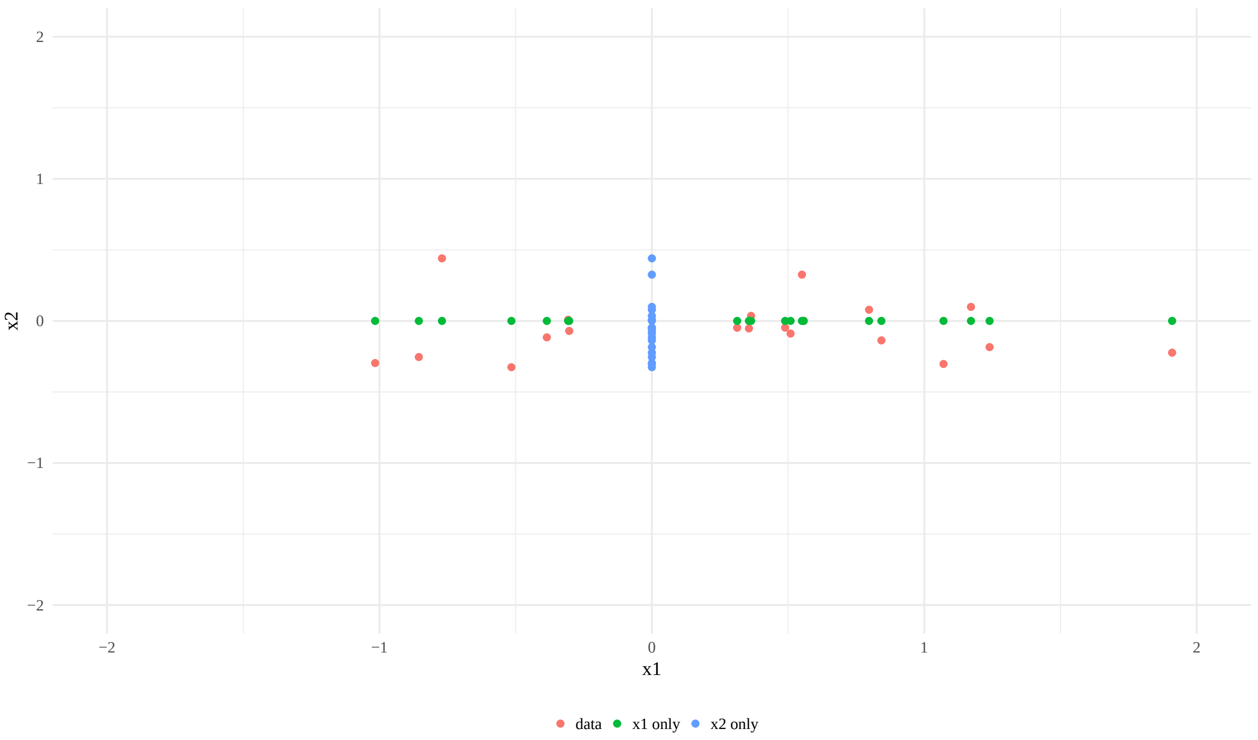
This is expressed via the SVD: $X - \overline{X} = UDV^\top$ as

$$Z = XV = UD$$

Lower dimensional embeddings

Suppose we have predictors x_1 and x_2

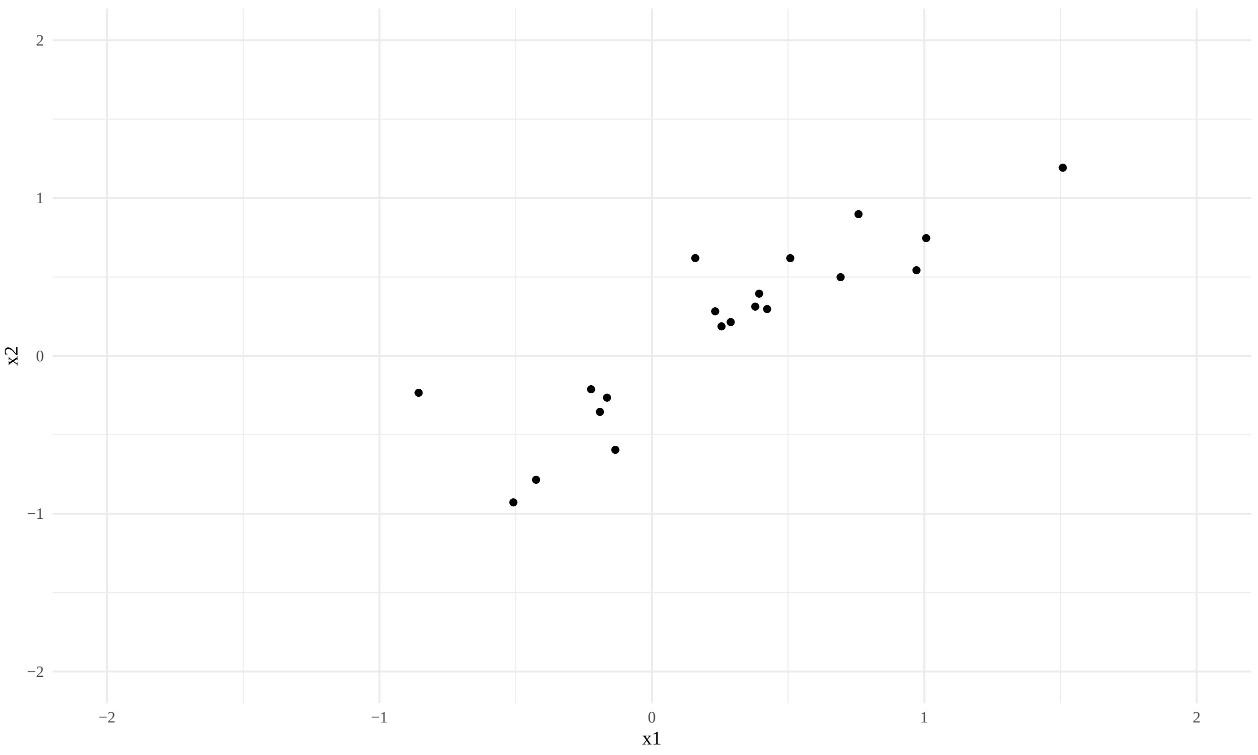
- We more faithfully preserve the structure of the data by keeping x_1 and setting x_2 to zero than the opposite



Lower dimensional embeddings

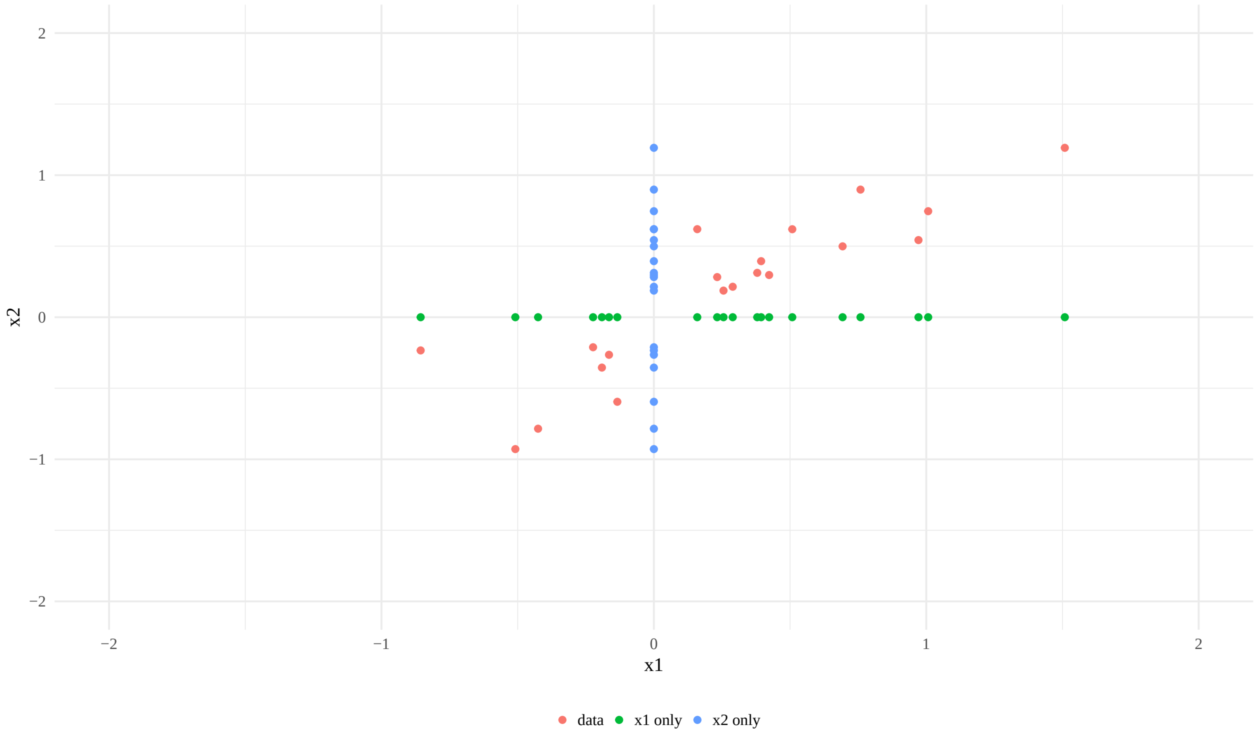
An important feature of the previous example is that x_1 and x_2 aren't correlated

What if they are?



Lower dimensional embeddings

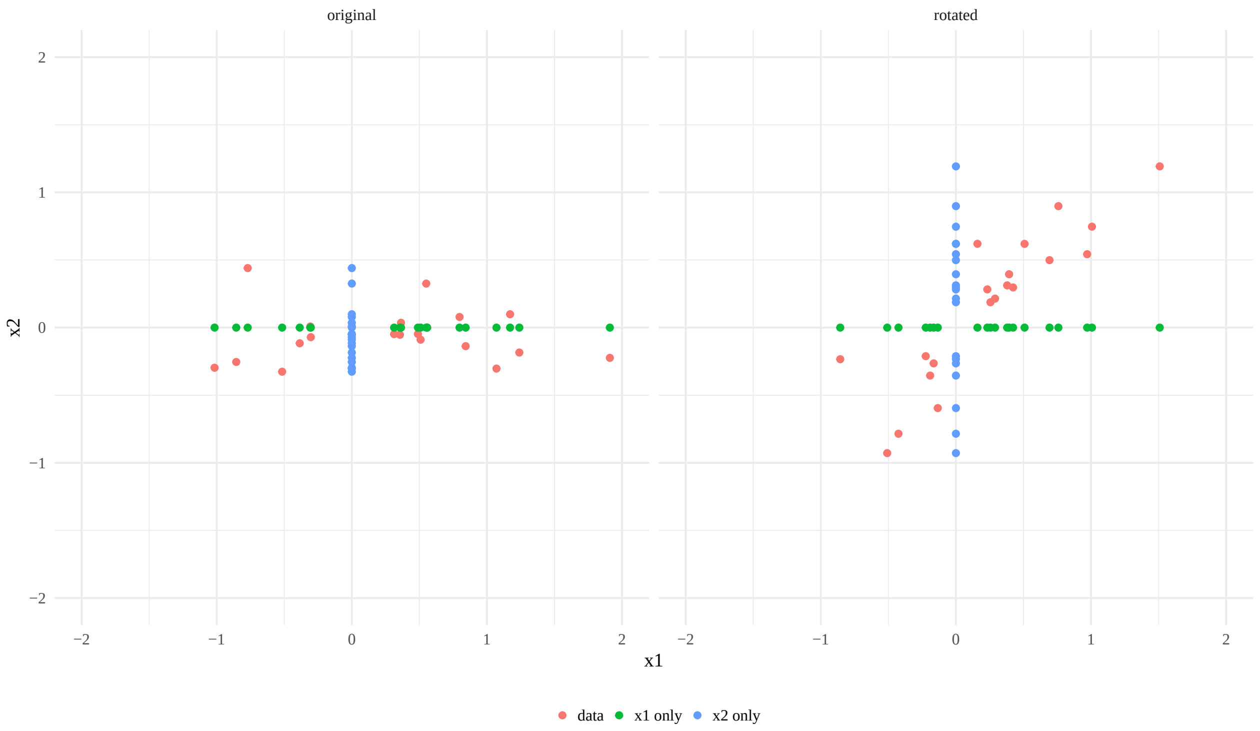
We lose a lot of structure by setting either x_1 or x_2 to zero



Lower dimensional embeddings

There isn’t that much structurally different between the examples

One is just a rotation of the other



PCA

If we knew how to rotate our data, then we could more easily retain the structure.

PCA gives us exactly this rotation

Optimization

If we want to find the first K principal components, the relevant optimization program is:

$$\min_{\mu,(\lambda_i),V_K} \sum_{i=1}^{n} \|X_i - \mu - V_K\lambda_i\|^2$$

This representation is important

It shows that we are trying to reconstruct lower dimensional representations of the covariates

PCA

$$\min_{\mu,(\lambda_i),V_K} \sum_{i=1}^{n} \|X_i - \mu - V_K\lambda_i\|^2$$

We can partially optimize for μ and (λ_i) to find

- $\widehat{\mu} = \overline{X}$
- $\widehat{\lambda}_i = V_K^\top (X_i - \widehat{\mu})$

We can find

$$\min_V \sum_{i=1}^{n} \left\| (X_i - \widehat{\mu}) - VV^\top (X_i - \widehat{\mu}) \right\|^2$$

where V is constrained to be orthogonal

This is the so called Steifel manifold of rank-K orthogonal matrices

The solution is given by the singular vectors V

(General) spectral connectivity analysis

Metric embeddings

Spectral connectivity analysis (SCA)

- Linear and nonlinear
- Dimension reduction or feature creation
- Examples: PCA, Locally linear embeddings, Hessian maps, ~~Laplacian eigenmaps~~
- Useful tools in classification, clustering, (regression)

PCA (review)

Collect data: $X_1, \ldots, X_n$ where $X_i \in \mathbb{R}^p$.

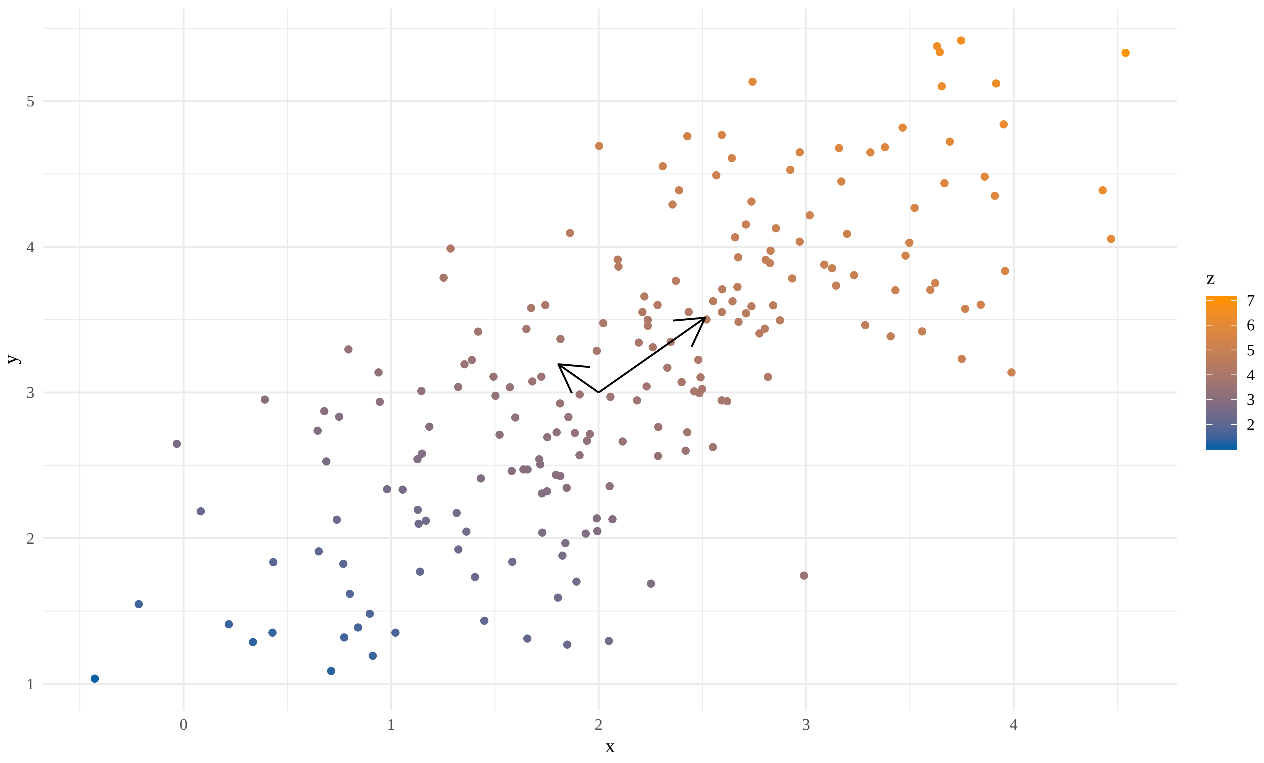
1. Center and (scale) the data matrix X
2. Compute the SVD of $X = U\Sigma V^\top$ or $XX^\top = U\Sigma^2 U^\top$ or $X^\top X = V\Sigma^2 V^\top$
3. Return $U_d \Sigma_d$, where Σ_d is the largest d eigenvalues of X

- You need to compute SVD of $XX^\top$ [or X]

When PCA works well

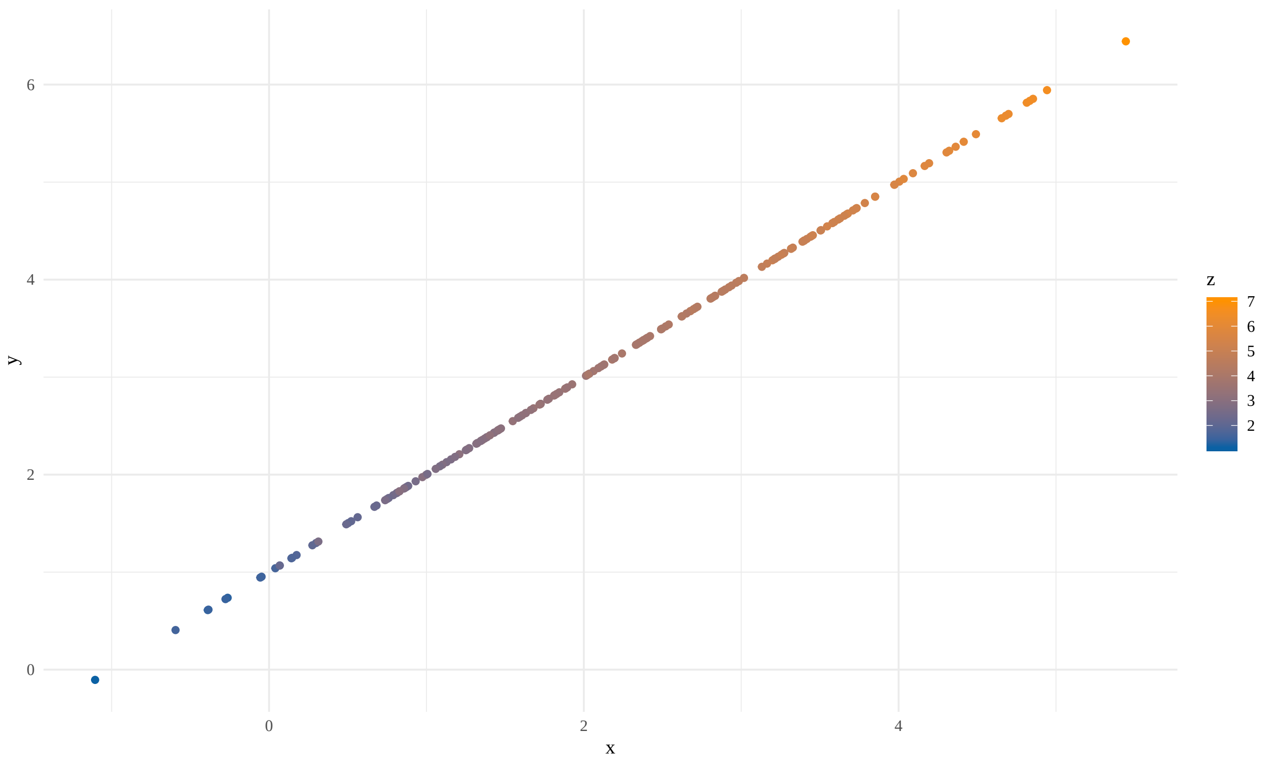
PCA "works" when the data can be represented (in a lower dimension) as 'lines' (or planes, or hyperplanes).

So, in two dimensions:



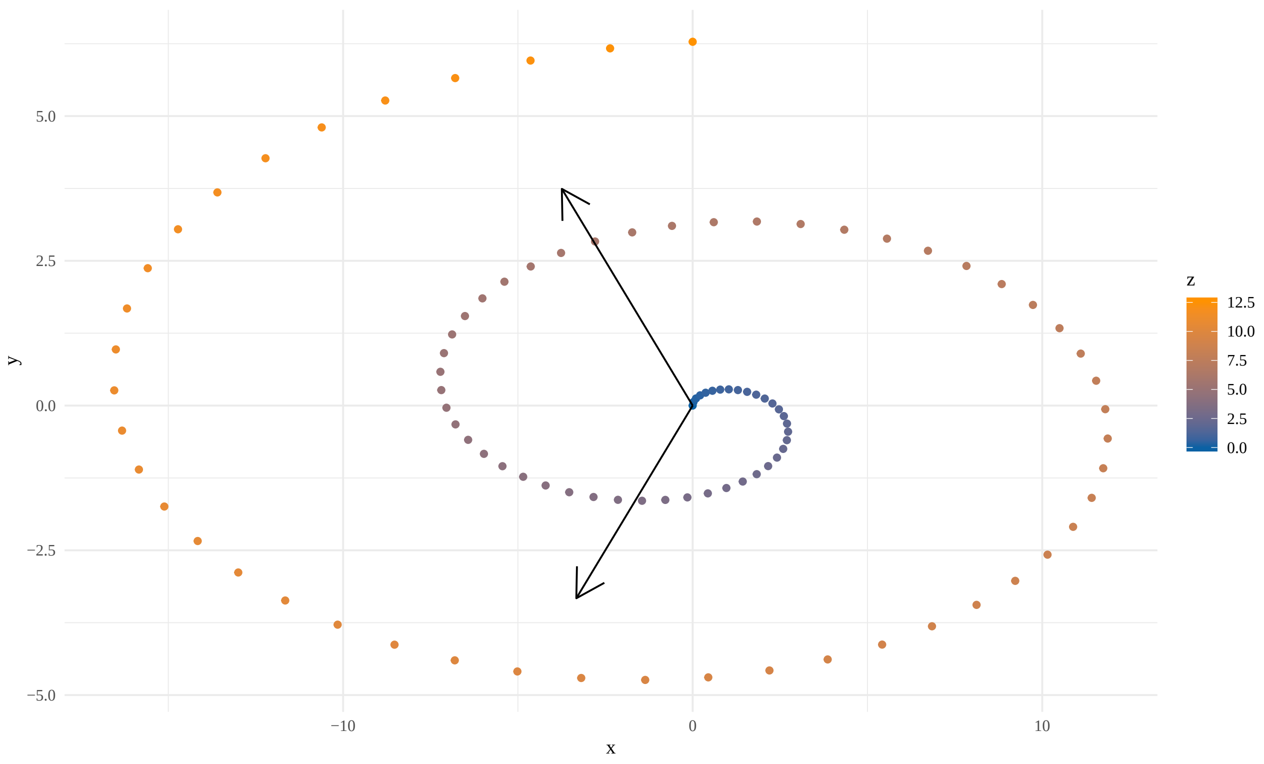
PCA reduced

Here, we can capture a lot of the variation and underlying ‘structure’ with just 1 dimension, instead of the original 2 (the coloring is for visualizing).



PCA bad

What about other data structures? Again in two dimensions

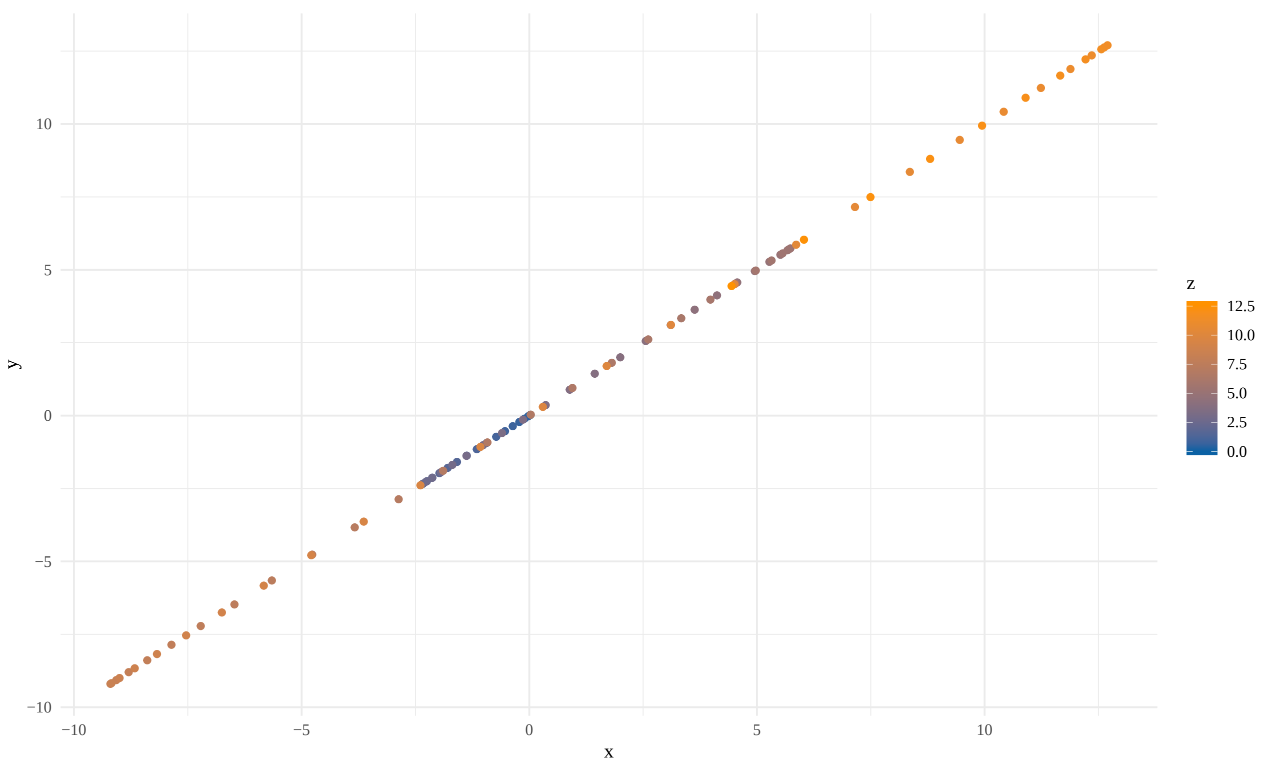


PCA bad (2)

Here, we have failed miserably.

There is actually only 1 dimension to this data (imagine walking up the spiral going from blue to orange).

However, when we write it as 1 PCA dimension, all the points are all 'mixed up'.



Explanation

- PCA wants to minimize distances (equivalently maximize variance). This means it 'slices' through the data at the 'meatiest' point, and then the next one, and so on. If the data are 'curved' this is going to induce artifacts.
- PCA also looks at things as being 'close' if they are near each other in a Euclidean sense.
- On the spiral, our intuition says that things are 'close' only if the distance is constrained to go along the curve. In other words, purple and blue are close, blue and orange are not.

Nonlinearity and CMDS

Kernel PCA (KPCA)

Classical PCA comes from $\tilde{X} = X - MX = UDV^\top$, where $M = \mathbf{1}\mathbf{1}^\top/n$ and $\mathbf{1} = (1, 1, \ldots, 1)^\top$

However, we can just as easily get it from the outer product

$$\mathbb{K} = \tilde{X}\tilde{X}^\top = (I - M)XX^\top(I - M) = UD^2U^\top$$

The intuition behind KPCA is that $\mathbb{K}$ is an expansion into a kernel space, where

$$\mathbb{K}_{i,i'} = k(\tilde{X}_i, \tilde{X}_{i'}) = \langle \tilde{X}_i, \tilde{X}_{i'} \rangle$$

Reminder: Anytime we see an inner product, we can kernelize it

Kernel PCA

Following this intuition, the approach is simple:

1. Specify a kernel k
   (e.g. $k(X, X') = \exp\{-\gamma^{-1} \left|\left|X - X'\right|\right|_2^2\}$)
2. Form $K_{i,i'} = k(X_i, X_{i'})$
3. Standardize and get eigenvector decomposition

$$\mathbb{K} = (I - M)K(I - M) = UD^2U^\top$$

This implicitly finds the inner product:

$$k(X_i, X_{i'}) = \langle \phi(X_i), \phi(X_{i'}) \rangle$$

However, we need only specify the kernel

Kernel PCA

The scores are still $Z = UD$

The q^{th} KPCA score is (up to centering)

$$Z_{iq} = \sum_{i'=1}^{n} \beta_{i'q} k(X_i, X_{i'})$$

where $\beta_{i',q} = u_{i'q}/d_q$

Note: As we don't explicitly generate the feature map, there are no loadings

Kernel PCA

Reminder: To get the first PC in classical PCA, we want to solve

$$\max_{\alpha} \mathbb{V}\alpha^\top X \quad \text{subject to} \quad \left|\left|\alpha\right|\right|_2^2 = 1$$

Translate this into the kernel setting, and we are trying to solve

$$\max_{g \in \mathcal{H}_k} \mathbb{V}g(X) \text{ subject to } \left|\left|g\right|\right|_{\mathcal{H}_k} = 1$$

The representer theorem states that a solution to this problem is

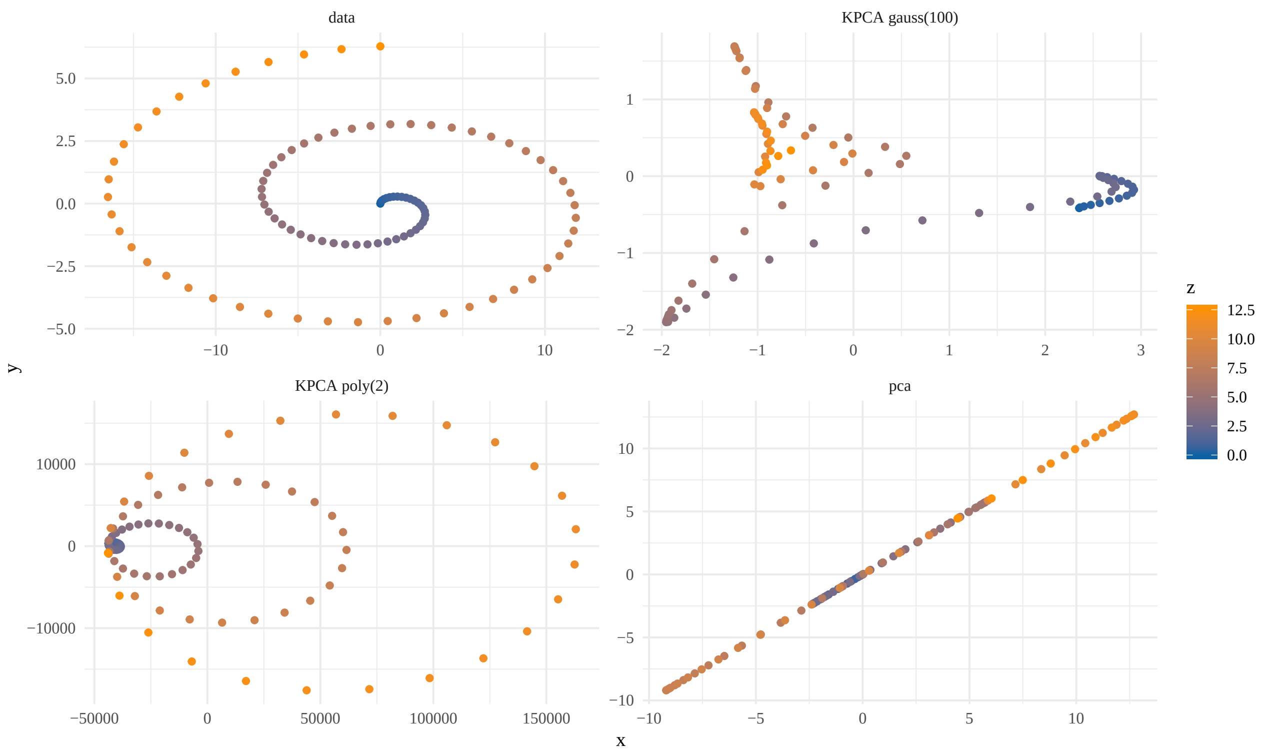
$$\widehat{g}(X) = \sum_{i=1}^{n} \beta_i k(X, X_i)$$

Compare

$$Z_{iq} = \sum_{i'=1}^{n} \beta_{i'q} k(X_i, X_{i'})$$

where $\beta_{i',q} = u_{i'q}/d_q$

KPCA example



KPCA: summary

Kernel PCA seeks to generalize the notion of similarity using a kernel map

This can be interpreted as finding smooth, orthogonal directions in a RKHS

This can allow us to start picking up nonlinear (in the original feature space) aspects of our data

This new representation can be passed to a supervised method to form a semisupervised learner

Semi-supervised detour

Basic semi-supervised

1. You get data $\mathcal{D}_{train} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$.
2. You do something unsupervised on the X's to create features (like PCA).
3. You use the learned features to find a predictor $\widehat{f}$ using the new features.

Semisupervised learning in practice

Looking at:

$$Z_{iq} = \sum_{i'=1}^{n} \beta_{i'q} k(X_i, X_{i'})$$

this is only defined at our observed features

Write

- $\mathcal{D}_{train} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$
- $\mathcal{D}_{test} = \{(X_1^*, Y_1^*), \ldots, (X_{n^*}^*, Y_{n^*}^*)\}$

Two common scenarios are

1. We are given $\mathcal{D}_{train}$ and $X_1^*, \ldots, X_{n^*}^*$ to build $\widehat{f}$
2. We are given only $\mathcal{D}_{train}$ to build $\widehat{f}$

Case 1

We are given $\mathcal{D}_{train}$ and $X_1^*, \ldots, X_{n^*}^*$ to build $\widehat{f}$

Then we can just use straight forward KPCA

(Or any unsupervised learning step)

1. Form $\mathbb{K}$ on $\mathcal{D}_{train}$ and $X_1^*, \ldots, X_{n^*}^*$
2. Get UD
3. Pass $Z_q = UD[, 1 : q]$ to train $\widehat{f}$
4. Get $\widehat{Y} = \widehat{f}(Z_q)$

Case 2

We are given only $\mathcal{D}_{train}$ to build $\widehat{f}$

Now, we don't know the coordinates of $X_1^*, \ldots, X_{n^*}^*$ in the representation space

To get a new observation X^* embedded into this representation:

$$Z_0 = D^{-1}U^\top(I - M)[k^* - K\mathbf{1}/n]$$

where $k^* = [k(X^*, X_1), \ldots, k(X^*, X_n)]^\top$

Then we compute:

1. Form $\mathbb{K}$ on $\mathcal{D}_{train}$
2. Get UD
3. Pass $Z_q = UD[, 1 : q]$ to train $\widehat{f}$
4. Form Z_q^* for all $X_1^*, \ldots, X_{n^*}^*$
5. Get $\widehat{Y}_{test} = \widehat{f}(Z_q^*)$

Classical multidimensional scaling

CMDS

A broader class comes from the following procedure (see Figure 4.2 in the text):

1. Calculate a matrix of distances (or dissimilarities) between data points (Δ)

2. Choose some function $\tau : \mathbb{R} \to \mathbb{R}$ and set $B = \tau(\Delta^2)$.
3. Write $B = U \Sigma U^\top$ (find the eigendecomposition)
4. Approximate your data with $U_{[d]} \Sigma_{[d]}^{1/2}$ where $[d]$ means "the first d columns"

CMDS (I think) is steps 2-4: embedding "dissimilarities" by using the eigendecomposition.

What I'll call Laplacian Eigenmaps

We can get an estimate of the distance in the unknown geometry that the data come from (known as a 'nonlinear' manifold) by altering the usual Euclidean distance.

In this case, I'll use the heat kernel (like most everyone) but this is not necessary.

Distance between x and y:

- PCA: $\|x - y\|_2$
- Laplacian eigenmaps : $\exp\left(-\frac{\|x-y\|_2^2}{\epsilon}\right)$

Some notes:

- The name 'Laplacian Eigenmaps' comes from getting the eigenvector decomposition of the Laplacian restricted to the manifold (which is the second derivative version of the gradient). This gives crucial information about its geometry.
- The form of the distance says that if $\|x - y\|_2^2$ is much bigger than ϵ, then the distance is effectively zero. This encodes the idea that things that are 'close' in Euclidean distance are 'close' on the manifold as well.
- If the manifold is smooth, then local Euclidean distance is an approximation to the distance on the manifold.

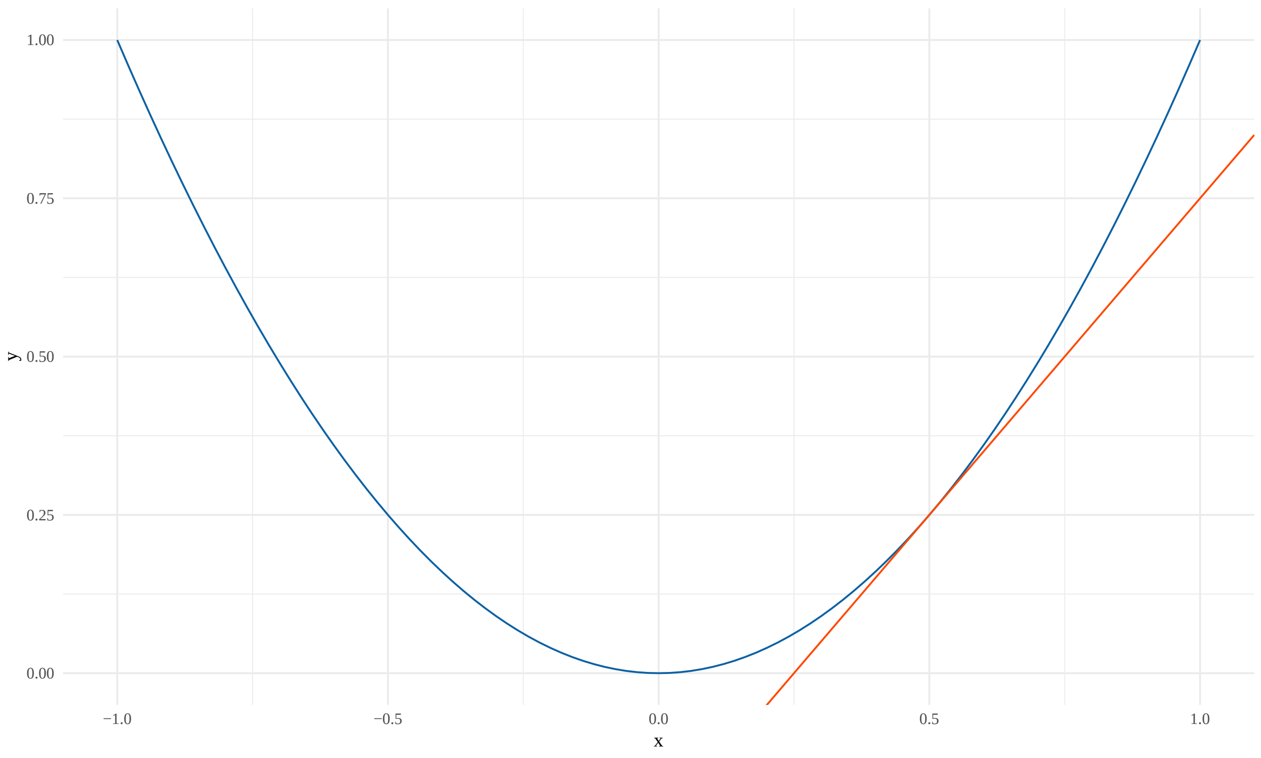
What is a manifold?

How good of an approximation is Euclidean distance?

This question is equivalent to how asking: how quickly does the tangent (space) change?

In 1-D, the tangent space is just the first derivative at that point:

$$f(x) = x^2 \Rightarrow f'(x) = 2x.$$



What is a manifold

Therefore, the quality of the (local) Euclidean distance, depends on the second derivative

(ie: how fast does the first derivative change?)

In higher dimensions, the second derivative is known as the **Laplacian**:

$$\sum_j \frac{\partial^2 f}{\partial x_j^2}$$

(Note: This is also known as the divergence of the gradient)

What are Laplacian Eigenmaps, then?

Imagine the operator $\mathbb{L}$ that performs this operation:

$$\mathbb{L}f = \sum_j \frac{\partial^2 f}{\partial x_j^2}$$

Then $\mathbb{L}$ is the **Laplacian**, mapping a function to the divergence of its gradient

- **Key Idea:** We can get the eigenvectors/eigenvalues of $\mathbb{L}$. Analogously to PCA, we can now do inference with these eigenvectors.

Note: There is a substantial overlap with KPCA, the difference being the centering of K and the row sum versus column sum normalization

Laplacian Eigenmaps (procedure)

Collect data: $X_1, \ldots, X_n$ where $X_i \in \mathbb{R}^p$.

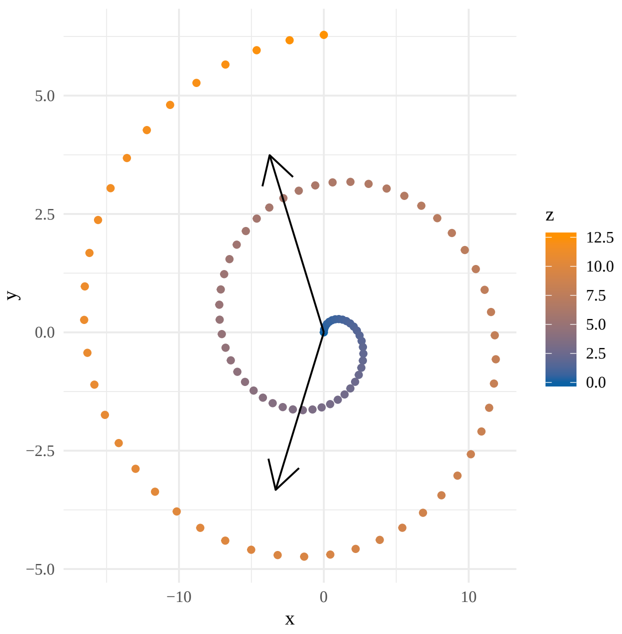
1. Center and scale the data matrix X
2. Compute $\mathbb{K}$ where
$$\mathbb{K}_{ij} = \exp\left(-\frac{\|X_i - X_j\|_2^2}{\epsilon}\right)$$
3. Form the Laplacian $\mathbb{L} = \mathbb{I} - \mathbb{M}^{-1}\mathbb{K}$ where $\mathbb{M} =$ `diag(rowSums(`$\mathbb{K}$`))`
4. Compute the SVD of $\mathbb{L} = U\Sigma U^\top$.
5. Return $U_d\Sigma_d^{-1}$, where Σ_d contains the **smallest** d nonzero eigenvalues of $\mathbb{L}$

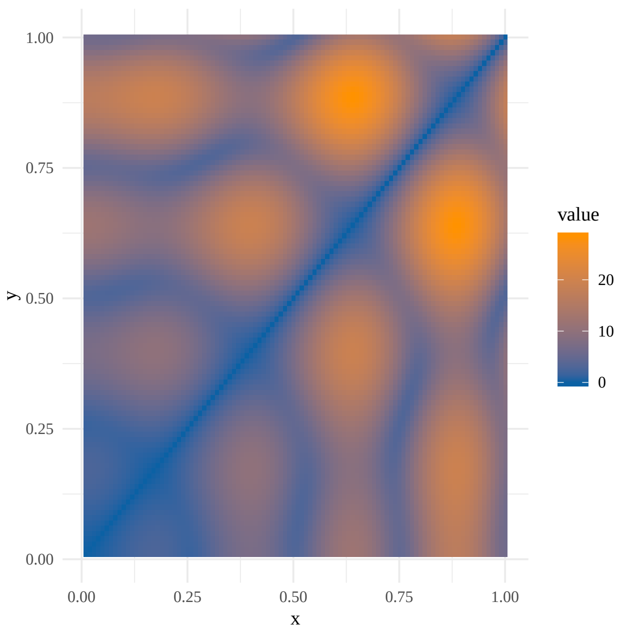
   (Note that the eigenvectors of $\mathbb{L}$ and $\mathbb{M}^{-1}\mathbb{K}$ are the same, but $\Sigma(\mathbb{L}) = \mathbb{I} - \Sigma(\mathbb{M}^{-1}\mathbb{K})$)

- This is basically just CMDS on $\tau(\Delta)$.
- ~~You need to compute SVD of $\mathbb{L}$.~~

Deeper investigation 1.

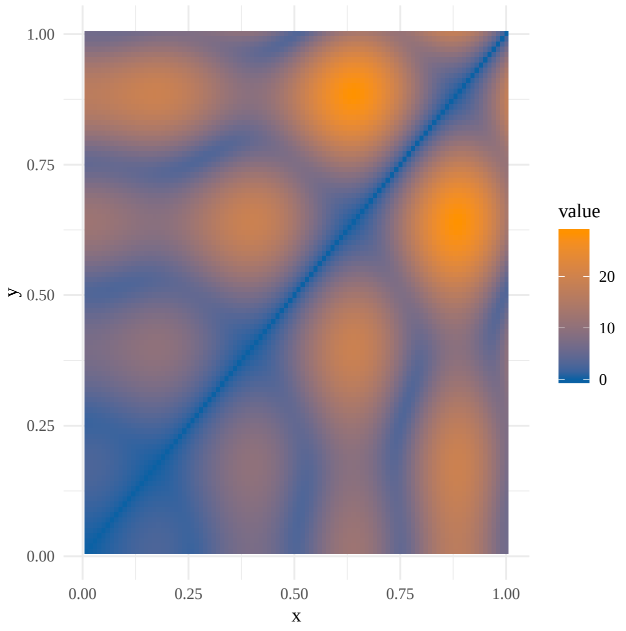
The distance matrix

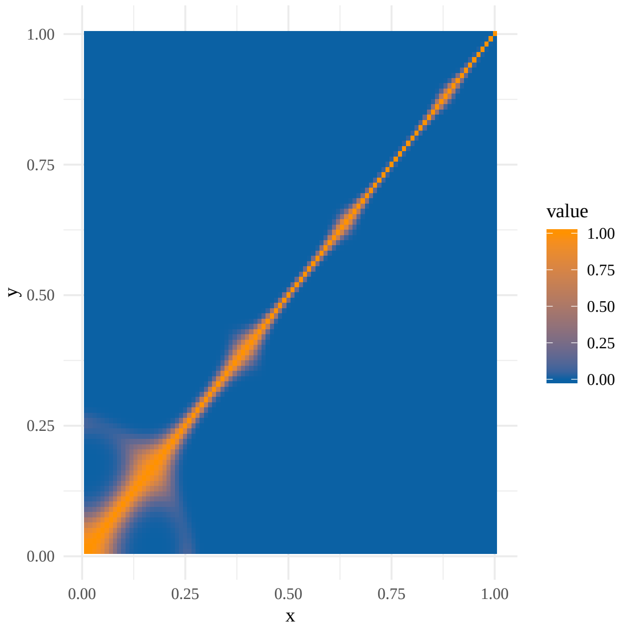




More deeper

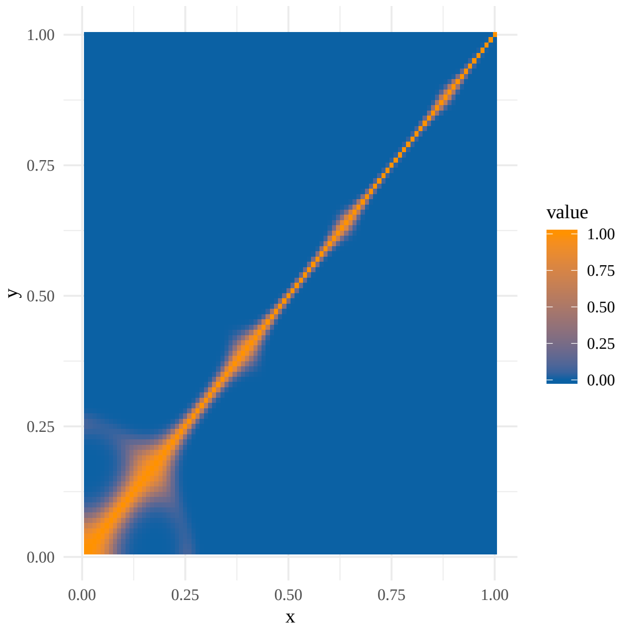
Exponentiate $-\Delta/\gamma$

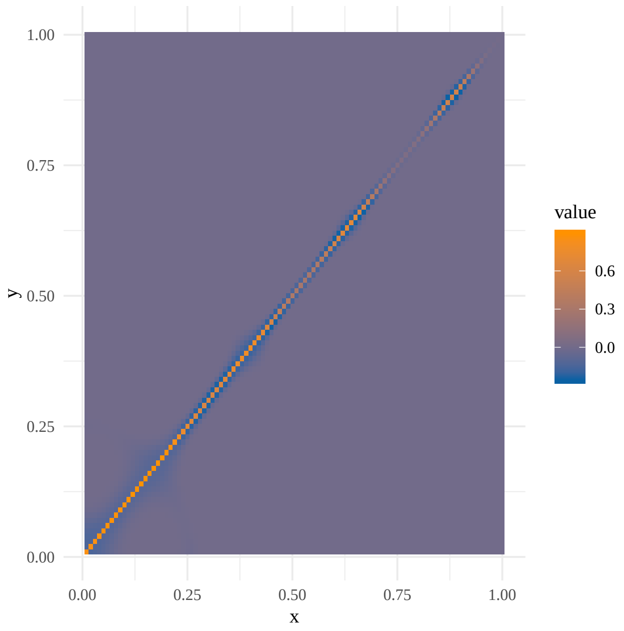




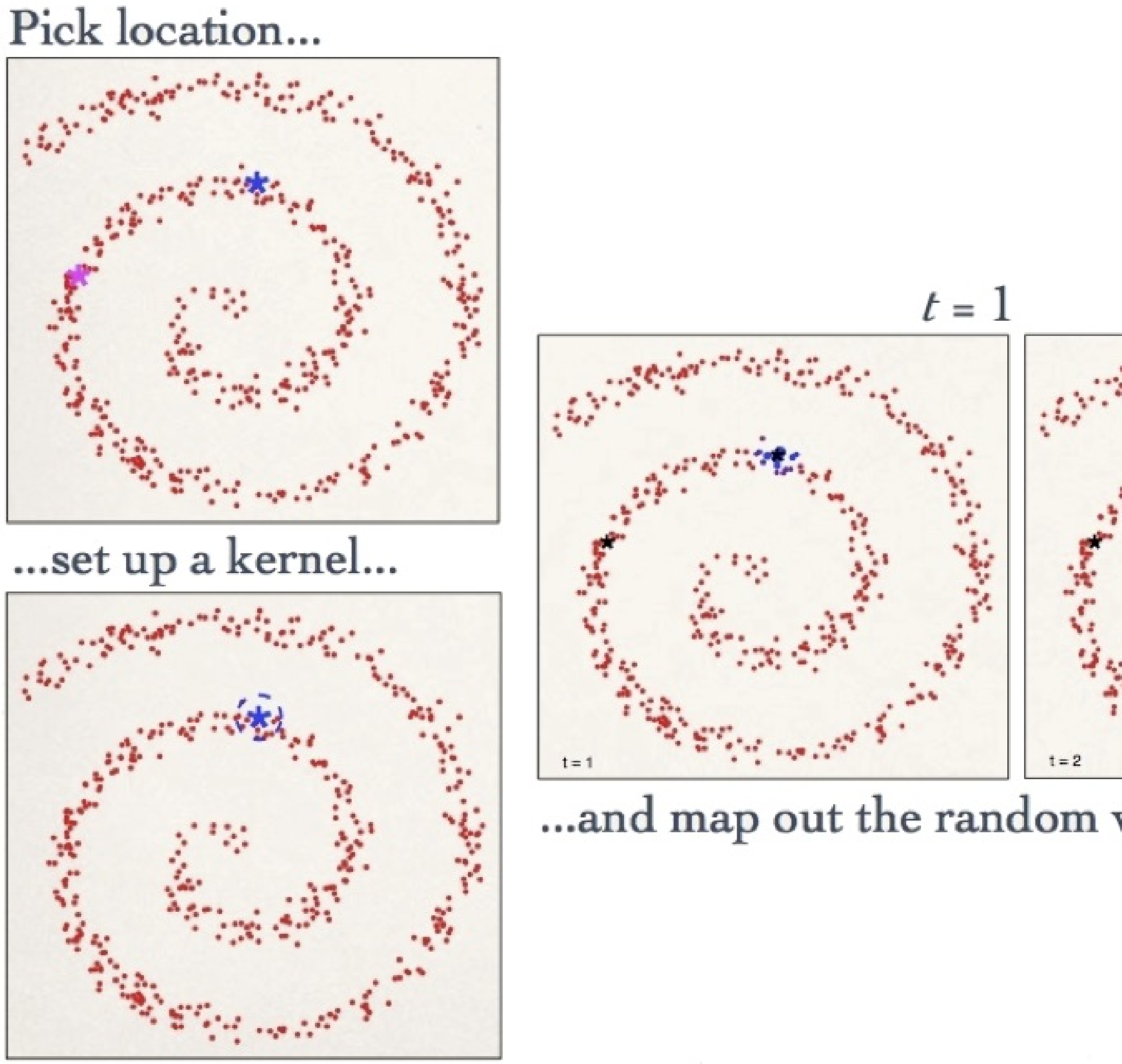
More deeperer

Form Laplacian: $\mathbb{L} = \mathbb{I} - \mathbb{M}^{-1}\mathbb{K}$

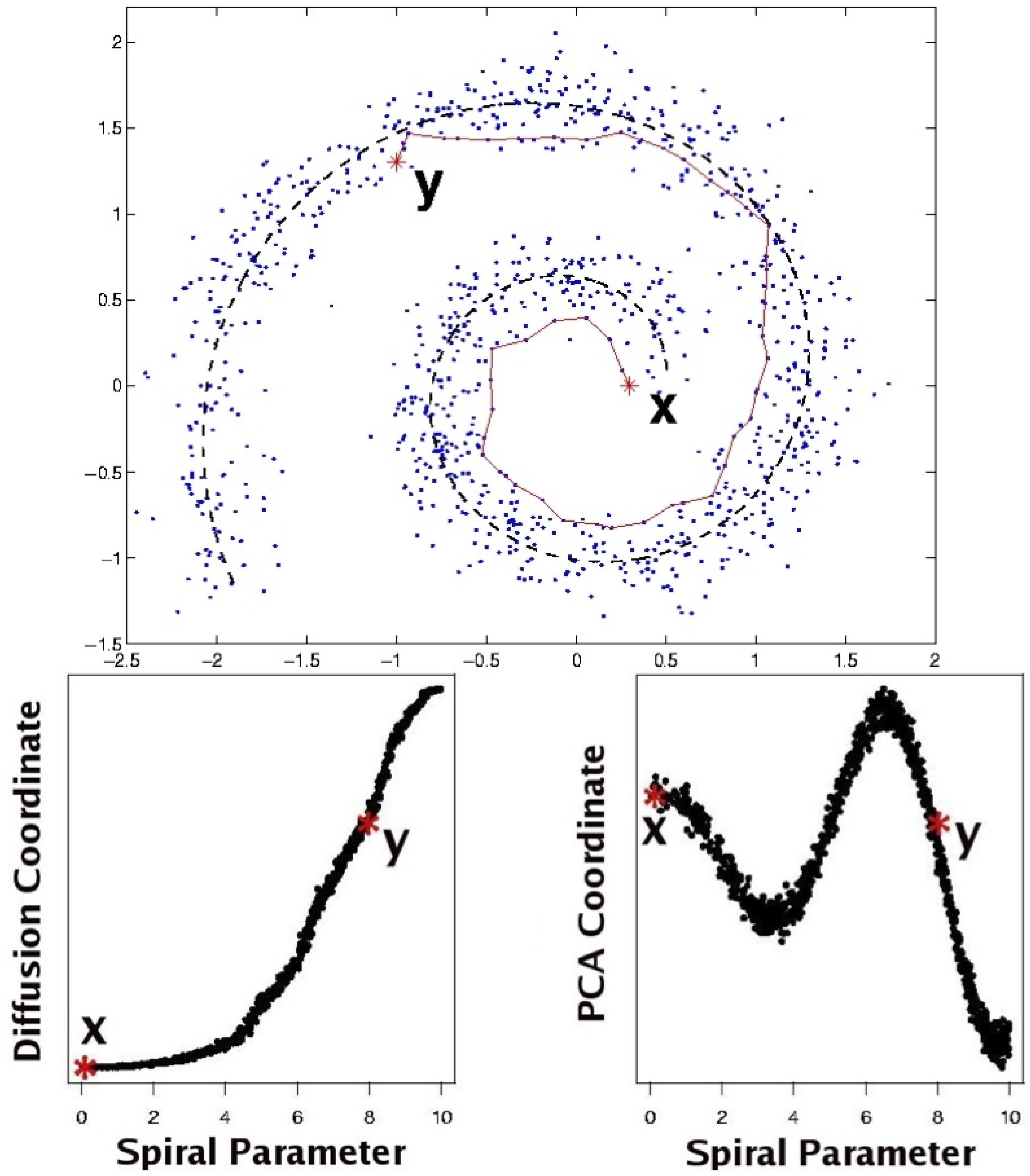




Diffusion distance



Projection



y
x
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2.5
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
Diffusion Coordinate
Spiral Parameter
x
y
0
2
4
6
8
10
PCA Coordinate
Spiral Parameter
x
y
0
2
4
6
8
10

Another example

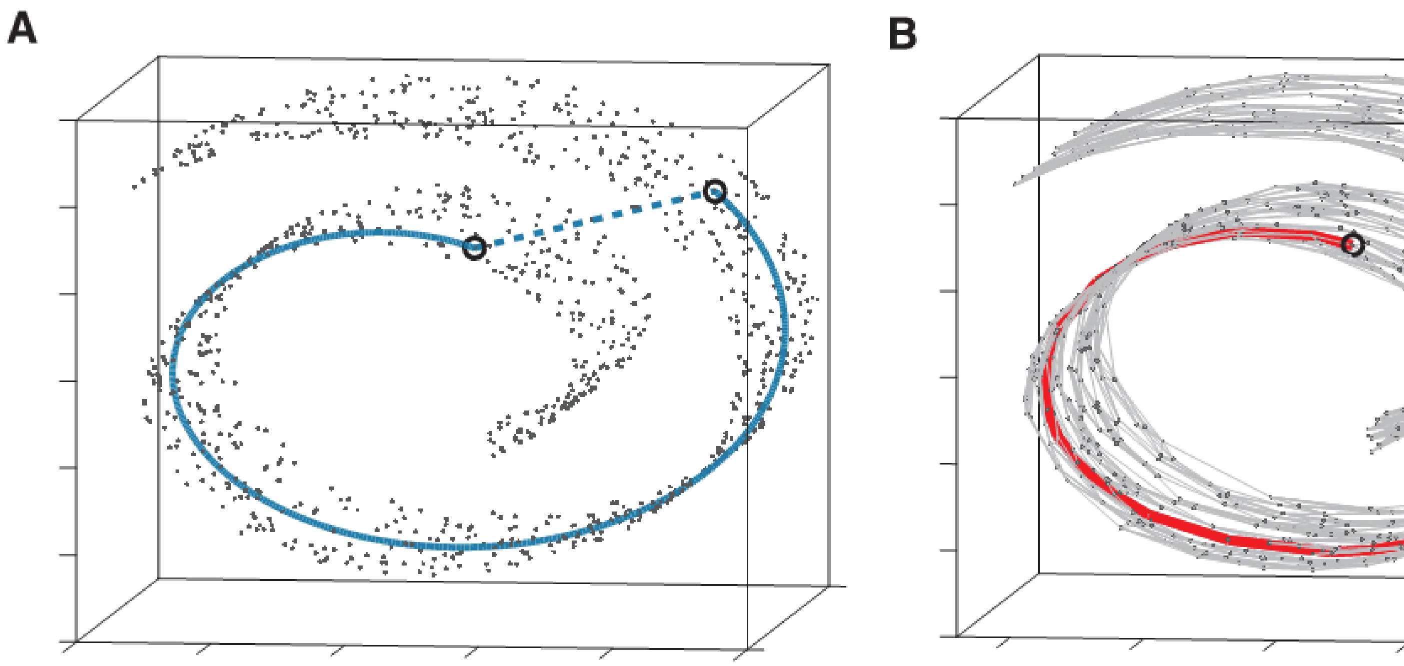


Fig. 3. The "Swiss roll" data set, illustrating how Isomap exploits geodesic paths for nonlinear dimensionality reduction. (**A**) For two arbitrary points (circled) on a nonlinear manifold, their Euclidean distance in the high-dimensional input space (length of dashed line) may not accurately reflect their intrinsic similarity, as measured by geodesic distance along the low-dimensional manifold (length of solid curve). (**B**) The neighborhood graph *G* constructed in step one of Isomap (with *K* = 7 and *N* =

1000 geod path step neigh now path

Clustering whirlwind

K-means

K-means

1. Select a number of clusters K.
2. Let $C_1, \ldots, C_K$ partition $\{1, 2, 3, \ldots, n\}$ such that
    - All observations belong to some set C_j.
    - No observation belongs to more than one set.

3. K-means attempts to form these sets by making within-cluster variation, $W(C_k)$, as small as possible.

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k).$$

4. To Define W, we need a concept of distance. By far the most common is Euclidean

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|X_i - X_{i'}\|_2^2 .$$

That is, the average (Euclidean) distance between all cluster members.

K-means

It turns out

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k).$$

is too hard of a problem to solve computationally (K^n partitions!).

So, we make a greedy approximation:

1. Randomly assign observations to the K clusters
2. Iterate until the cluster assignments stop changing:
   - For each of K clusters, compute the centroid, which is the p-length vector of the means in that cluster.
   - Assign each observation to the cluster whose centroid is closest (in Euclidean distance).

This procedure is guaranteed to decrease (1) at each step.

K-means: A Summary

To fit K-means, you need to

1. Pick K (inherent in the method)
2. Convince yourself you have found a good solution (due to the randomized approach to the algorithm).

It turns out that 1. is difficult to do in a principled way. We will discuss this next

For 2., a commonly used approach is to run K-means many times with different starting points. Pick the solution that has the smallest value for

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k)$$

Choosing the Number of Clusters

Why is it important?

- It might make a big difference (concluding there are $K = 2$ cancer sub-types versus $K = 3$).
- One of the major goals of statistical learning is automatic inference. A good way of choosing K is certainly a part of this.

Reminder: What does K-means do?

Given a number of clusters K, we (approximately) minimize:

$$\sum_{k=1}^{K} W(C_k) = \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} ||X_i - X_{i'}||_2^2.$$

We can rewrite this in terms of the centroids as

$$W(K) = \sum_{k=1}^{K} \sum_{i \in C_k} ||X_i - \overline{X}_k||_2^2,$$

Minimizing W in K

Of course, a lower value of W is better. Why not minimize W?

Within-cluster variation measures how tightly grouped the clusters are.

As we increase K, this will always decrease.

What we are missing is between-cluster variation, ie: how spread apart the groups are

$$B = \sum_{k=1}^{K} |C_k| ||\overline{X}_k - \overline{X}||_2^2,$$

where $|C_k|$ is the number of points in C_k, and $\overline{X}$ is the grand mean of all observations:

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

Sadly, just like W can be made arbitrarily small, B will always be increasing with increasing K.

CH index

Ideally, we would like our cluster assignment to simultaneously have small W and large B.

This is the idea behind CH **index**. For clustering assignments coming from K clusters, we record CH score:

$$CH(K) = \frac{B(K)/(K-1)}{W(K)/(n-K)}$$

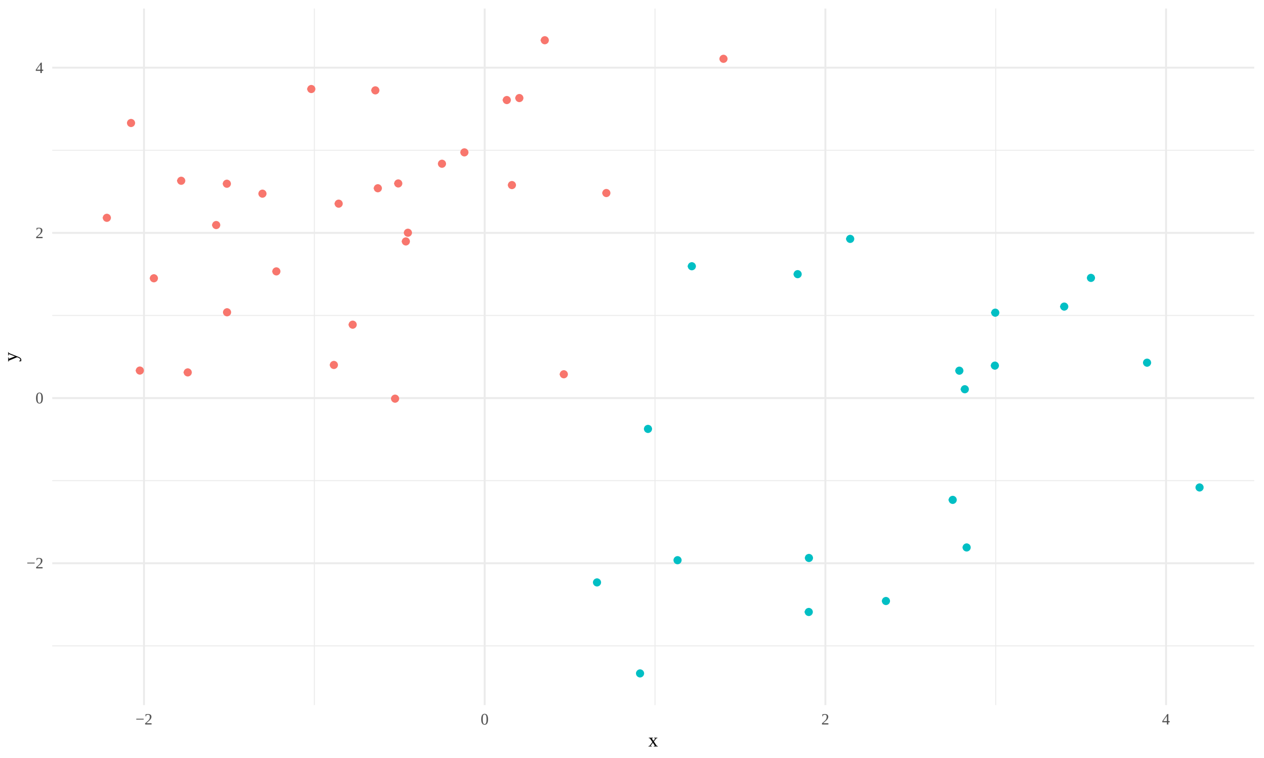
To choose K, pick some maximum number of clusters to be considered ($K_{\max} = 20$, for example) and choose the value of K that

$$\widehat{K} = \arg \max_{K \in \{2, \ldots, K_{\max}\}} CH(K).$$

Note: CH is undefined for $K = 1$ (see Calinski, Harabasz, 1974)

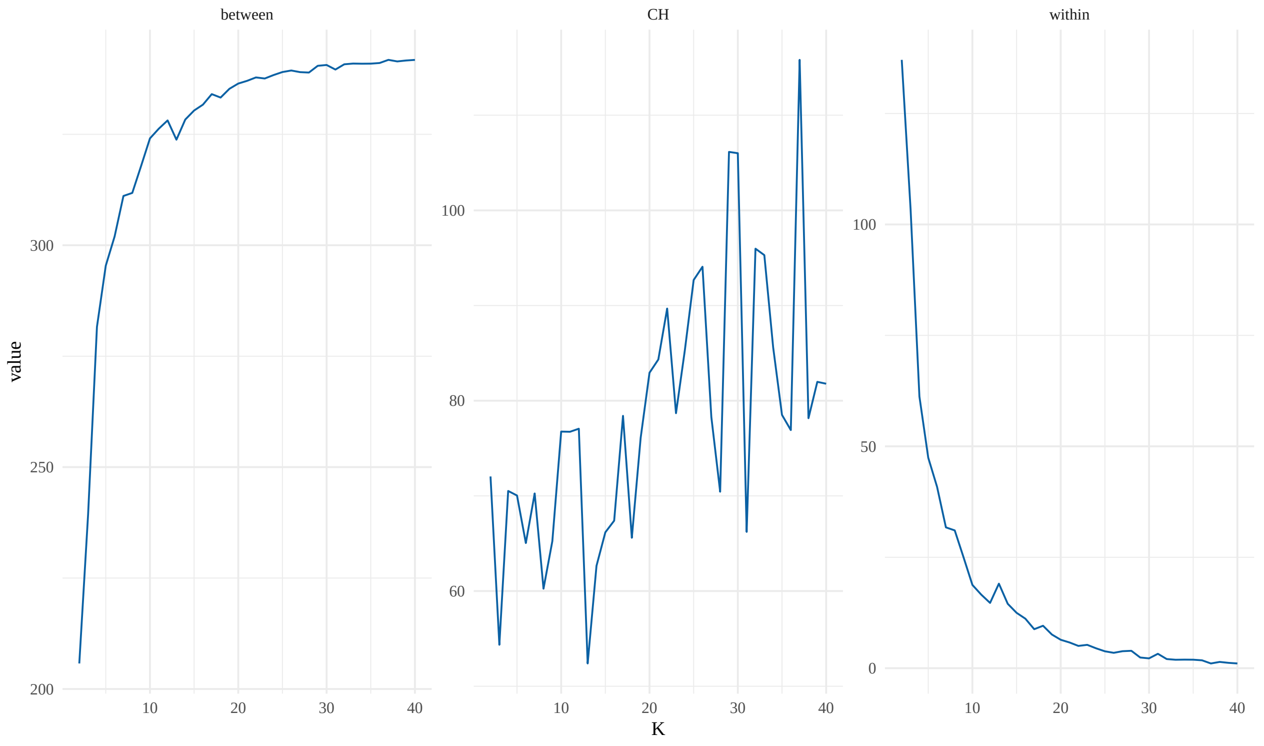
Dumb example

- There are 2 clusters



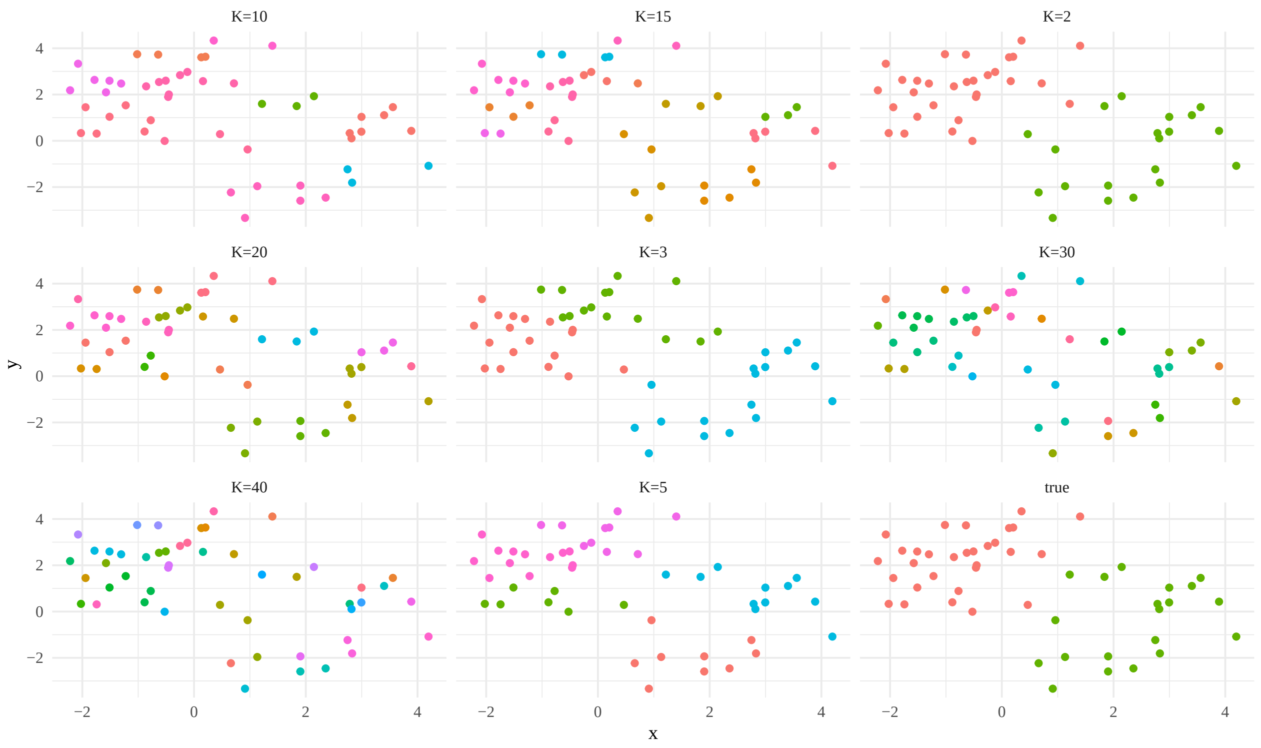
Dumb example

- Within, between, and CH index
- We would **maximize** CH



Dumb example

- Various solutions



Hierarchical clustering

From K-means to hierarchical clustering

Recall two properties of K-means clustering

1. It fits exactly K clusters.
2. Final clustering assignments depend on the chosen initial cluster centers.

Alternatively, we can use hierarchical clustering. This has the advantage that

1. No need to choose the number of clusters before hand.
2. There is no random component (nor choice of starting point).

There is a catch: we need to choose a way to measure the distance between clusters, called the **linkage**.

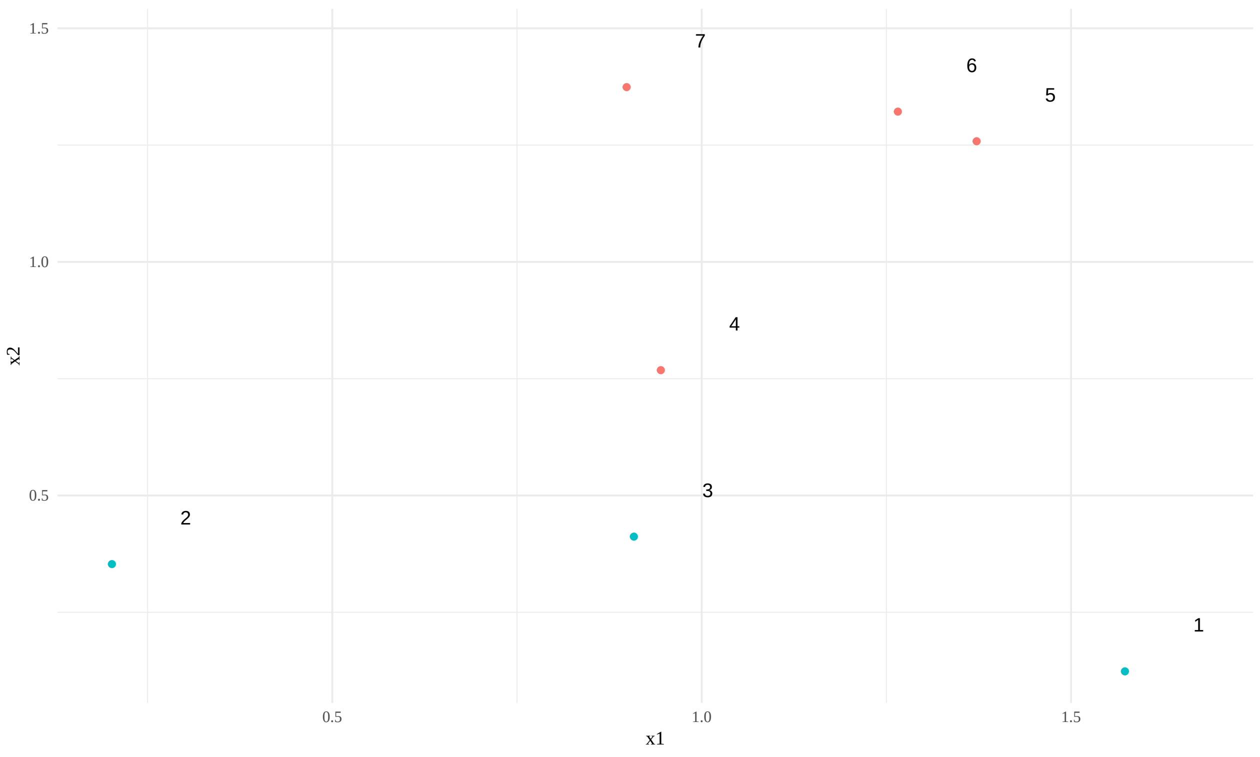
Given the linkage, hierarchical clustering produces a sequence of clustering assignments.

At one end, all points are in their **own** cluster.

At the other, all points are in **one** cluster.

In the middle, there are **nontrivial** solutions.

Agglomerative example



Given these data points, an agglomerative algorithm chooses a cluster sequence by combining the points into groups.

We can also represent the sequence of clustering assignments as a dendrogram

Cutting the dendrogram horizontally partitions the data points into clusters

Linkages

Notation: Define $X_1, \dots, X_n$ to be the data

Let the dissimiliarities be d_{ij} between each pair X_i, X_j

At any level, clustering assignments can be expressed by sets $G = \{i_1, i_2, \dots, i_r\}$. given the indices of points in this group. Define $|G|$ to be the size of G.

Linkage: The function $d(G, H)$ that takes two groups G, H and returns the linkage distance between them.

Agglomerative clustering, given the linkage:

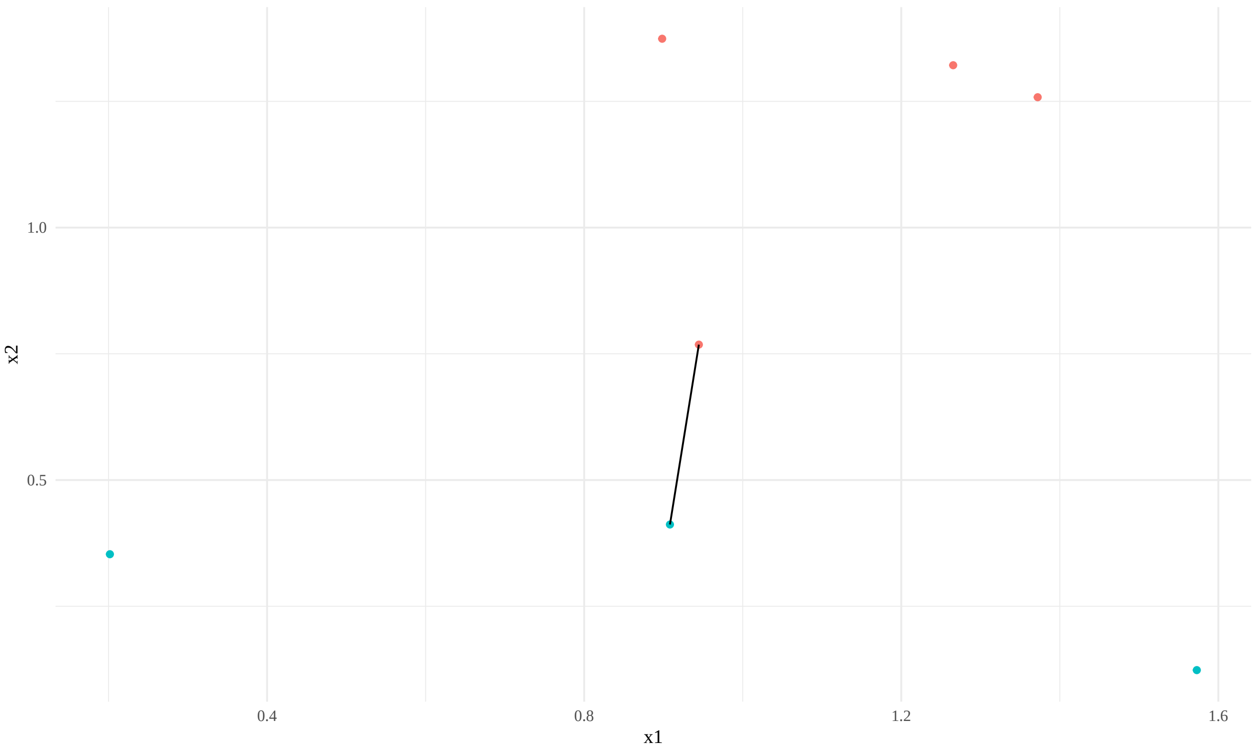
- Start with each point in its own group
- Until there is only one cluster, repeatedly merge the two groups G, H that minimize $d(G, H)$.

Single linkage

In **single linkage** (a.k.a nearest-neighbor linkage), the linkage distance between G, H is the smallest dissimilarity between two points in different groups:

$$d_{\text{single}}(G, H) = \min_{i \in G,\, j \in H} d_{ij}$$

Example:

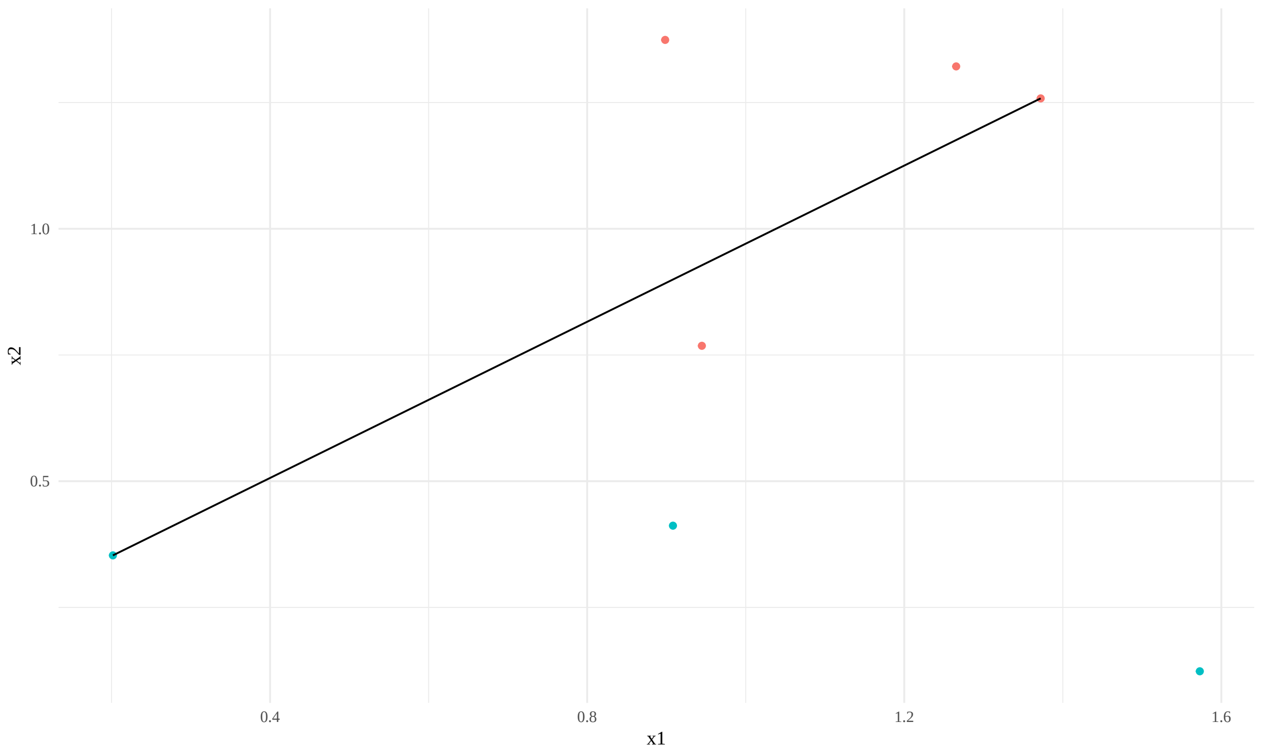


Complete linkage

In **complete linkage** (i.e. farthest-neighbor linkage), linkage distance between G, H is the largest dissimilarity between two points in different clusters:

$$d_{\text{complete}}(G, H) = \max_{i \in G,\, j \in H} d_{ij}.$$

Example:

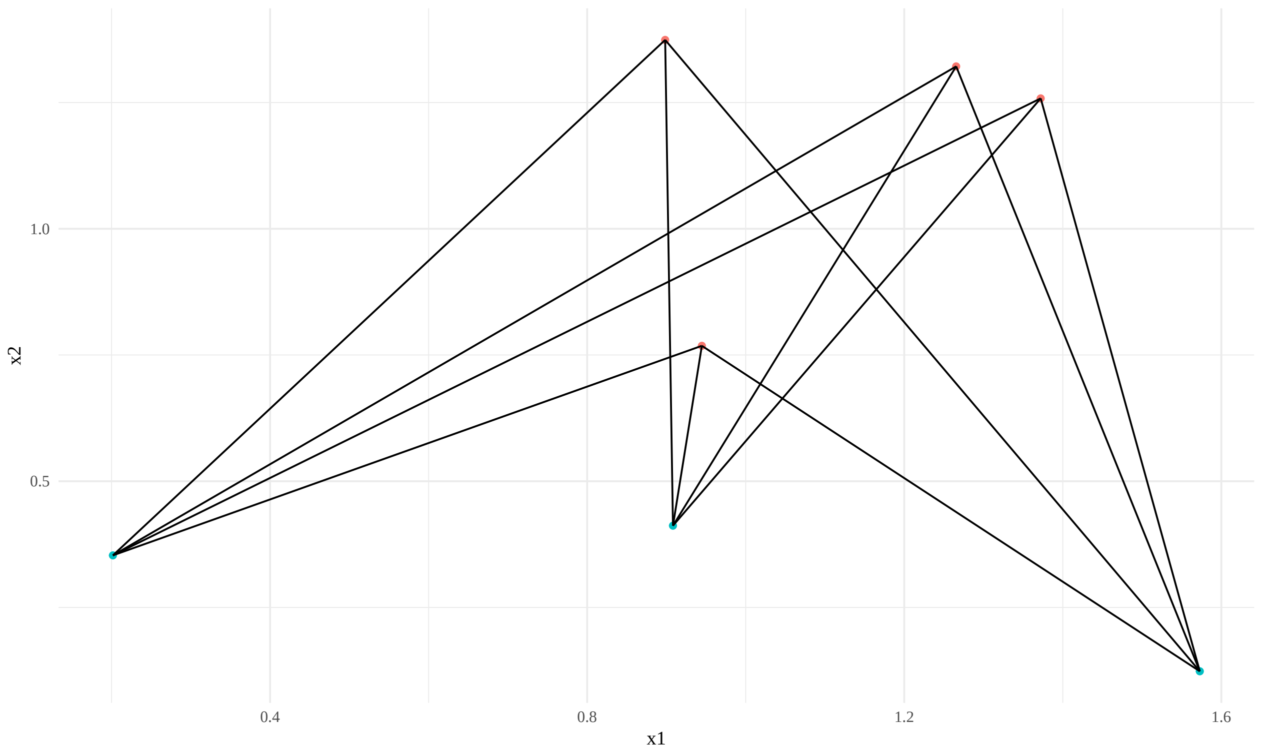


Average linkage

In **average linkage**, the linkage distance between G, H is the average dissimilarity over all points in different clusters:

$$d_{\text{average}}(G, H) = \frac{1}{|G| \cdot |H|} \sum_{i \in G,\, j \in H} d_{ij}.$$

Example:



Common properties

Single, complete, and average linkage share the following:

- They all operate on the dissimilarities d_{ij}. This means that the points we are clustering can be quite general (number of mutations on a genome, polygons, faces, whatever).
- Running agglomerative clustering with any of these linkages produces a dendrogram with no inversions
- No inversions means that the linkage distance between merged clusters only increases as we run the algorithm.

In other words, we can draw a proper dendrogram, where the height of a parent is always higher than the height of either daughter.

(We'll return to this again shortly)

Centroid linkage

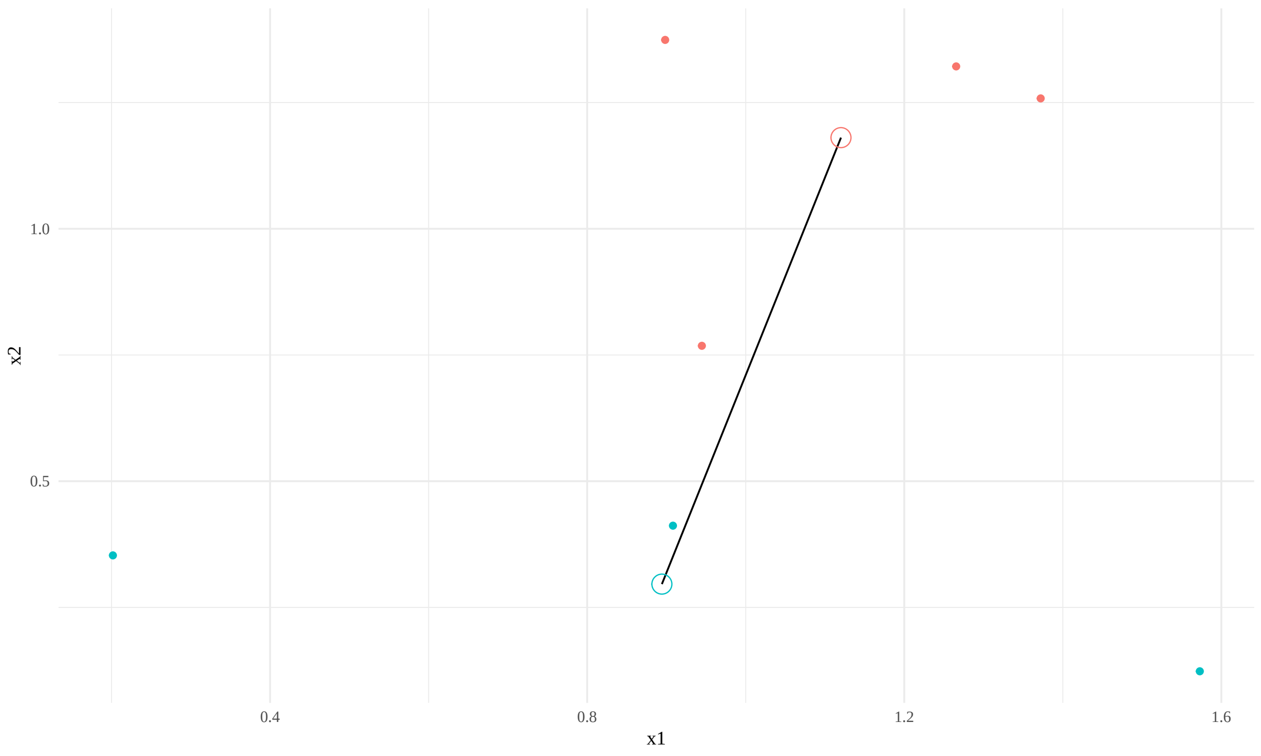
Centroid linkage is a commonly used and relatively new approach. Assume

- $X_i \in \mathbb{R}^p$
- $d_{ij} = ||X_i - X_j||_2^2$

Let $\overline{X}_G$ and $\overline{X}_H$ denote group averages for G, H. Then

$$d_{\text{centroid}} = ||\overline{X}_G - \overline{X}_H||_2^2$$

Example:



Centroid linkage

Centroid linkage is

- … quite intuitive
- … nicely analogous to K-means.
- … very related to average linkage (and much, much faster)

However, it has may introduce inversions. **inversions**

Shortcomings of some linkages

- **Single**: suffers from chaining: a single pair of close points merges two clusters. Therefore, clusters can be too spread out and not compact enough.
- **Complete linkage:** suffers from crowding: a point can be closer to points in other clusters than to points in its own cluster. Therefore, the clusters are compact, but not far enough apart.
- **Average linkage:** tries to strike a balance between these two. But…
  - It isn't clear what properties the resulting clusters have when we cut an average linkage tree.
  - Results of average linkage clustering can change with a monotone increasing transformation of the dissimilarities (that is, if we changed the distance, but maintained the ranking of the distances, the cluster solution could change).
  - Neither of these problems afflict single or complete linkage.
- **Centroid linkage:**
  - same monotonicity problem

- still the cut issues
- and inversions

Distances

Note how all the clustering depends on the distance functions.

Can do lots of things besides Euclidean

This is very important