Overview and applications of dimensionality reduction

Samuele Soraggi

Data Scientist
Bioinformatics Research Center,
Aarhus University

Overview and applications of dimensionality reduction

Slides here
https://samuelesoraggi.github.io/Projection_and_clustering_tutorial/

Content/Objectives:

time

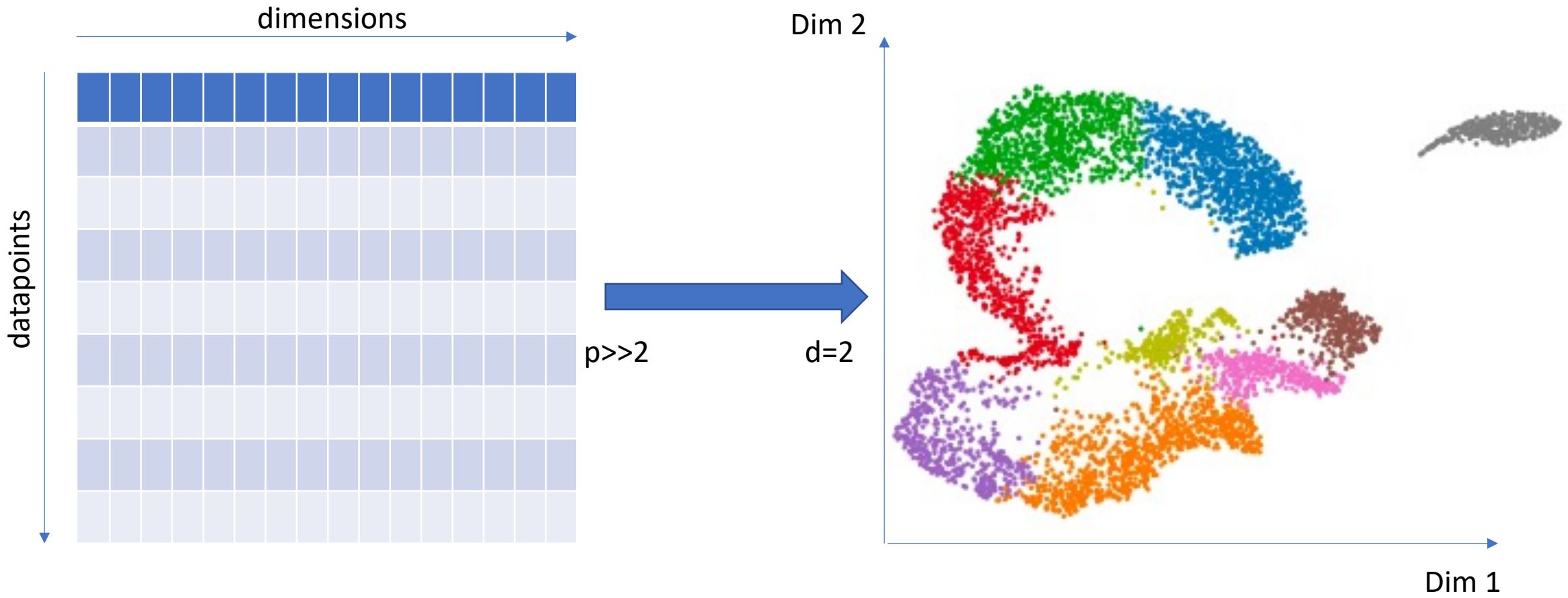
- Recognize types of dim.red. methods
- Master principles of how they work
- Some examples and practical issues

- Manifold learning: tSNE & UMAP
- Computational improvements of tSNE

- Conclusions
- Break
- Exercise illustration and setup

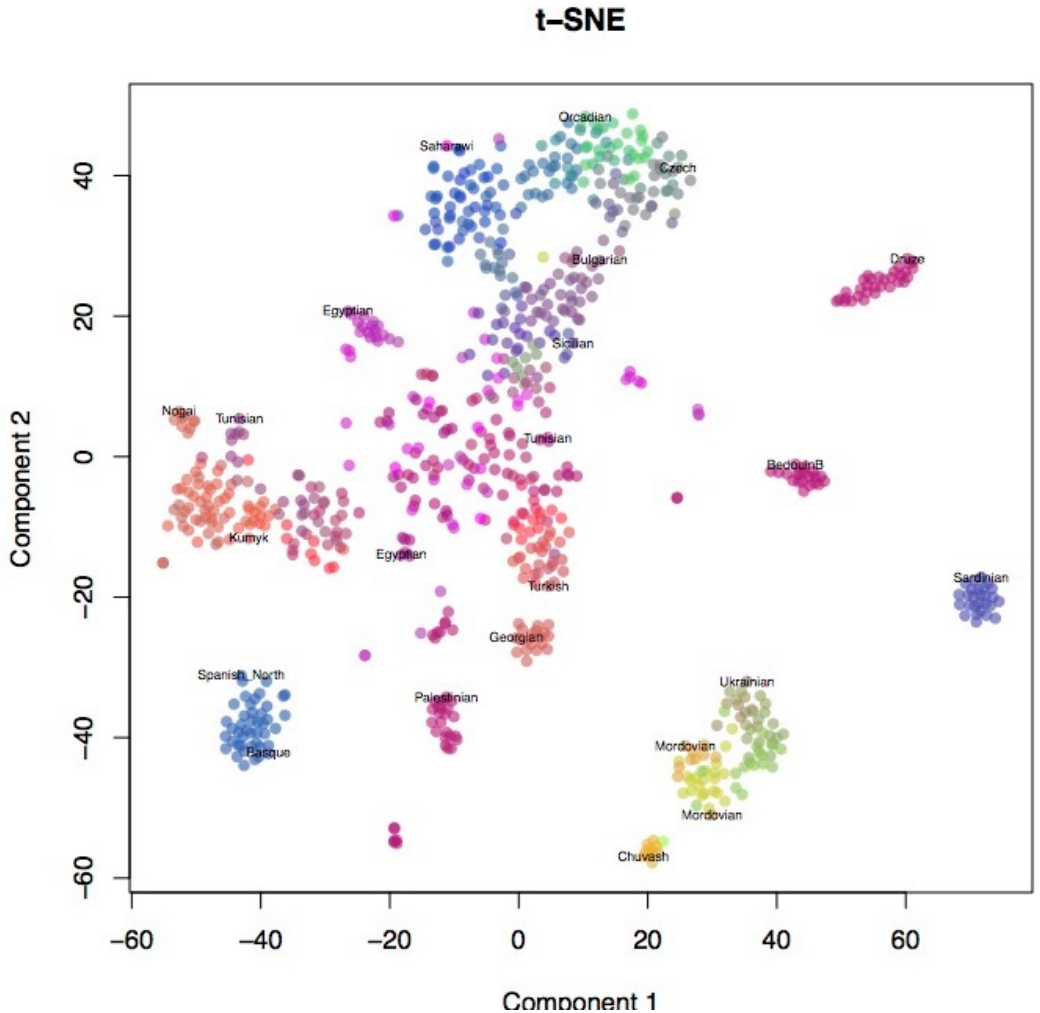
Dimensionality reduction

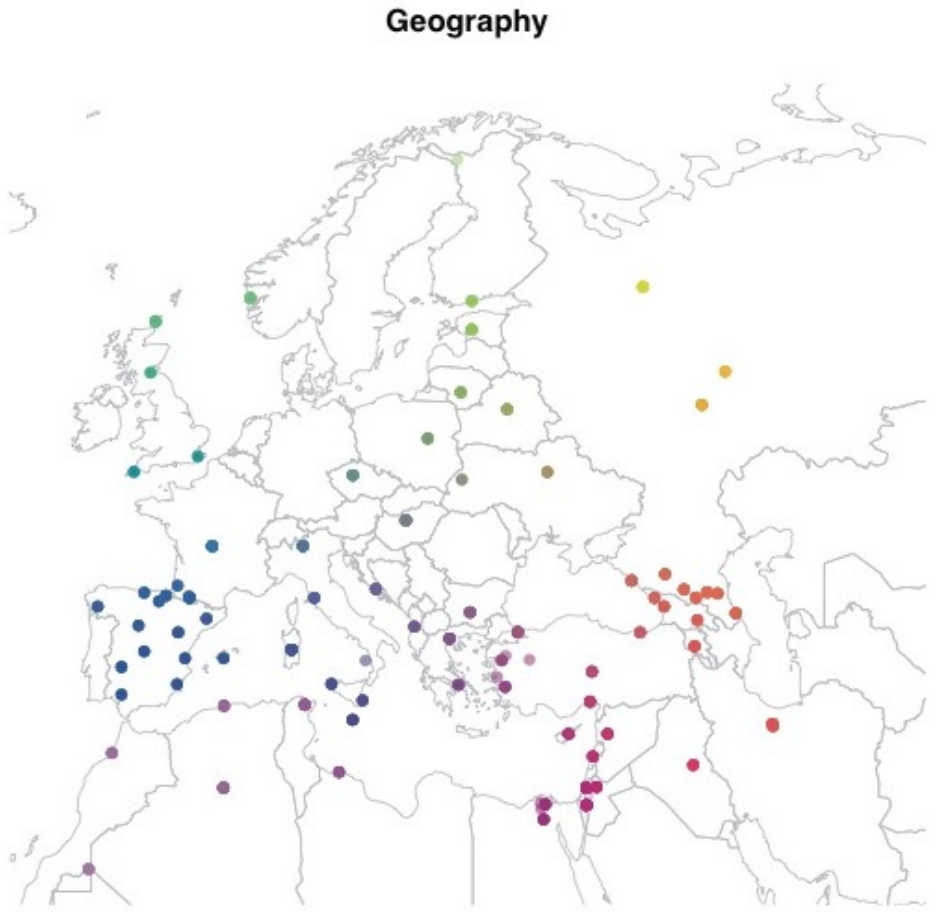
- Mapping data from a space of dimension *p* to a space of dimension *d<<p*



Dimensionality reduction: some applications

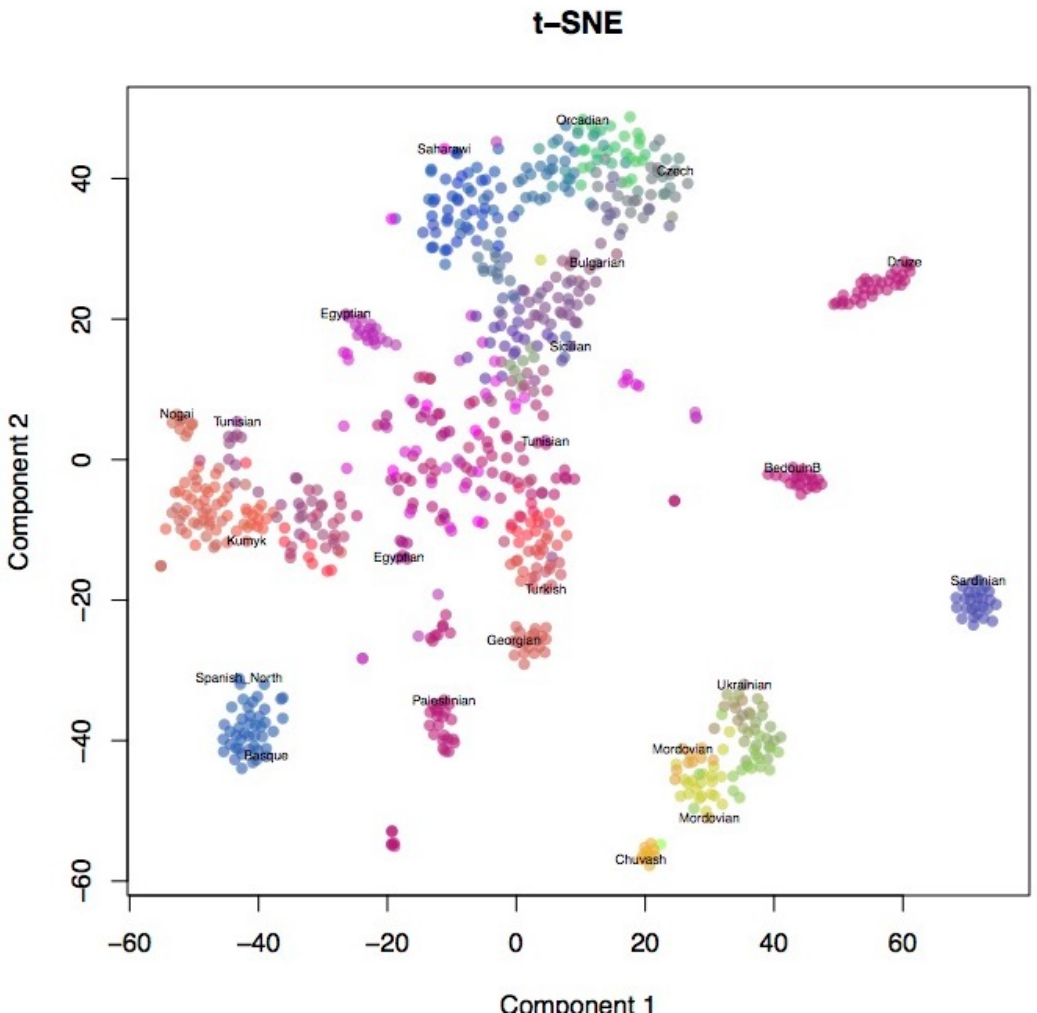
- SNPs (columns) from individual genomes (rows) reflect geographical structure

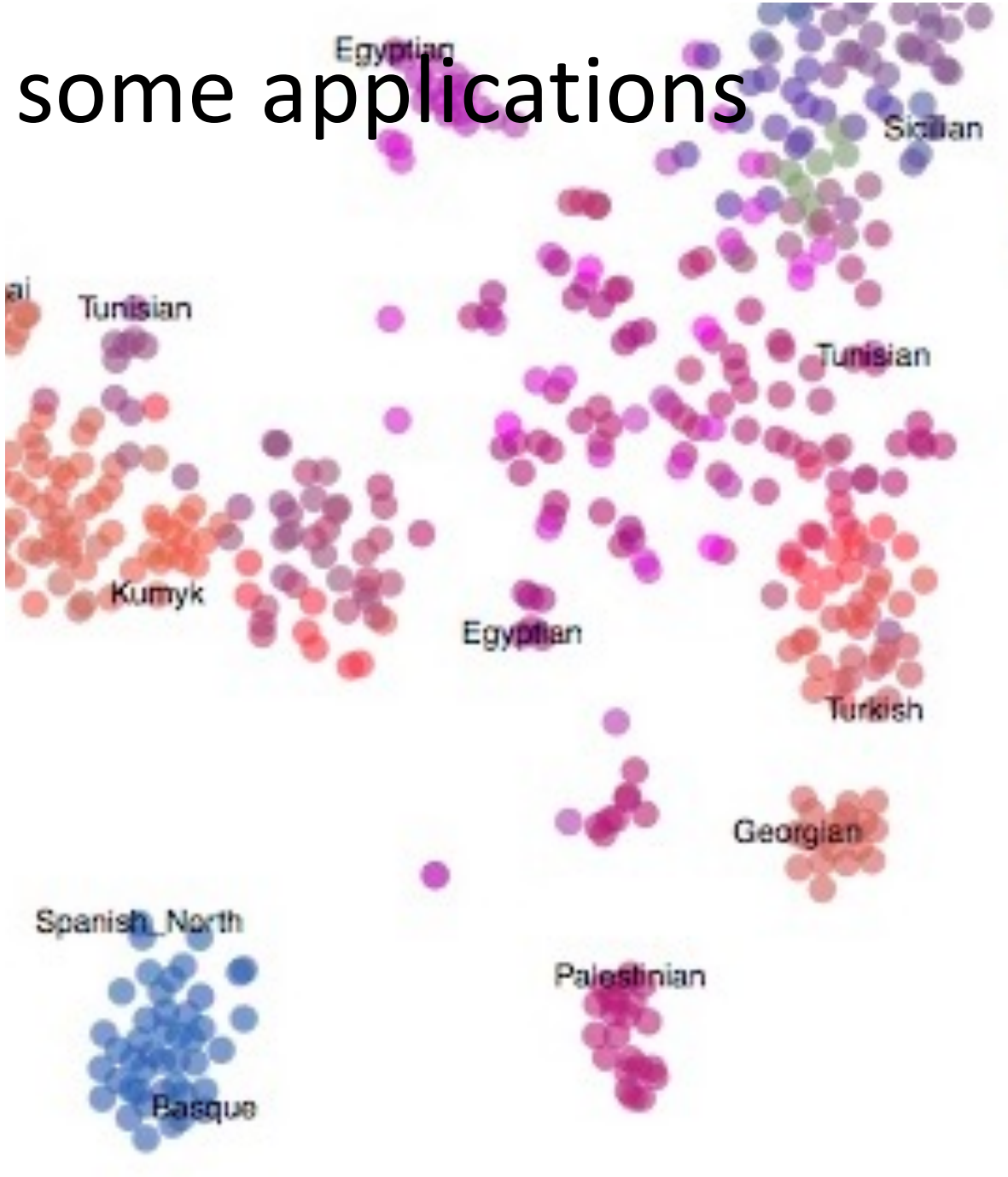




Dimensionality reduction : some applications

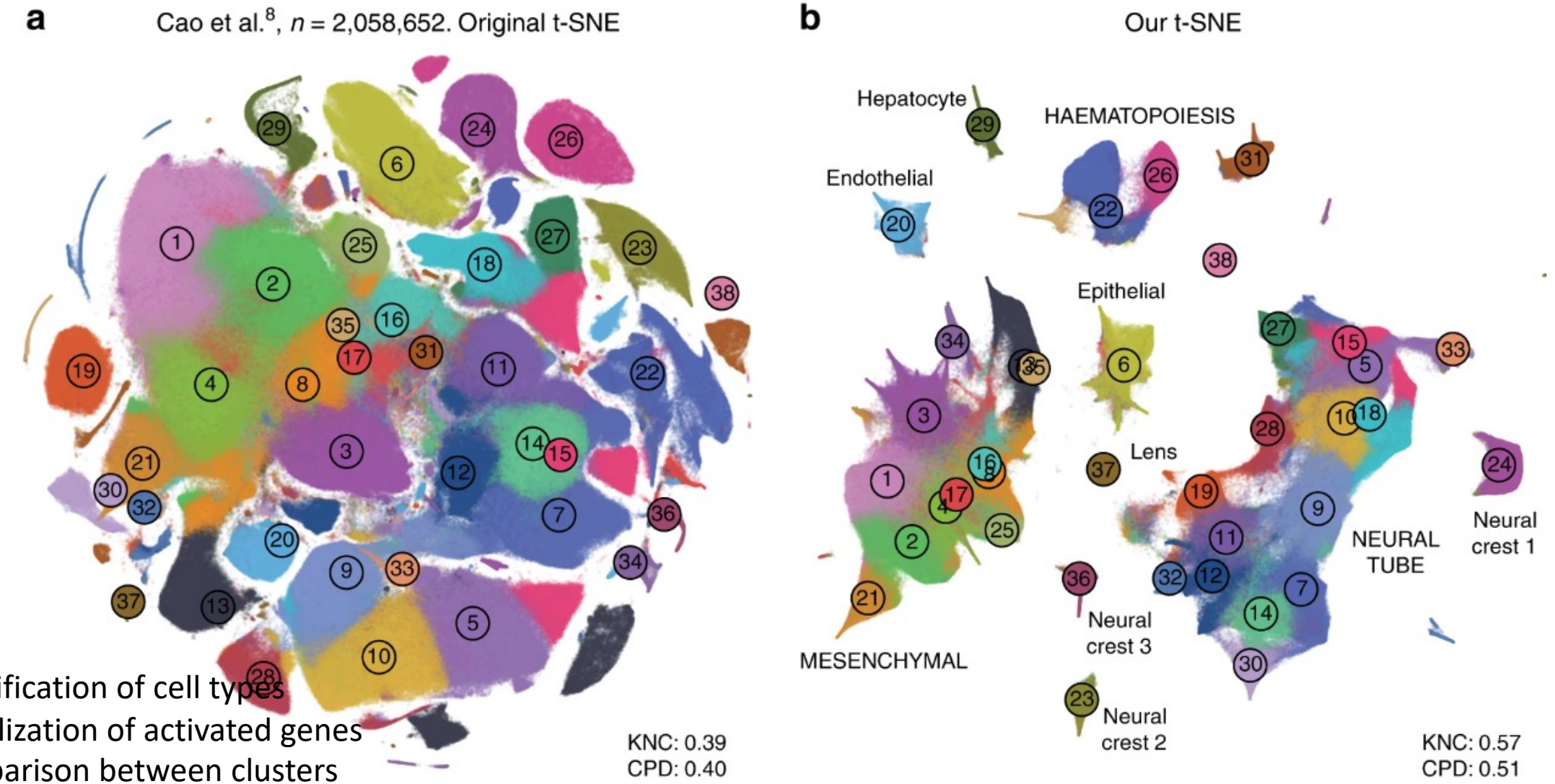
- SNPs (columns) from individual genomes (rows) reflect geographical structure





Dimensionality reduction: some applications

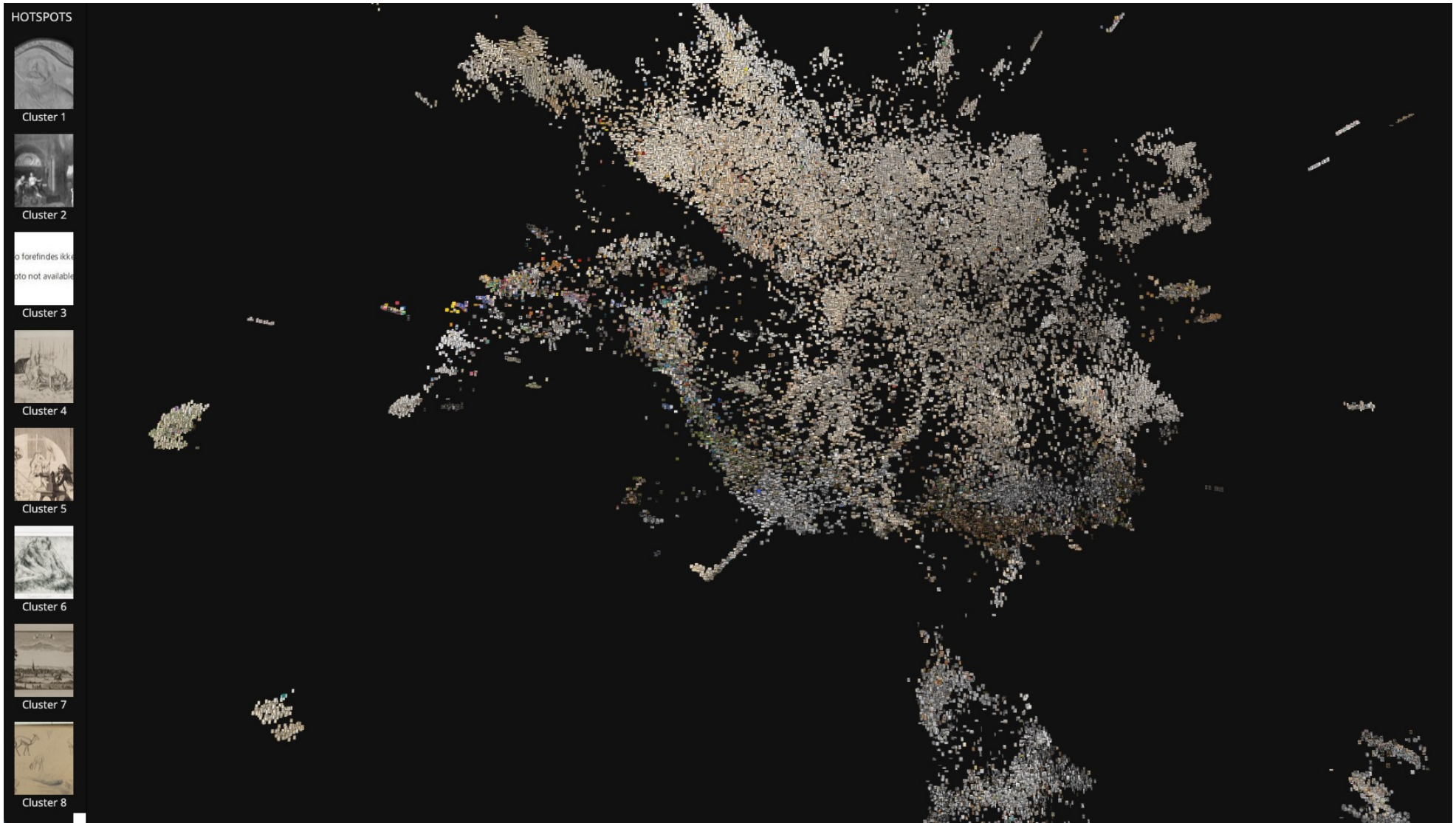
- Transcriptomics: 1.3 Million cells (rows) and ~30_000 genes (columns)



- Identification of cell types
- Visualization of activated genes
- Comparison between clusters

Dimensionality reduction

- Art: categorizing and exploring large quantities of paintings



Dimensionality reduction: methods types

	Linear (Parametric)	Non-linear	Non parametric
Unsupervised	PCA	kPCA ISOMAP	MDS tSNE UMAP
Supervised	LDA	kLDA	

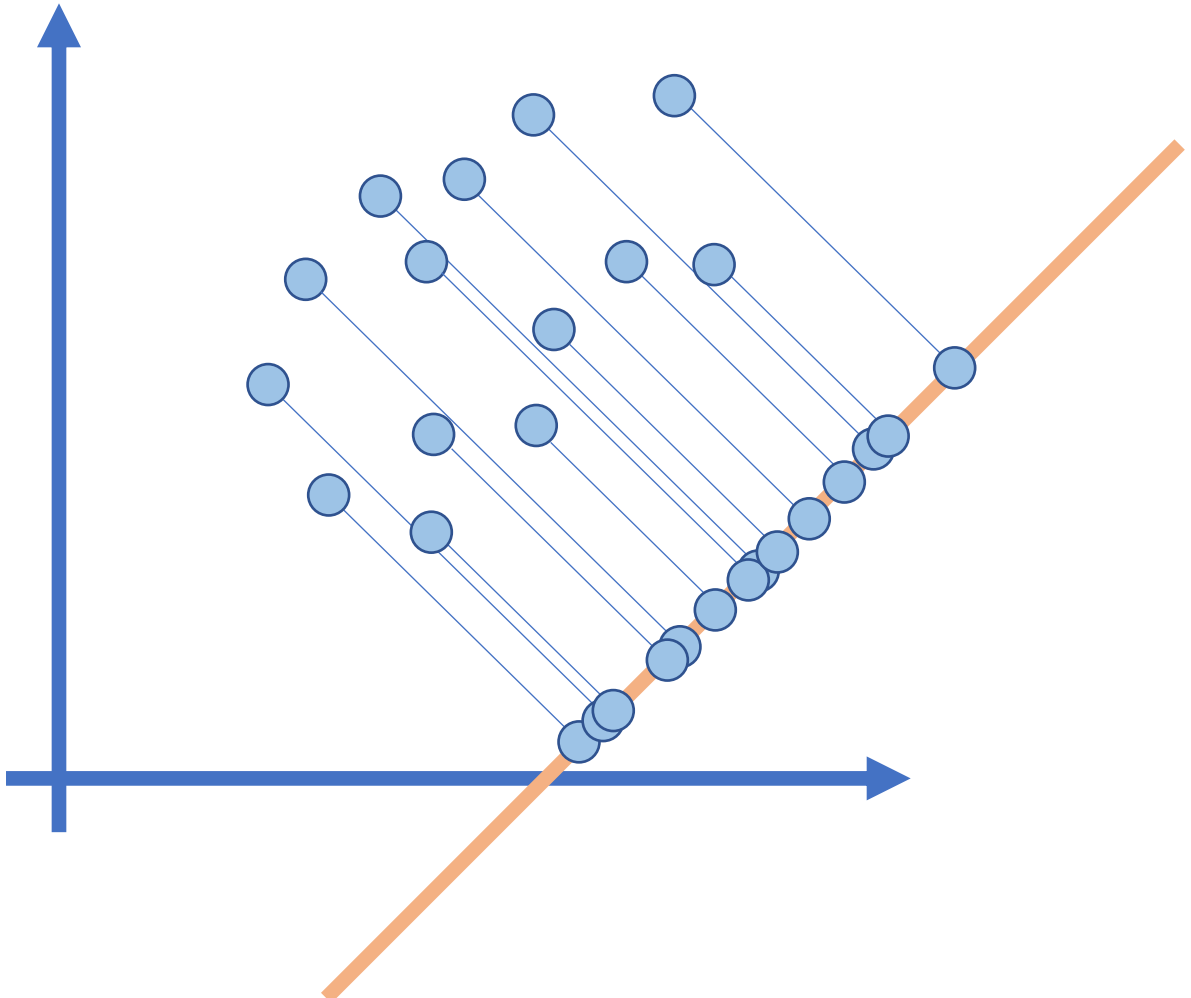
Dimensionality reduction: methods types

	Linear (Parametric)	Non-linear	Non parametric
Unsupervised	PCA UMAP (only parametric)	kPCA ISOMAP	MDS tSNE UMAP
Supervised	LDA	kLDA	UMAP

Dimensionality reduction: PCA

	L	Non-L
Unsup	**PCA**	kPCA
Sup	LDA	kLDA

$p=2 \rightarrow d=1$



Based on

$XX^t = U\Sigma^2U^t$

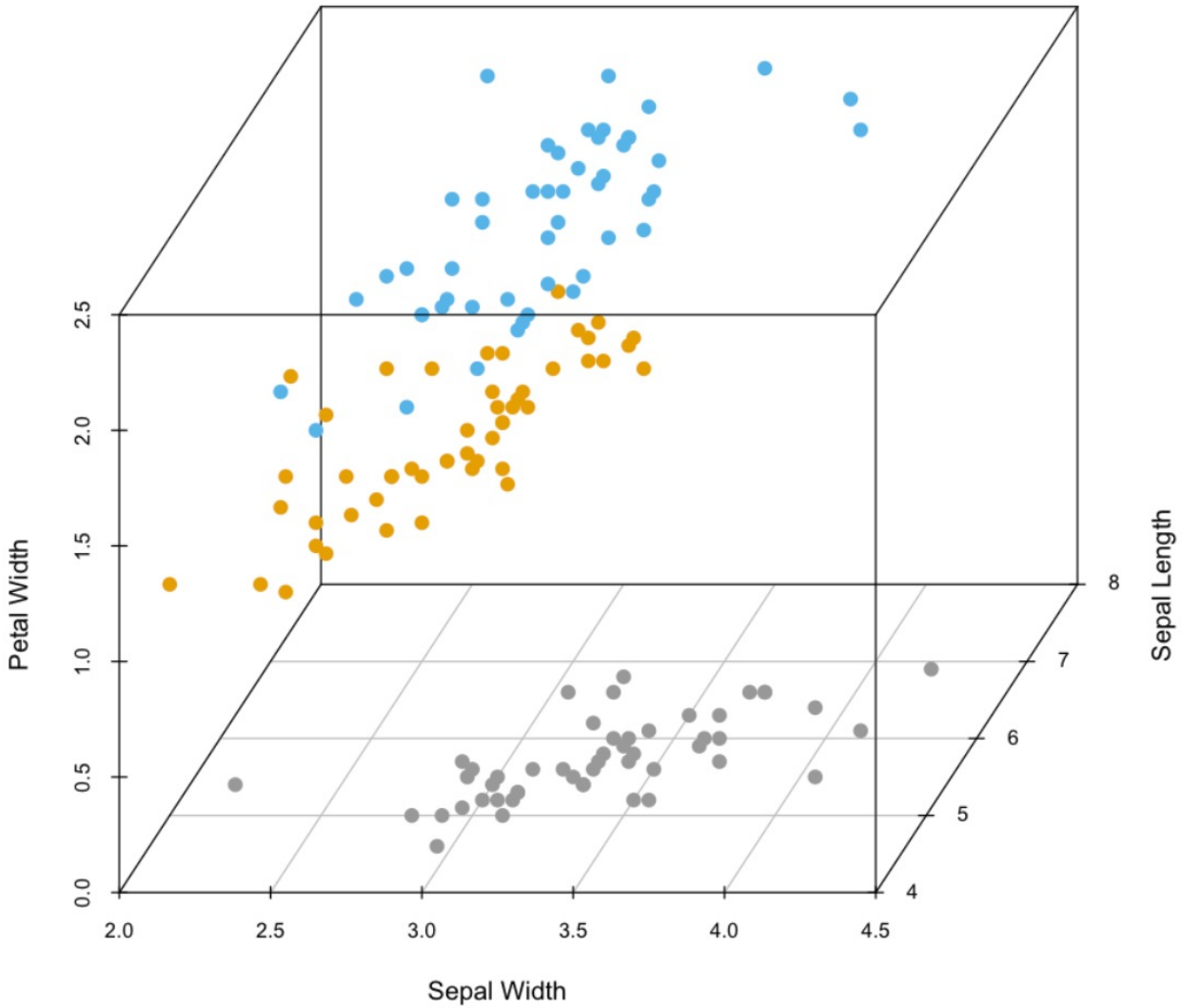
$Y = XU = U\Sigma$

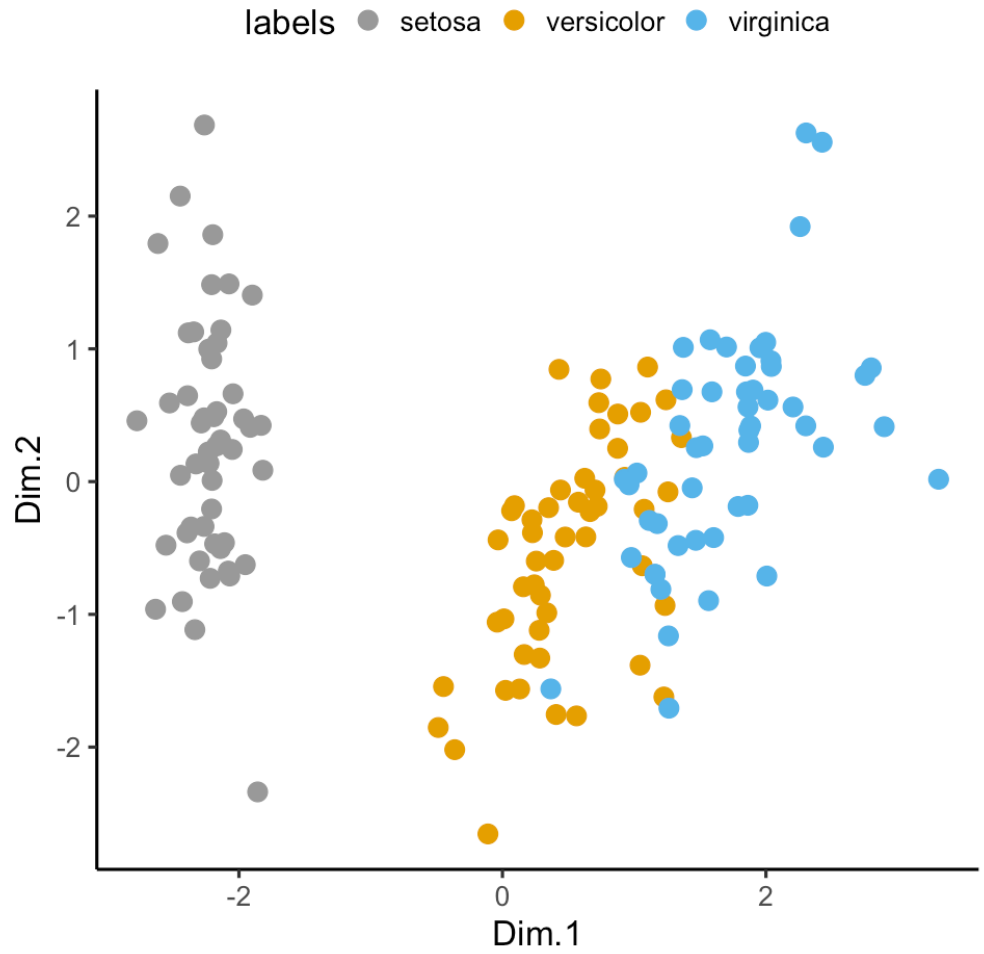
minimizing reconstruction error

$\|U\Sigma - U_{red}\Sigma_{red}\|_2^2$ or $\|X - X_{red}\|_2^2$

Dimensionality reduction: limits of linearity

	L	Non-L
Unsup	**PCA**	kPCA
Sup	LDA	kLDA

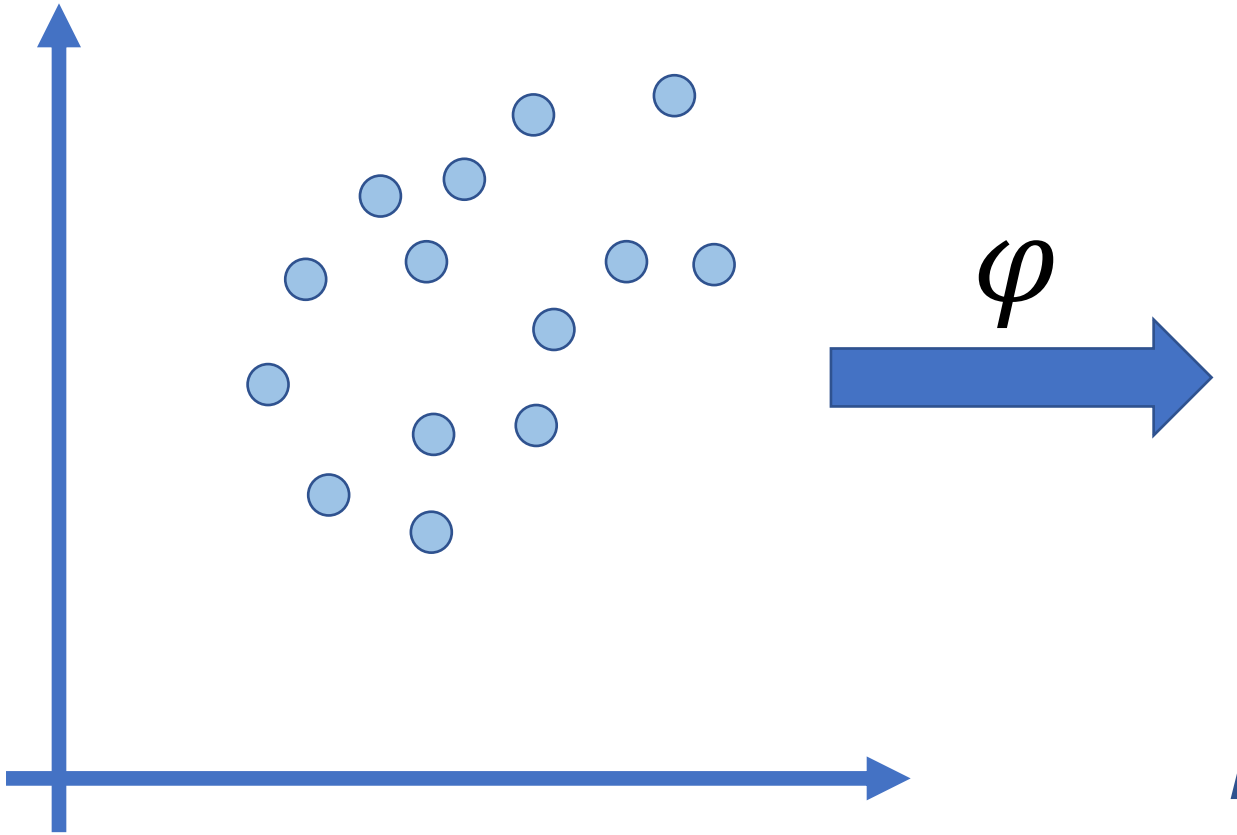




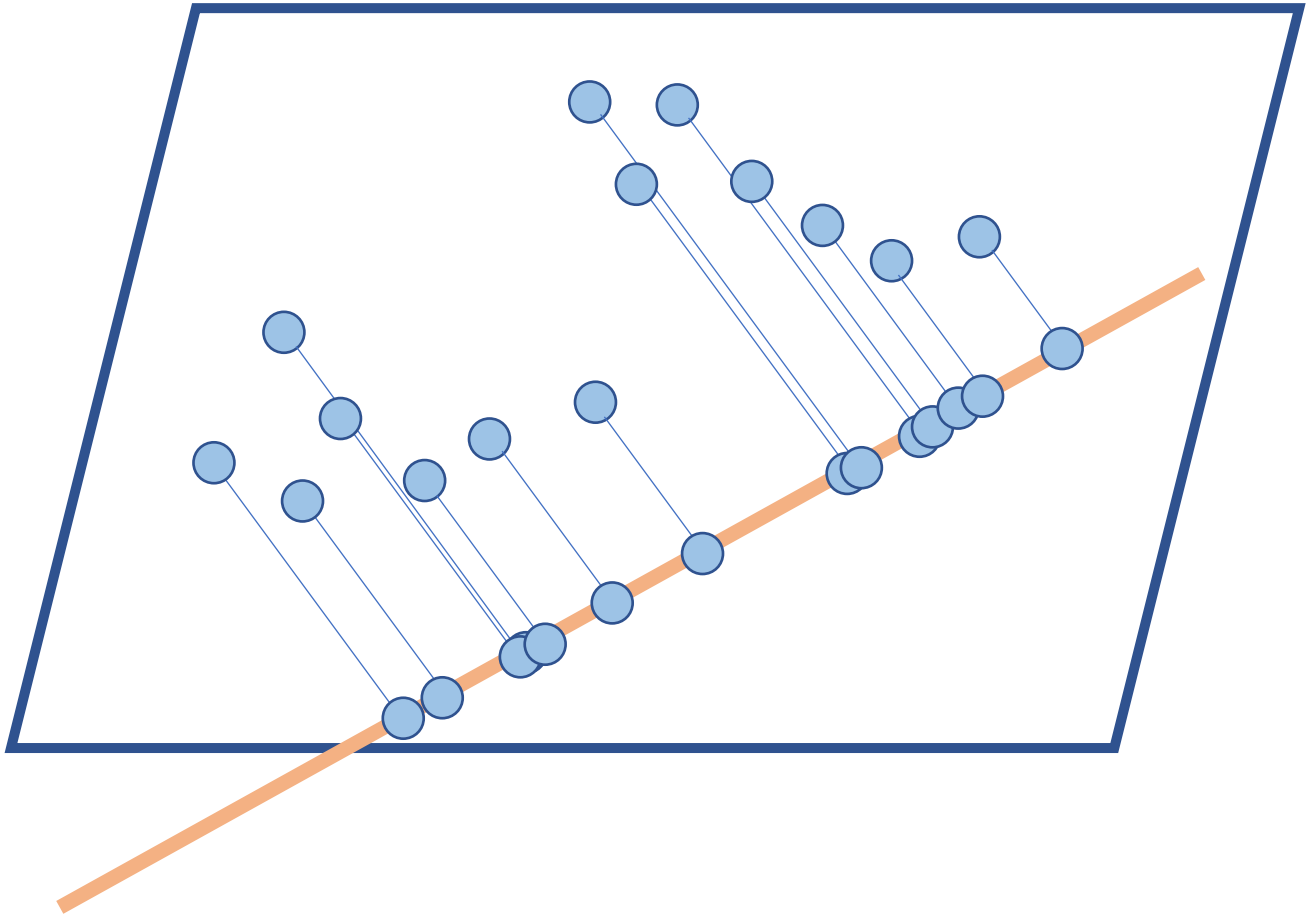
Dimensionality reduction: kernelPCA

	L	Non-L
Unsup	PCA	**kPCA**
Sup	LDA	kLDA

$p=2 \rightarrow p' \rightarrow d=1$



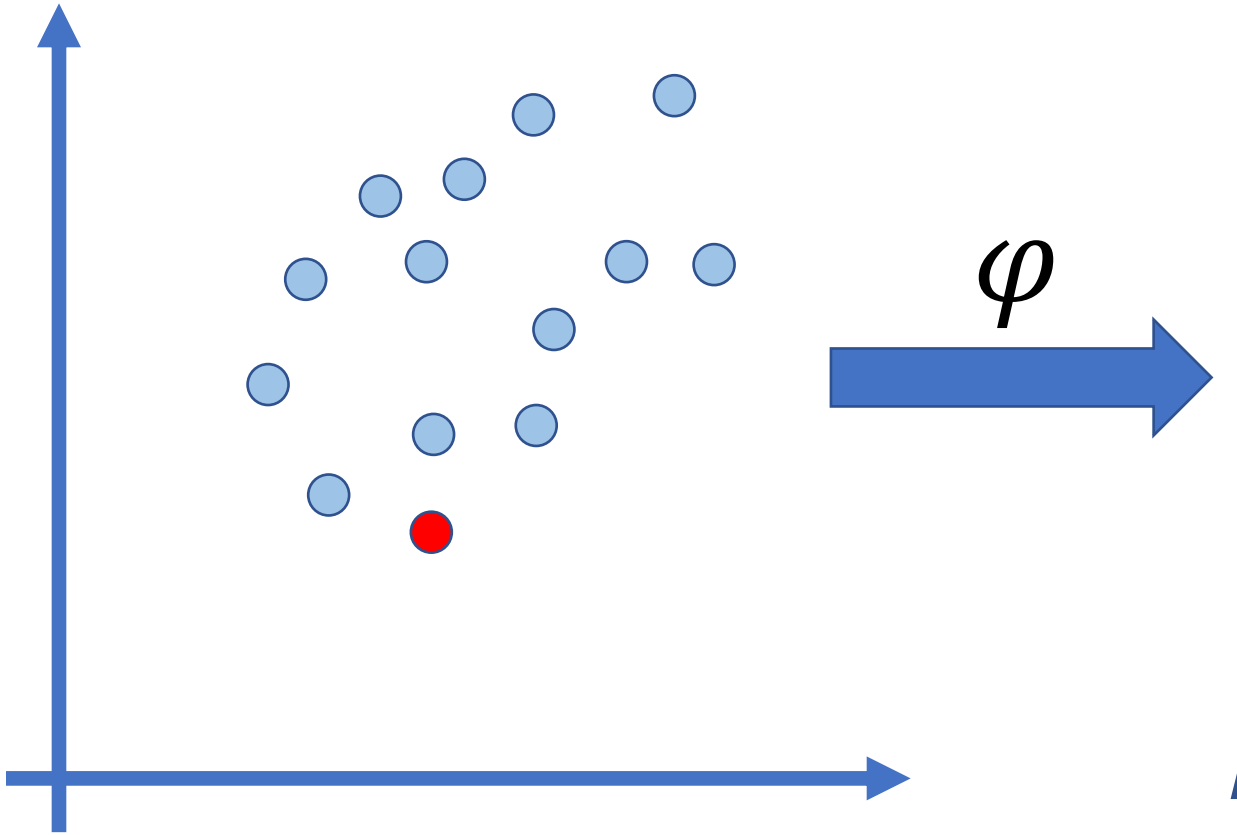
φ



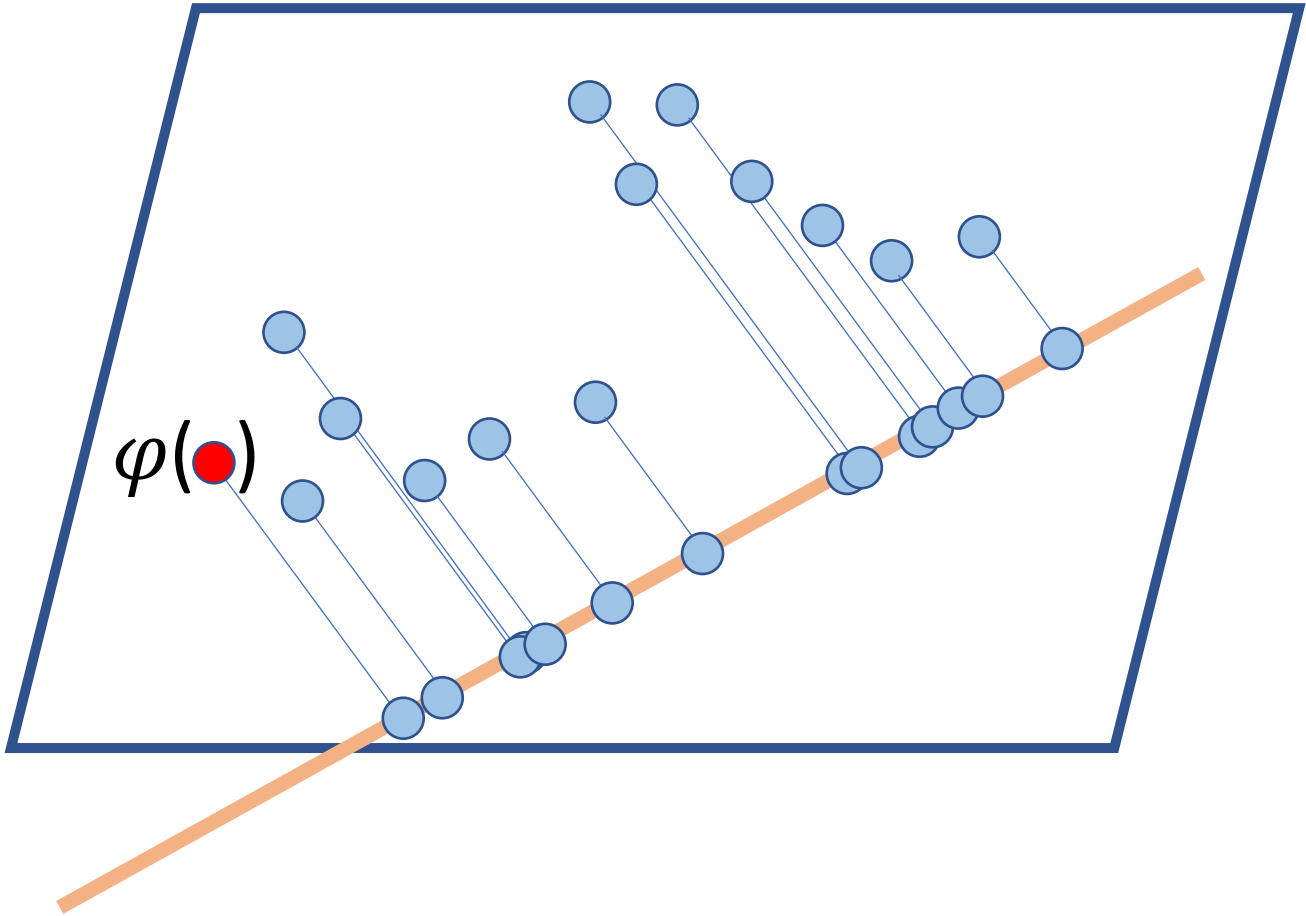
Dimensionality reduction: kernelPCA

	L	Non-L
Unsup	PCA	**kPCA**
Sup	LDA	kLDA

$p=2 \rightarrow p' \rightarrow d=1$

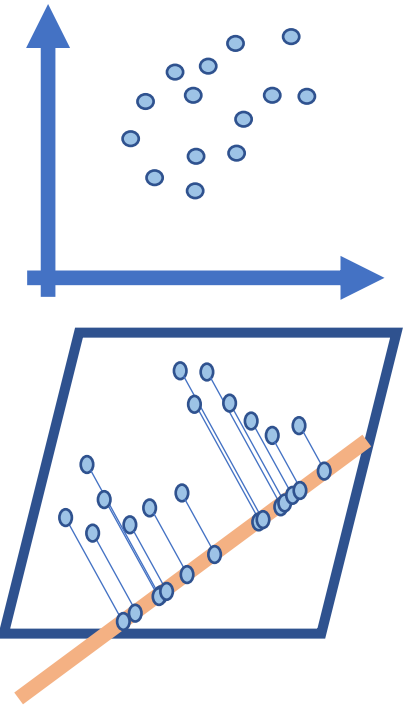


φ



Dimensionality reduction: kernelPCA

	L	Non-L
Unsup	PCA	**kPCA**
Sup	LDA	kLDA



Analogous to PCA

$$XX^t \quad \longleftrightarrow \quad \varphi(X)\varphi(X)^t = [\varphi(x_i)\varphi(x_j)^t\,]_{ij}$$

$$Y = U\boldsymbol{\Sigma} \quad \longleftrightarrow \quad \mathrm{Y} = \mathrm{Q}\,\Lambda^{1/2}$$

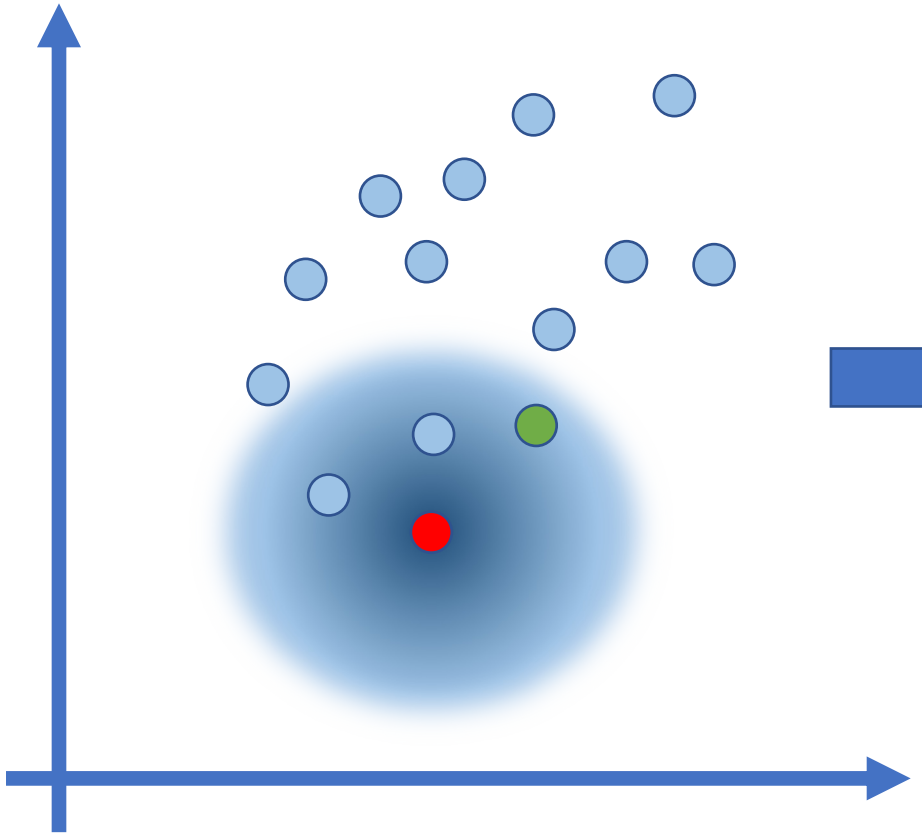
by assuming centered kernel φ and a centered kernel matrix

$$\tilde{\mathrm{M}}_{ij} = \varphi(x_i)\varphi(x_j)^{\mathrm{t}} = \mathrm{Q}\Lambda\mathrm{Q}^{\mathrm{t}}$$

Dimensionality re duction: kernel trick

	L	Non-L
Unsup	PCA	**kPCA**
Sup	LDA	kLDA

$p=2 \rightarrow p' \rightarrow d=1$



φ

$$[M]_{ij} = [\ \varphi(x_i, x_j)\ ]$$

Avoids dot-products

Gaussian/RBF $\varphi(x_i, x_j) = exp(-\|x_i - x_j\|^2 / 2\sigma^2)$ $\sigma \geqslant 0$

Polynomial $\varphi(x_i, x_j) = (x_i^T \cdot x_j + c)^d$ $c \geqslant 0, d \geqslant 1$

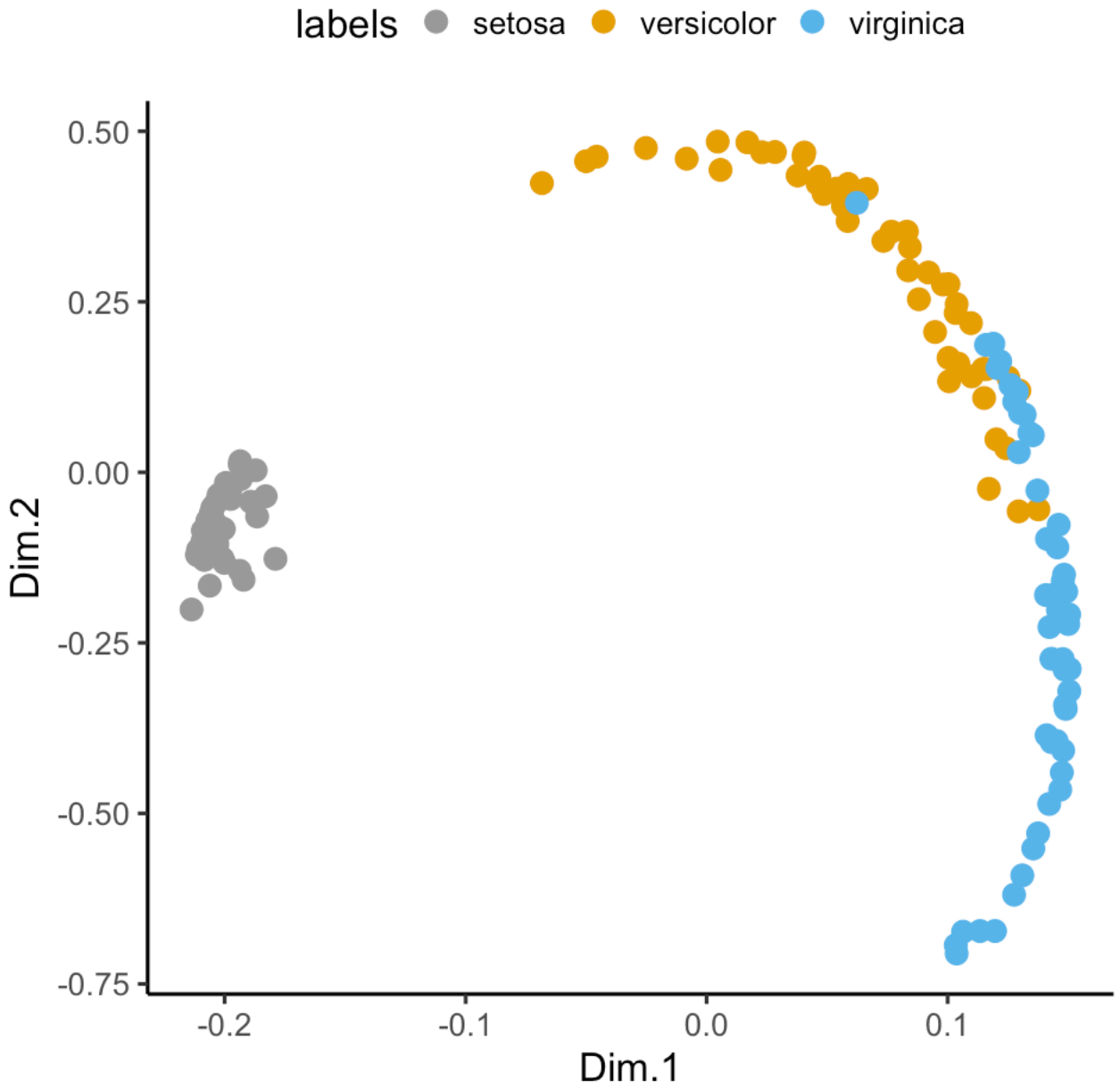
Linear $\varphi(x_i, x_j) = x_i^T \cdot x_j$

Laplacian $\varphi(x_i, x_j) = exp(-\alpha\|x_i - x_j\|)$ $\alpha \geqslant 0$

……..

Dimensionality reduction: kernelPCA

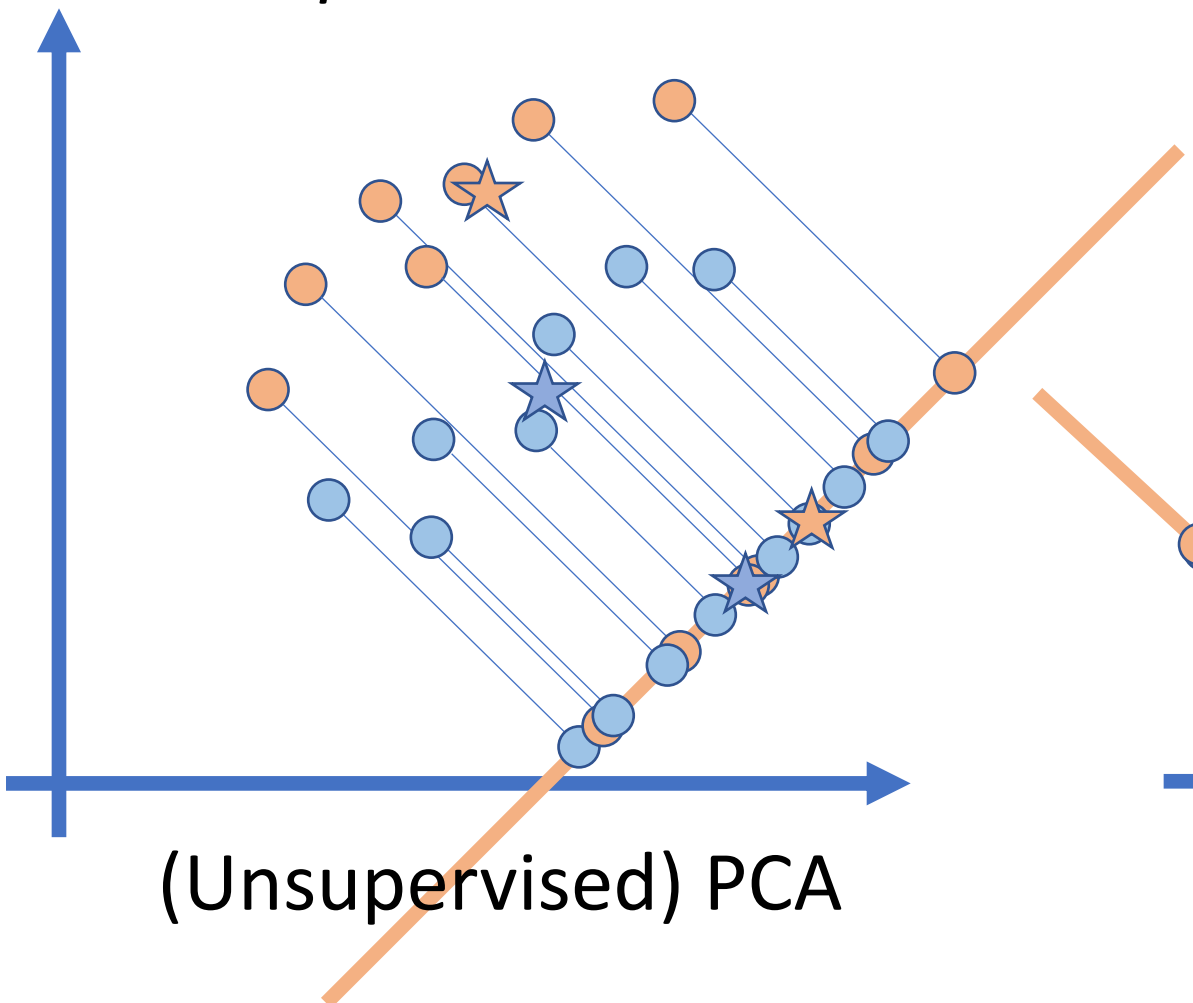
	L	Non-L
Unsup	PCA	**kPCA**
Sup	LDA	kLDA



Dimensionality reduction: LDA

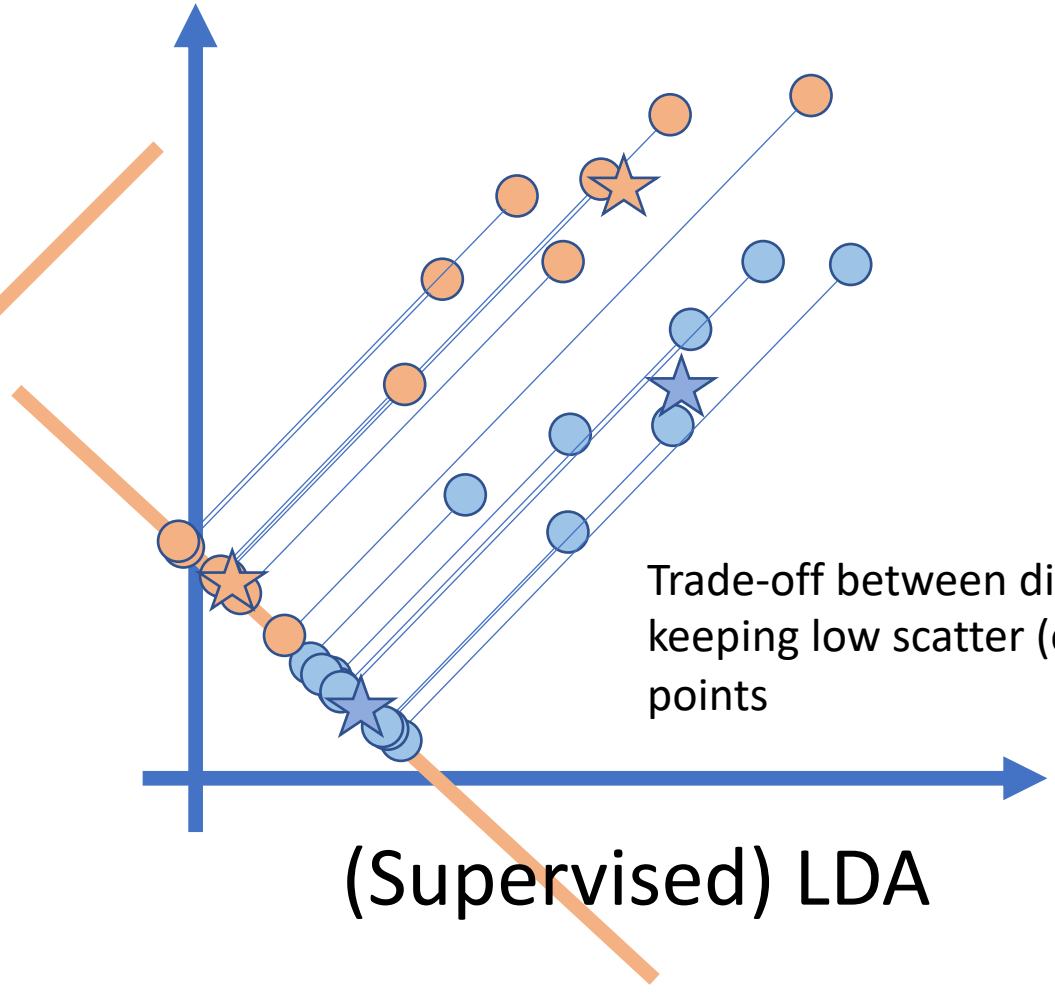
	L	Non-L
Unsup	PCA	kPCA
Sup	**LDA**	kLDA

p=2 → d=1



(Unsupervised) PCA

p=2 → d=1

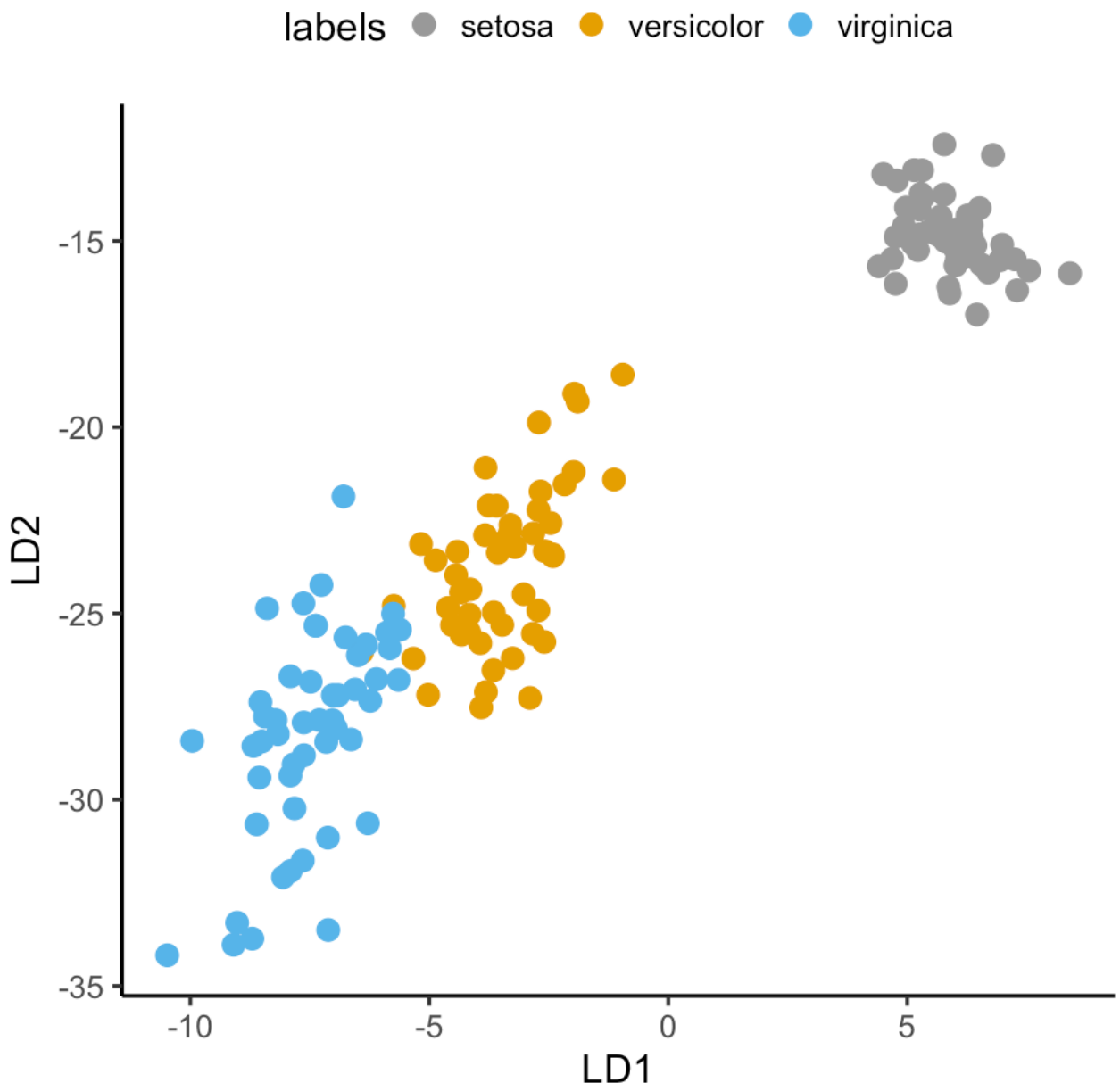


(Supervised) LDA

Trade-off between distancing means and keeping low scatter (covariance) between points

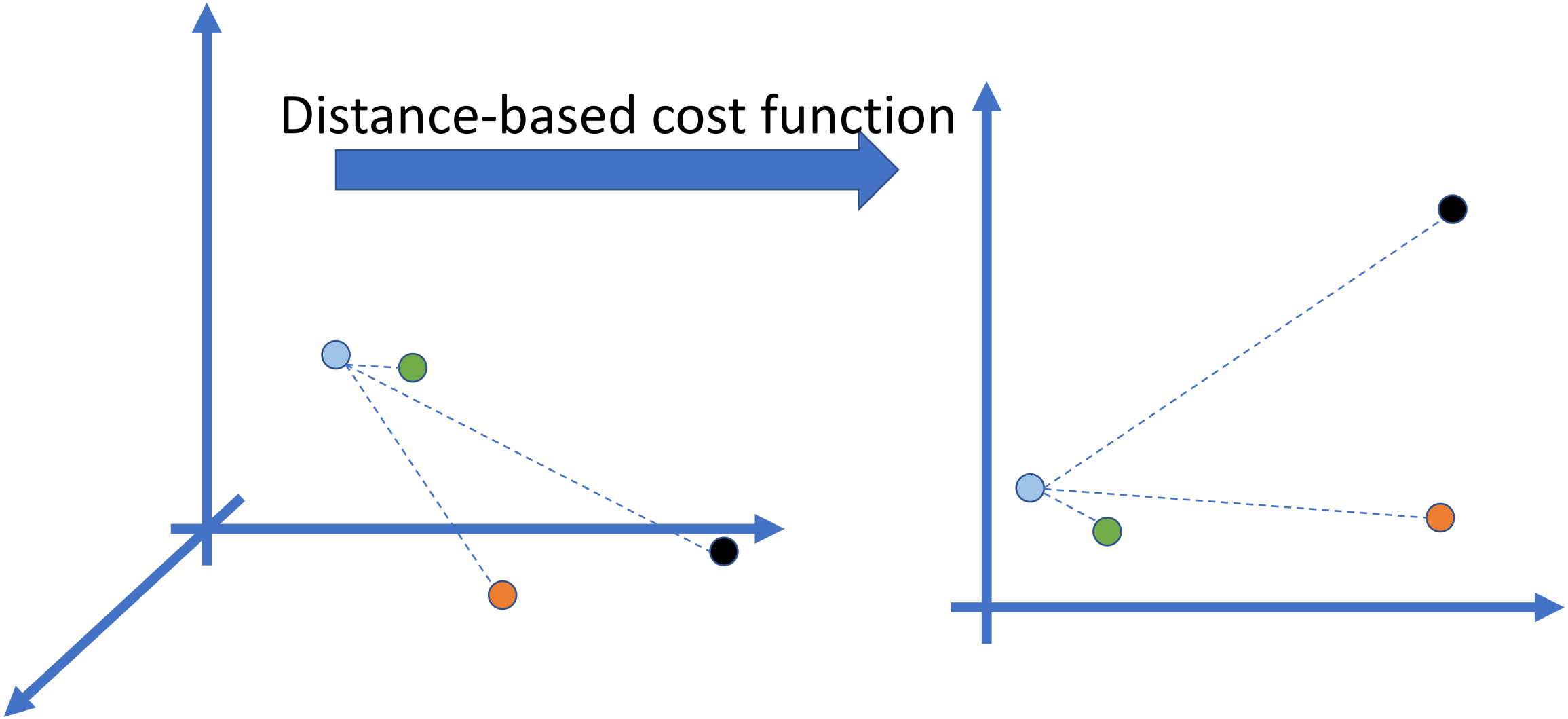
Dimensionality reduction: LDA

	L	Non-L
Unsup	PCA	kPCA
Sup	**LDA**	kLDA



Dimensionality reduction:

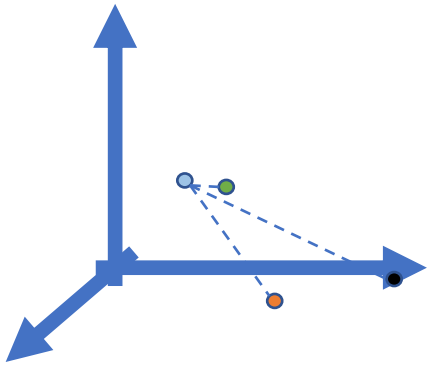
Non-parametric methods → MDS

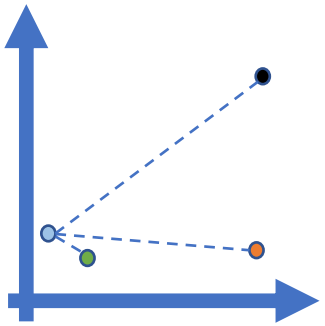


Dimensionality reduction:

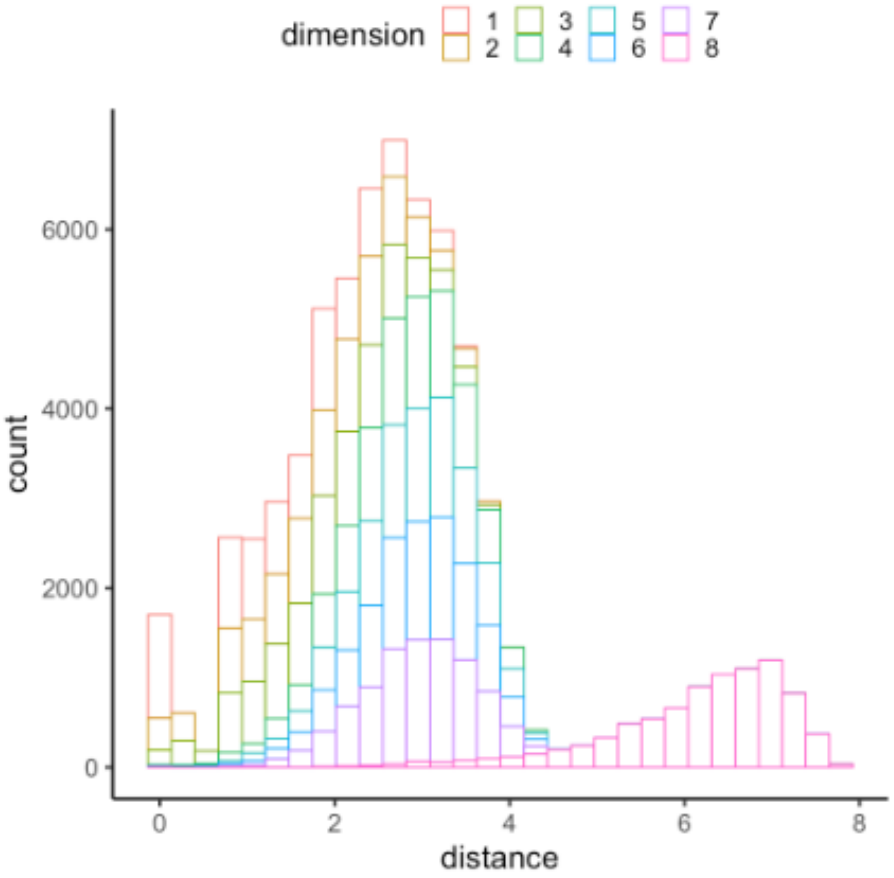
Non-parametric methods → MDS

	L	Non-L
Unsup	PCA	kPCA
Sup	**LDA**	kLDA





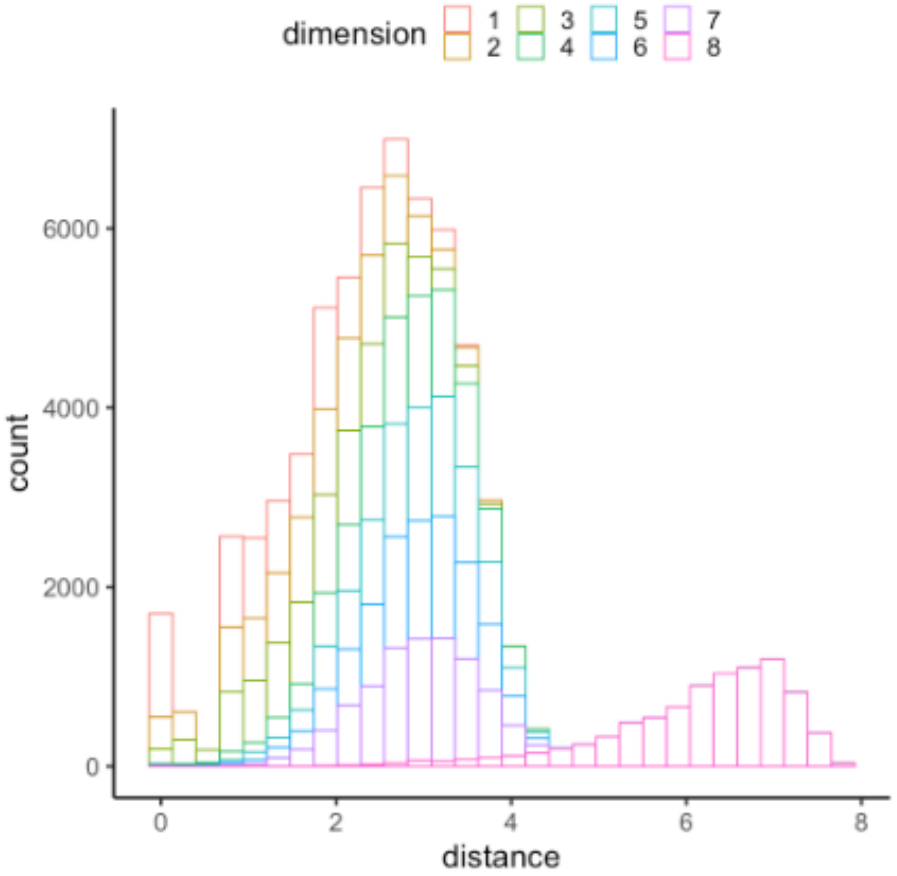
- The cost function
  $C = \boldsymbol{\Sigma}_{i<j} [x_{ij} - \| y_i - y_j \|]^2$
  is optimized (wrt projections y) to preserve distances
- *Cost quadratic in nr. of data points*
- There are LARGER DISTANCES in high dimensions!



Dimensionality reduction:

Distances: curse of dimensionality

	L	Non-L
Unsup	PCA	kPCA
Sup	**LDA**	kLDA



Distances increase in higher dimensions.

Actually,

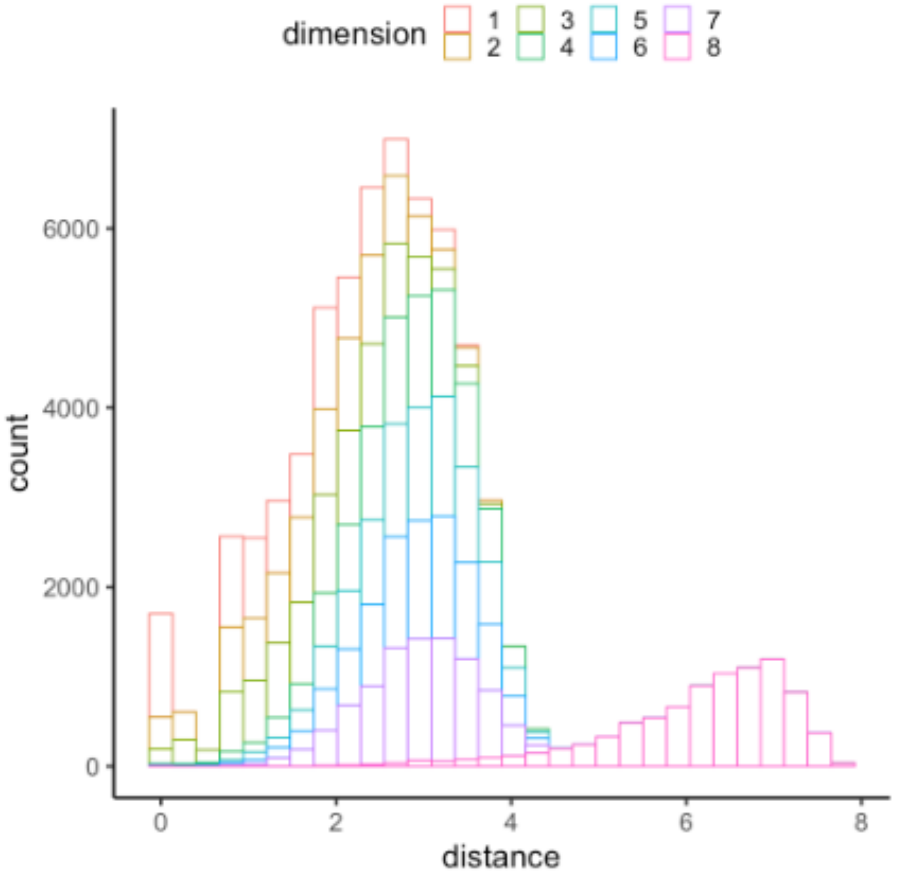
$$\forall \varepsilon > 0 : \lim_{d \to \infty} \mathrm{P}\left[dist_d \left(\frac{\mathrm{Dmax}_d - \mathrm{Dmin}_d}{\mathrm{Dmin}_d}, 0 \right) \leq \varepsilon \right] = 1$$

The ratio of distance between nearest and farthest neighbour against the distance to nearest neighbor **tends to zero** with increasing dimensions

Dimensionality reduction:

Distances: curse of dimensionality

	L	Non-L
Unsup	PCA	kPCA
Sup	**LDA**	kLDA



Distances increase in higher dimensions.

Actually,

$$\forall \varepsilon > 0 : \lim_{d \to \infty} \mathrm{P}\left[dist_d \left(\frac{\mathrm{Dmax}_d - \mathrm{Dmin}_d}{\mathrm{Dmin}_d}, 0 \right) \leq \varepsilon \right] = 1$$

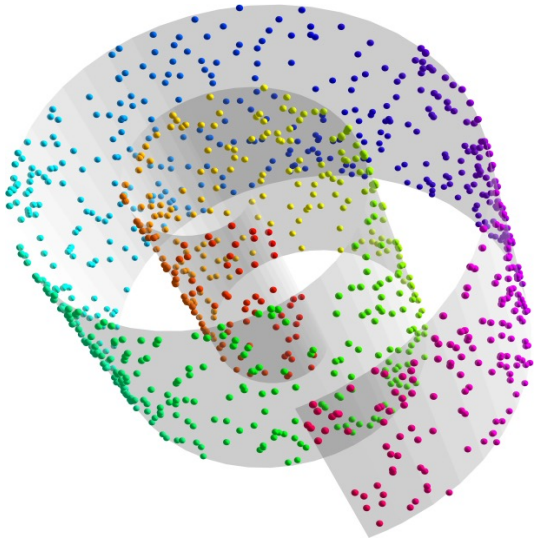
So distances **increase** and we have **no smaller distances** to preserve

Dimensionality reduction:

Distance preserving → Manifold learning

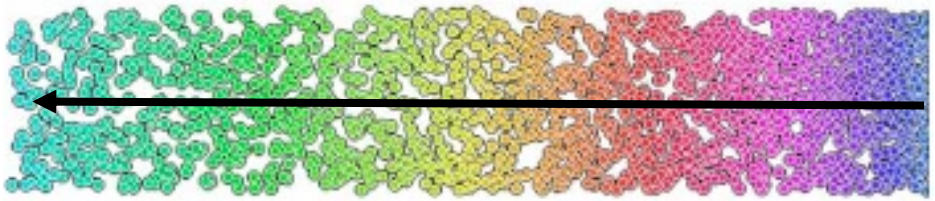
	L	Non-L
Unsup	PCA	kPCA
Sup	**LDA**	kLDA

- Your data points are part of a geometric surface of dimension *d*
- This surface is embedded in more dimensions *p*
- Learn the structure and not the distances



p=3

Embedded in 3D through **θ** (Rotation angle)



d=2

Credit: mkbergman.com and bioinformatics-training.github.io

Dimensionality reduction:

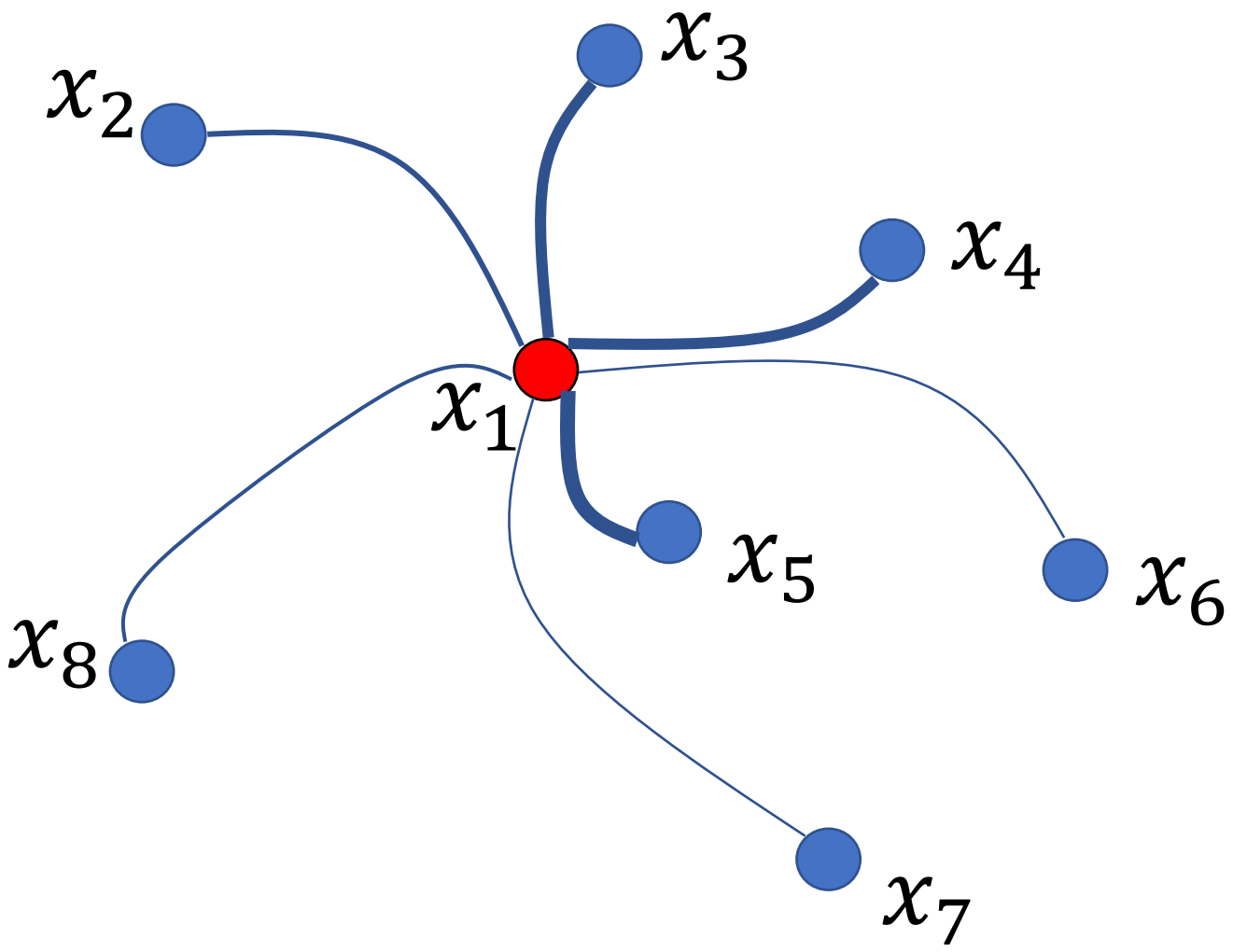
Distance preserving → Manifold learning

	L	Non-L
Unsup	PCA	kPCA
Sup	**LDA**	kLDA

- All the iris flowers in the world (with the four measures as in the *iris* dataset**) will not cover all $\mathbb{R}^4$**
- Most points of $\mathbb{R}^4$ are actually **noise** and not flowers
- This set of points is part in a **subspace** of dimension d of $\mathbb{R}^4$, *d<4*
- The **difference of two points** will likely not give an existing flowers' data (non-linear surface)
- The surface has good **geometric properties** (locally flat, differentiable, …) making it a Riemannian manifold

tSNE: Manifold description by graph

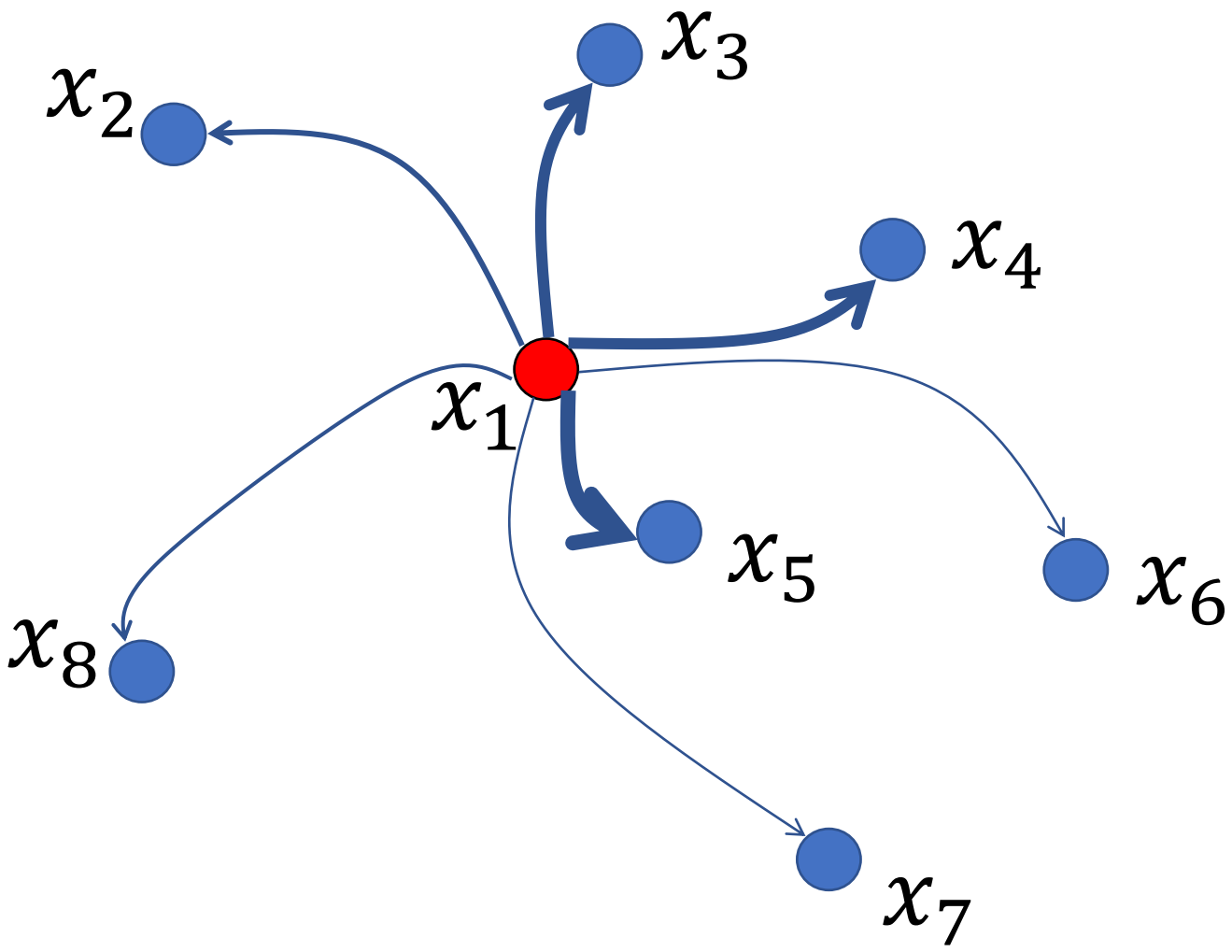
- Datapoints as **vertices** of a graph
- Pairwise **edges** are weighted by a distance function (local flatness)
- Graph algorithms are **well-known and implemented**



Example: graph-exploration computational packages make it fast to find the closest k elements of a data point x_1 (the **k-nearest-neighbors** of a point). In figure: k=7.

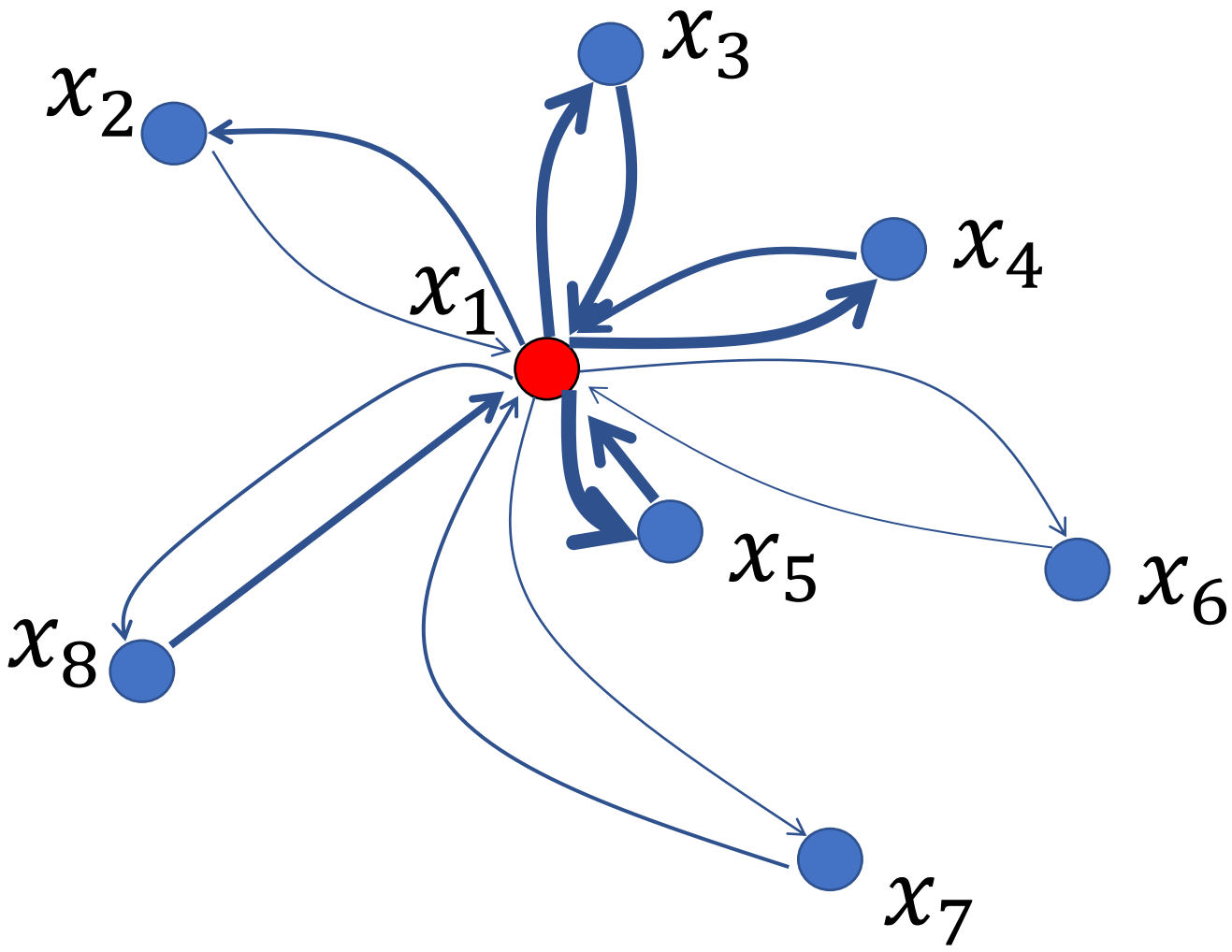
tSNE: Manifold description by graph

- Build outgoing edges weighted by distance in the k-nearest neighborhood of each data point
- Normalize them so they sum to one



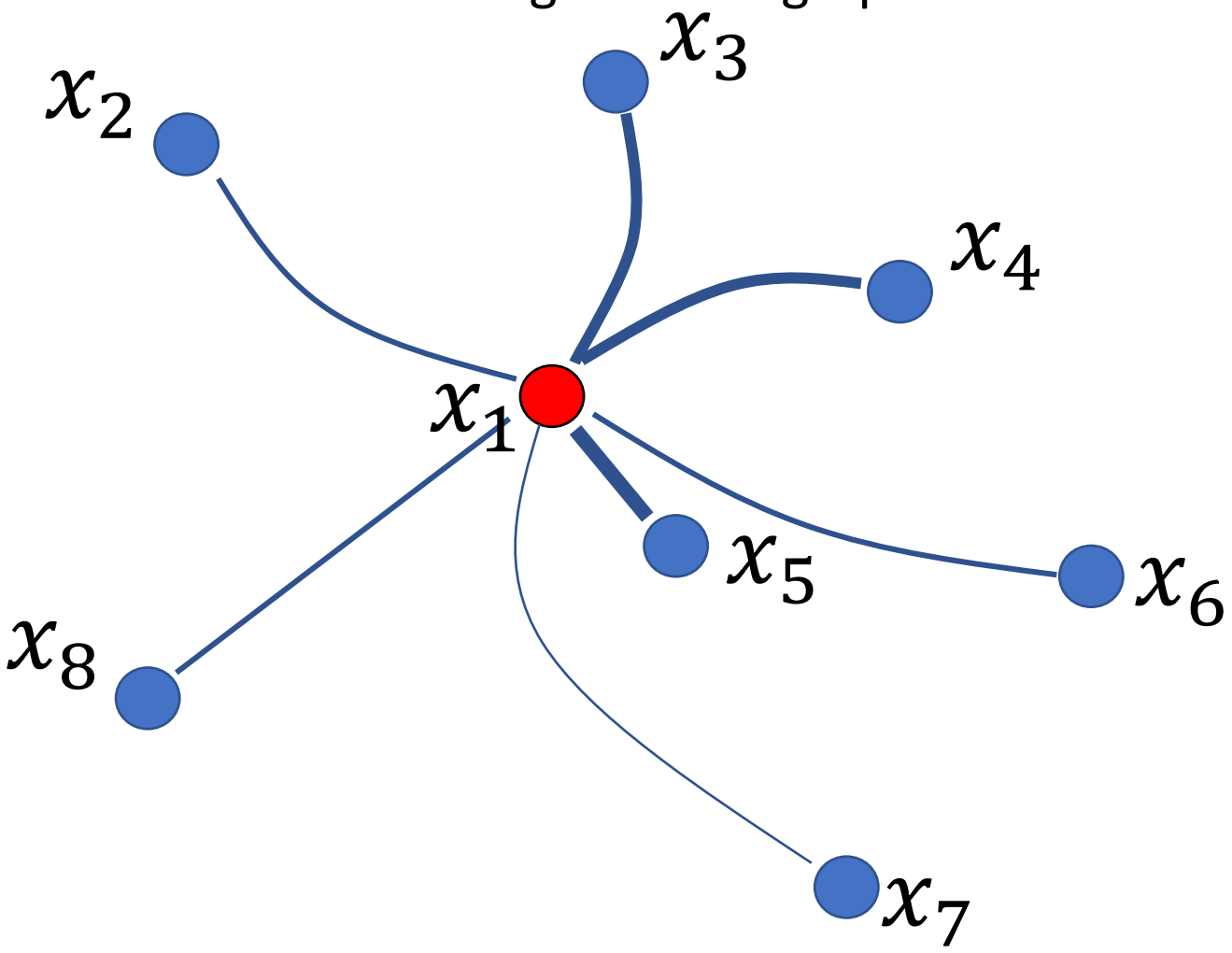
tSNE: Manifold description by graph

- Build outgoing edges weighted by distance in the k-nearest neighborhood of each data point
- Normalize them so they sum to one



tSNE: Manifold description by graph

- Average edges weight between each pair of nodes
- normalize ALL edges of the graph so that their sum is 1



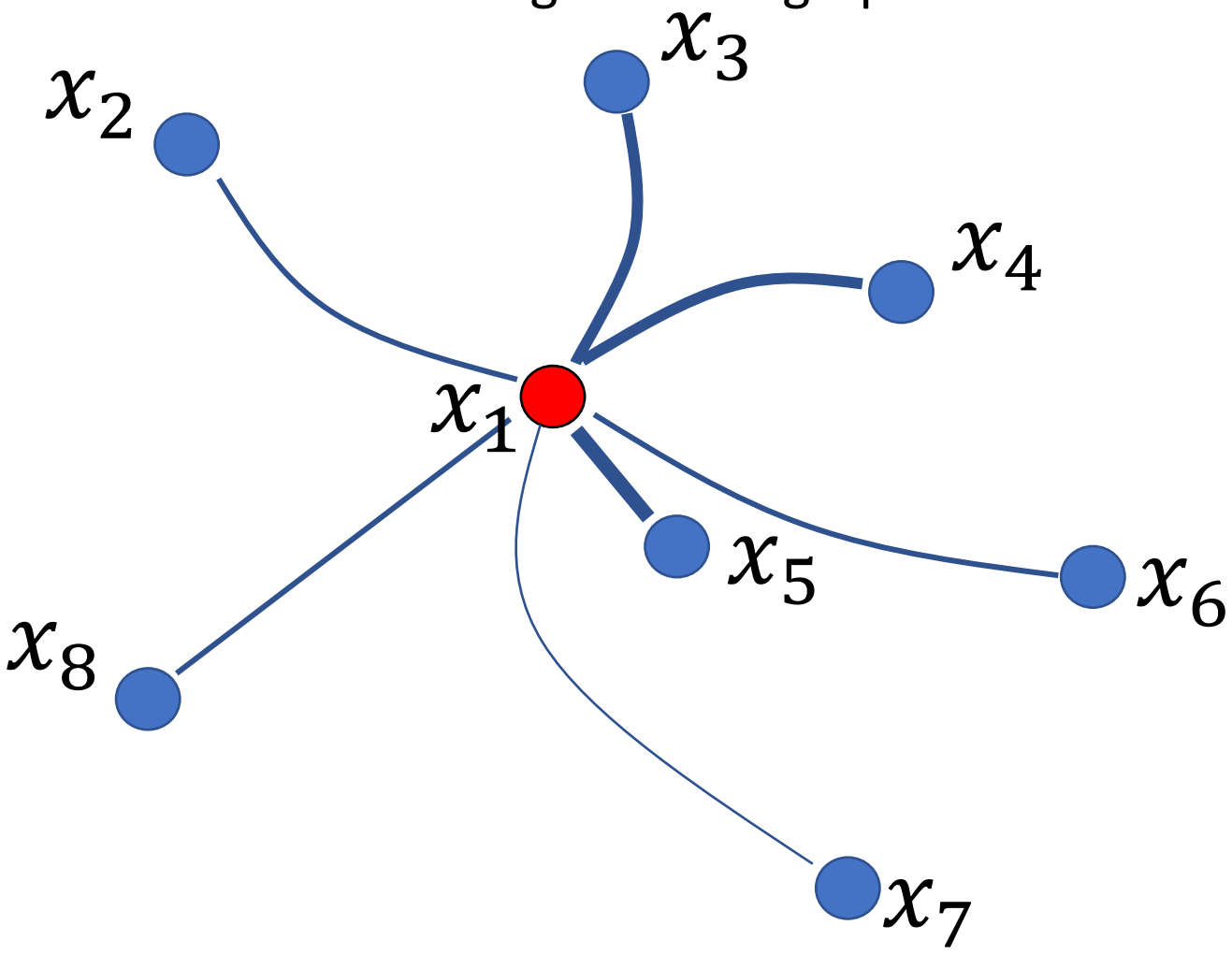
It corresponds to the formula

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2\,n}$$

A procedure of the same type is done in the low-dimension (d << D) space

tSNE: Manifold description by graph

- Average edges weight between each pair of nodes
- normalize ALL edges of the graph so that their sum is 1



It corresponds to the formula

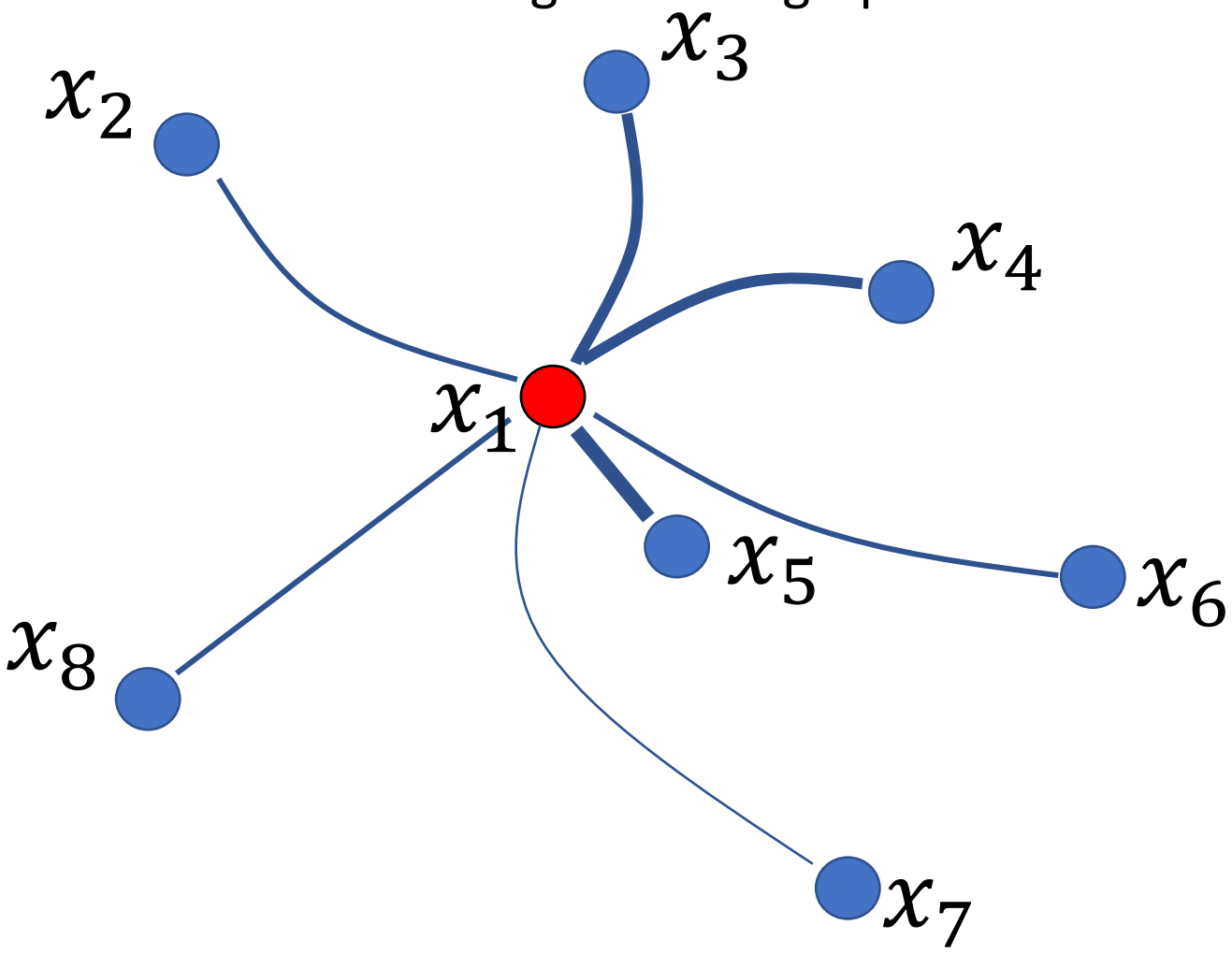
Mean of two normalized edges Between i and j

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

A procedure of the same type is done in the low-dimension (d << p) space

tSNE: Manifold description by graph

- Average edges weight between each pair of nodes
- normalize ALL edges of the graph so that their sum is 1



It corresponds to the formula

Mean of two normalized edges Between i and j

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

Mean of all edges

A procedure of the same type is done in the low-dimension (d << p) space

tSNE: high-dim graph implementation

In the book's code (p.280) ALL Distances are calculated and evaluated in a gaussian kernel for each point *i*

```
D <- as.matrix(dist(X))^2
Pji <- exp(-D[i, -i] / (2 * svals[i]))
```

P_{ji}'s are normalized and assigned to the matrix P, so we have the ingoing and outgoing edges normalized separately

```
Pji <- Pji / sum(Pji)
P[i, -i] <- Pji
```

The final normalization is then

```
return(0.5 * (P + t(P)) / sum(P))
```

```
function(X, perplexity=15)
{

  D <- as.matrix(dist(X))^2
  P <- matrix(0, nrow(X), nrow(X))
  svals <- rep(1, nrow(X))

  for (i in seq_along(svals))
  {
    srange <- c(0, 100)
    tries <- 0

    for(j in seq_len(50))
    {
      Pji <- exp(-D[i, -i] / (2 * svals[i]))
      Pji <- Pji / sum(Pji)
      H <- -1 * Pji %*% log(Pji, 2)

      if (H < log(perplexity, 2))
      {
        srange[1] <- svals[i]
        svals[i] <- (svals[i] + srange[2]) / 2
      } else {
        srange[2] <- svals[i]
        svals[i] <- (svals[i] + srange[1]) / 2
      }
    }
    P[i, -i] <- Pji
  }

  return(0.5 * (P + t(P)) / sum(P))
}
```

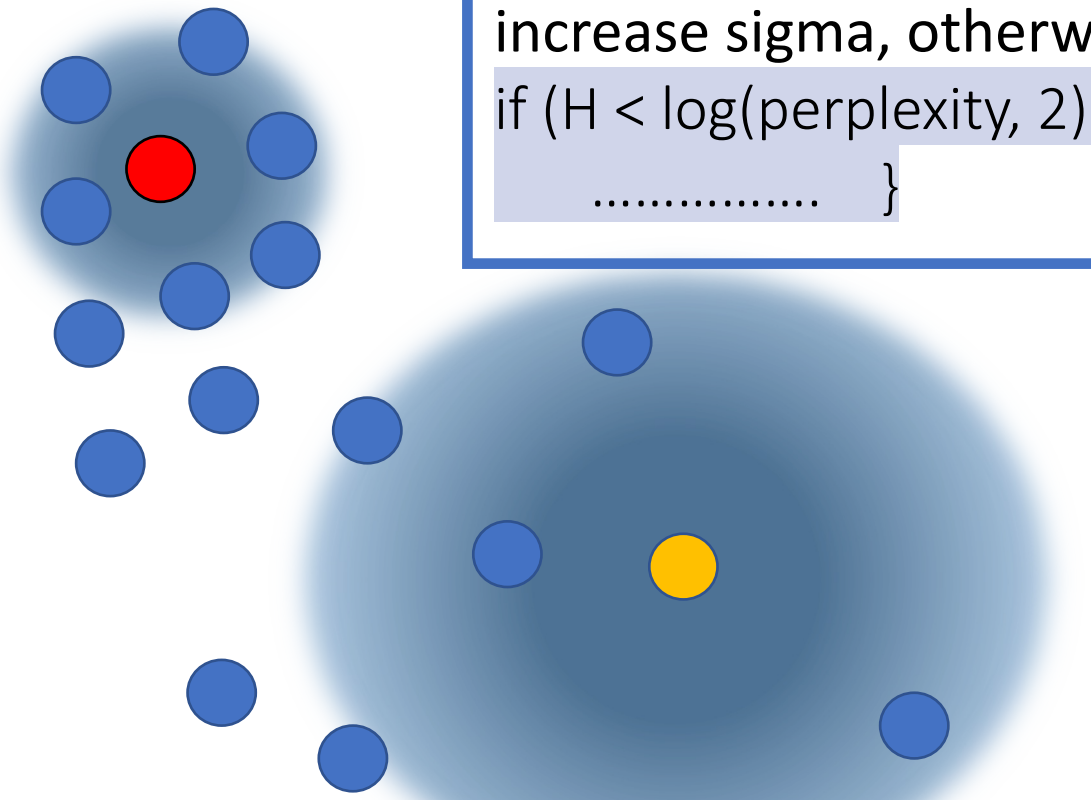
tSNE: high-dim graph implementation – Adaptive variance

Calculate entropy of i-th neighbors

```
H  <-  -1 * Pji %*% log(Pji, 2)
```

If H is less than the log-perplexity, increase sigma, otherwise reduce it

```
if (H < log(perplexity, 2)){
      ..............    }
```

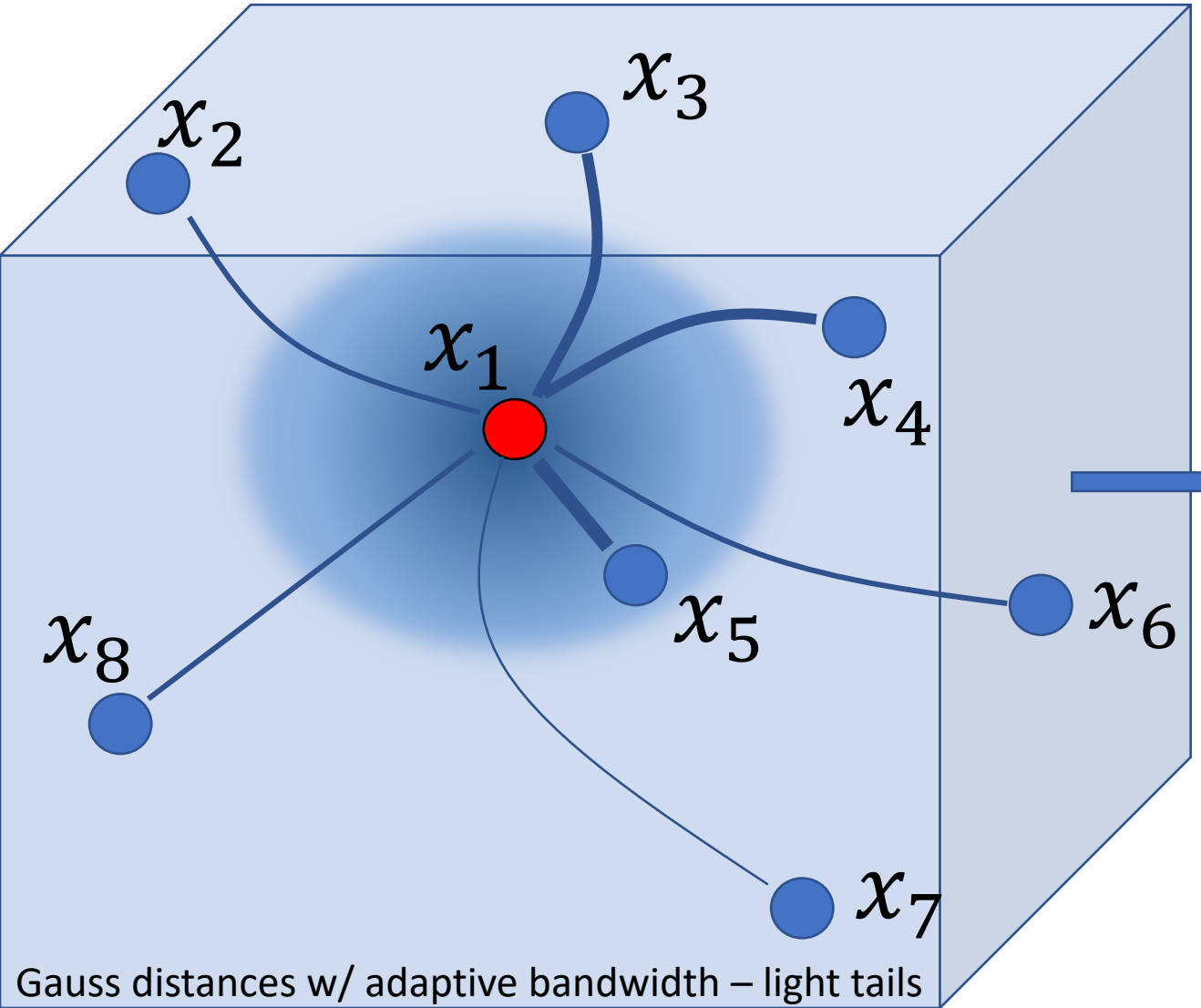


```
function(X, perplexity=15)
{

  D <- as.matrix(dist(X))^2
  P <- matrix(0, nrow(X), nrow(X))
  svals <- rep(1, nrow(X))

  for (i in seq_along(svals))
  {
    srange <- c(0, 100)
    tries <- 0

    for(j in seq_len(50))
    {
      Pji <- exp(-D[i, -i] / (2 * svals[i]))
      Pji <- Pji / sum(Pji)
      H <- -1 * Pji %*% log(Pji, 2)

      if (H < log(perplexity, 2))
      {
        srange[1] <- svals[i]
        svals[i] <- (svals[i] + srange[2]) / 2
      } else {
        srange[2] <- svals[i]
        svals[i] <- (svals[i] + srange[1]) / 2
      }
    }
    P[i, -i] <- Pji
  }

  return(0.5 * (P + t(P)) / sum(P))
}
```

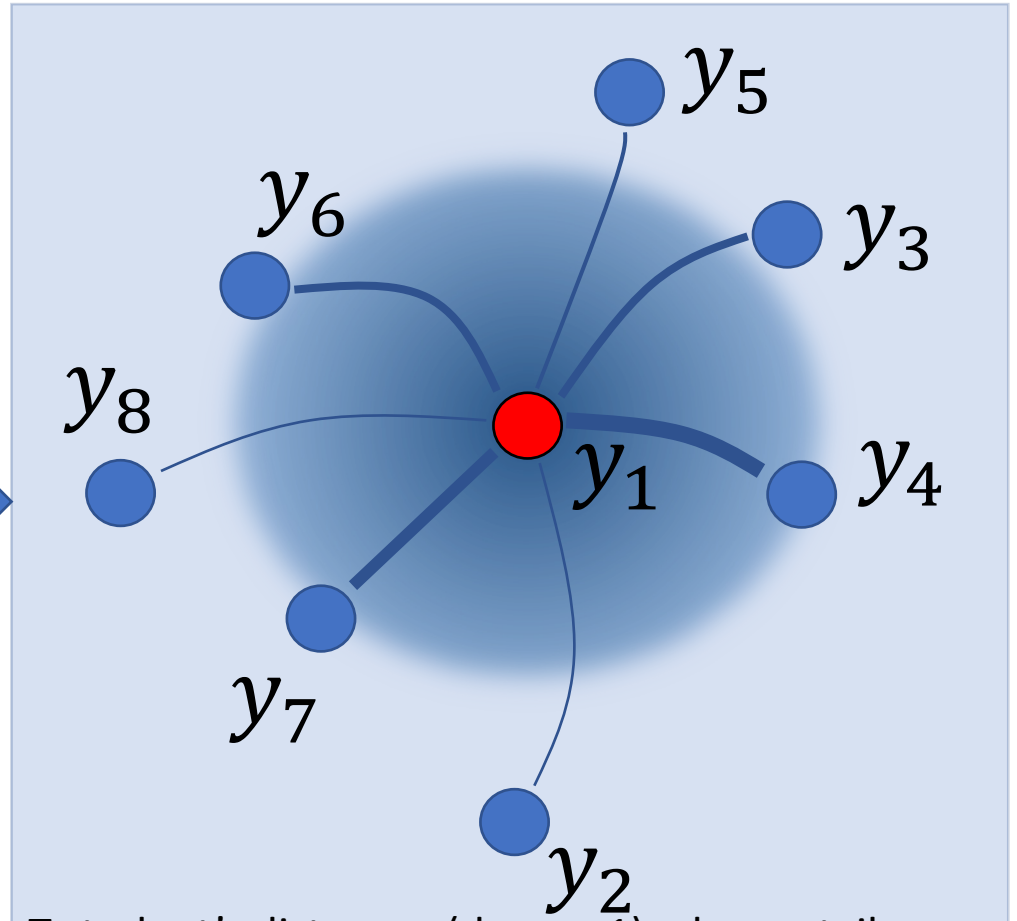
tSNE: optimization

In dimension D the edges are **fixed**

In dimension d<<D points must be arranged so that the edges' weights optimize a cost function

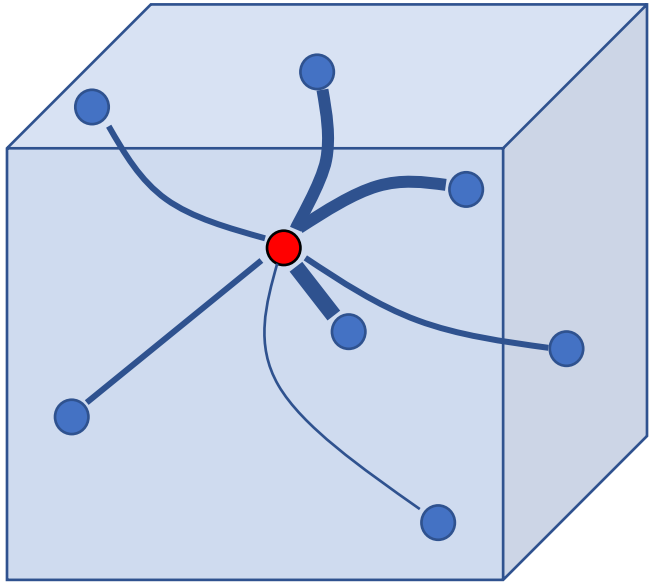


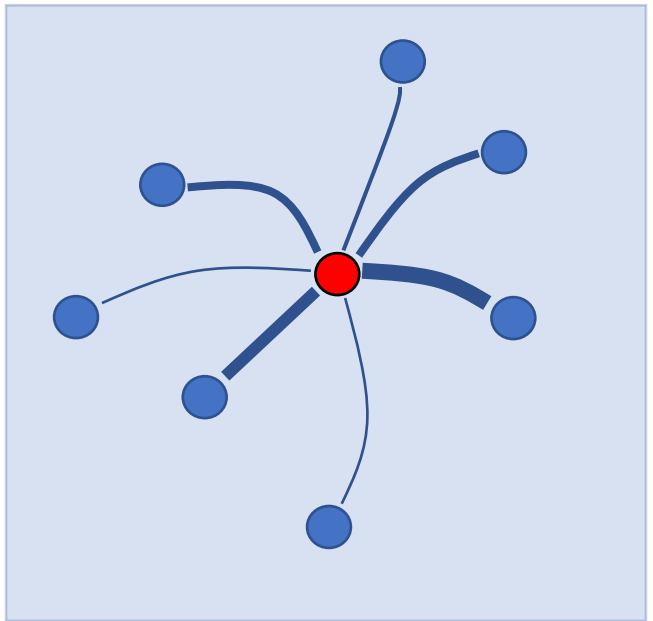
Gauss distances w/ adaptive bandwidth – light tails



T-student's distances (degree 1) – heavy tails

tSNE: optimization



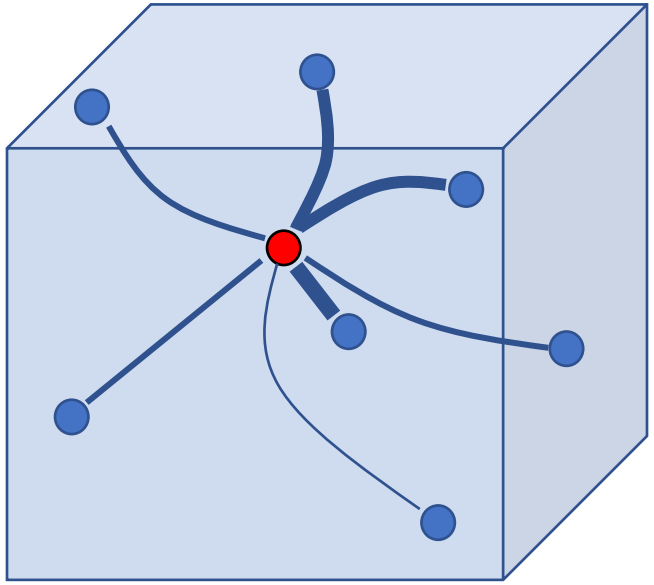


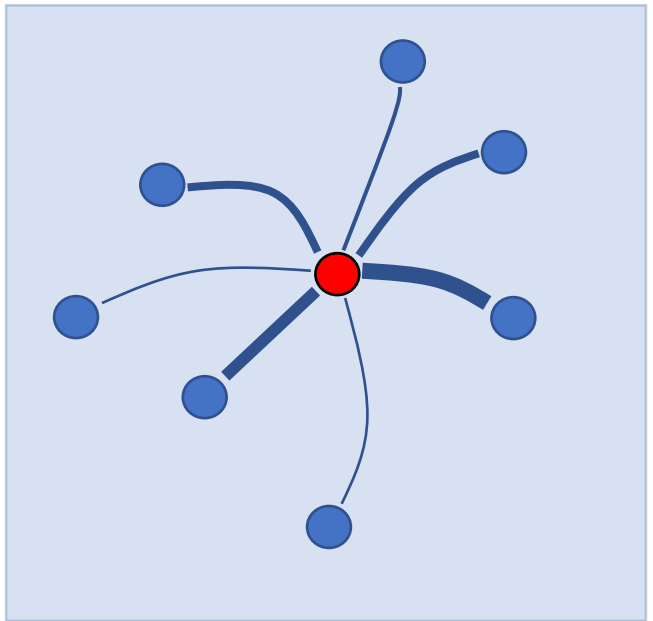
Cost function (KL divergence)

$$C = KL(P|Q) = \sum_{ij} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

- The p's are all fixed (D-dimensional space)
- To make C small (ideally zero), you want q's similar to p´s
- Normalization helps avoiding clusters of packed points
- tSNE mostly cares about how points are distributed locally even though it uses a t-student distribution

tSNE: optimization





- Optimization done with gradient descent.
- The gradient is w.r.t. the low-dimensional variables, then
- It is a vector of length *d* with each component being

$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) \left(1 + \|y_i - y_j\|_2^2\right)^{-1} [y_i - y_j]$$

tSNE: optimization

The gradient of C is

$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) \left(1 + \|y_i - y_j\|_2^2\right)^{-1} [y_i - y_j]$$

Why?

$$\frac{\partial C}{\partial y_i} = \frac{\partial C}{\partial d_{ij}} \frac{\partial d_{ij}}{\partial y_i}$$

With $d_{ij} = \|y_i - y_j\|_2^2$ helps in doing the derivatives.

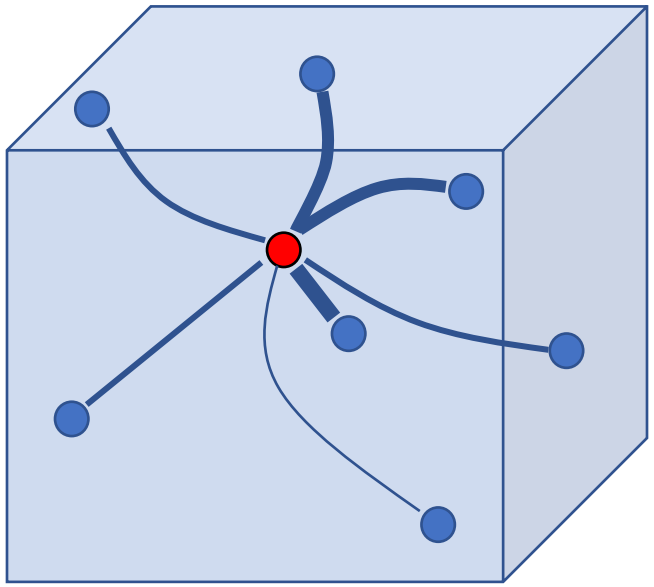
Look here.

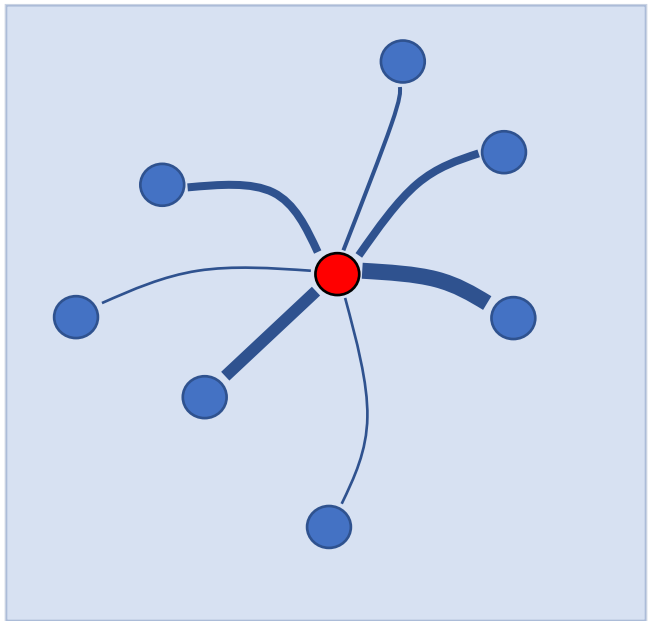
Or also, multiplying both sides by q_{ij} :

$$4 \left(\sum_{j \neq i} p_{ij} q_{ij} Z \left(\mathbf{y}_i - \mathbf{y}_j\right) - \sum_{j \neq i} q_{ij}^2 Z \left(\mathbf{y}_i - \mathbf{y}_j\right) \right)$$

$$Z = \sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}$$

tSNE: optimization





Optimization done with gradient descent. The gradient of C is

$$4\left(\sum_{j\neq i} p_{ij}q_{ij}Z\left(\mathbf{y}_i-\mathbf{y}_j\right)-\sum_{j\neq i} q_{ij}^2 Z\left(\mathbf{y}_i-\mathbf{y}_j\right)\right)$$

Attractive forces P: keep close points together

Repulsive forces Q: push points away from each other

tSNE: optimization

- Calculate pairwise distances Q_{ij} with t-distribution
- Normalize distances
- Calculate the first part of the cost function's gradient

$$4 \cdot \sum_{j \neq i} (P_{i,j} - Q_{i,j}) \cdot \left(1 + ||y_i - y_j||_2^2\right)^{-1}$$

- Multiply by the difference ($\mathbf{y}_i$-$\mathbf{y}_j$). This is done all at once with the matrix Y and by casting the dimensions
- Apply gradient descent and zero-center the data

```
function(X, perplexity=30, k=2L, iter=1000L, rho=100) {

  Y <- matrix(rnorm(nrow(X) * k), ncol = k)
  P <- casl_tsne_p(X, perplexity)
  del <- matrix(0, nrow(Y), ncol(Y))

  for (inum in seq_len(iter))
  {
    num <- matrix(0, nrow(X), nrow(X))
    for (j in seq_len(nrow(X))) {
      for (k in seq_len(nrow(X))) {
        num[j, k] = 1 / (1 + sum((Y[j,] - Y[k, ])^2))
      }
    }
    diag(num) <- 0
    Q <- num / sum(num)

    stiffnesses <- 4 * (P - Q) * num
    for (i in seq_len(nrow(X)))
    {
      del[i, ] <- stiffnesses[i, ] %*% t(Y[i, ] - t(Y))
    }

    Y <- Y - rho * del
    Y <- t(t(Y) - apply(Y, 2, mean))
  }

  Y
}
```

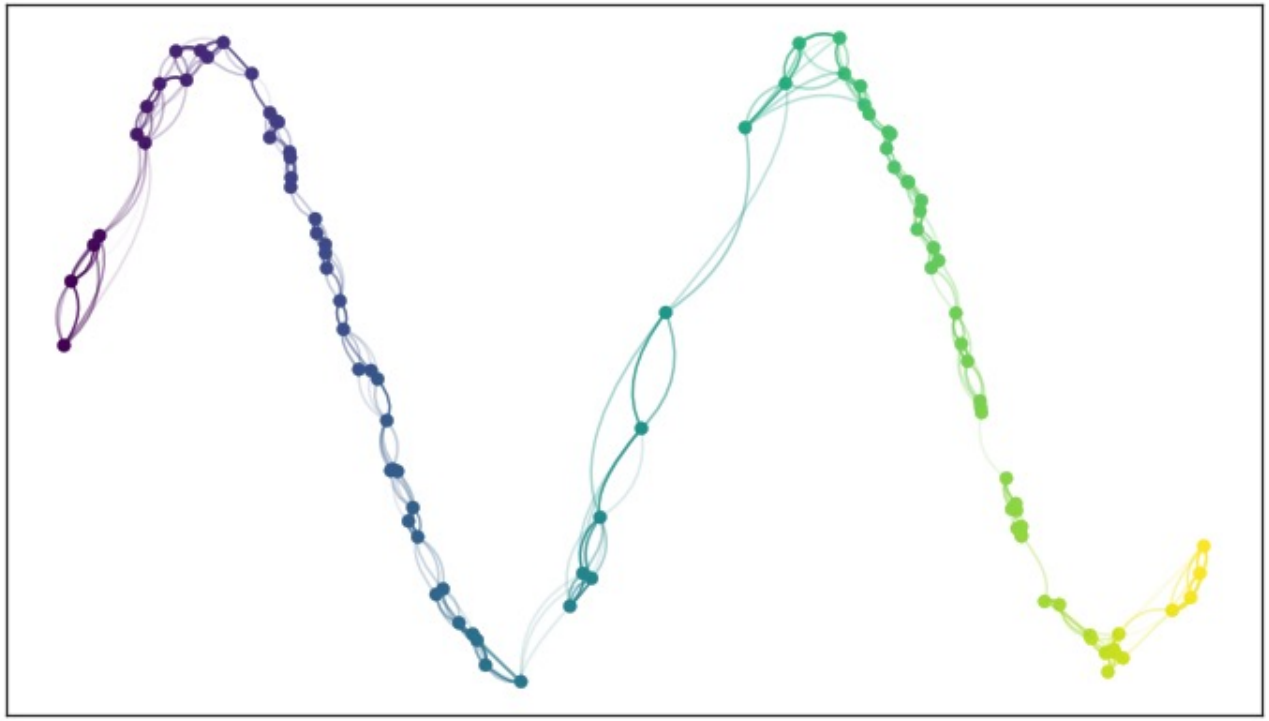
tSNE: optimization

- **Effective** makes good projections on non-linear manifolds
- **It makes sense** in the way it is formulated
- The locality of tSNE can be tuned (perplexity parameters)
- **Exponential cost** in the number d of tSNE dimensions
- **Does not work "online":** if you get a new data point, you have to redo ALL the calculations
- **Labels** informative of data points cannot be used

Those issues are solved by another graph-based technique called **UMAP**

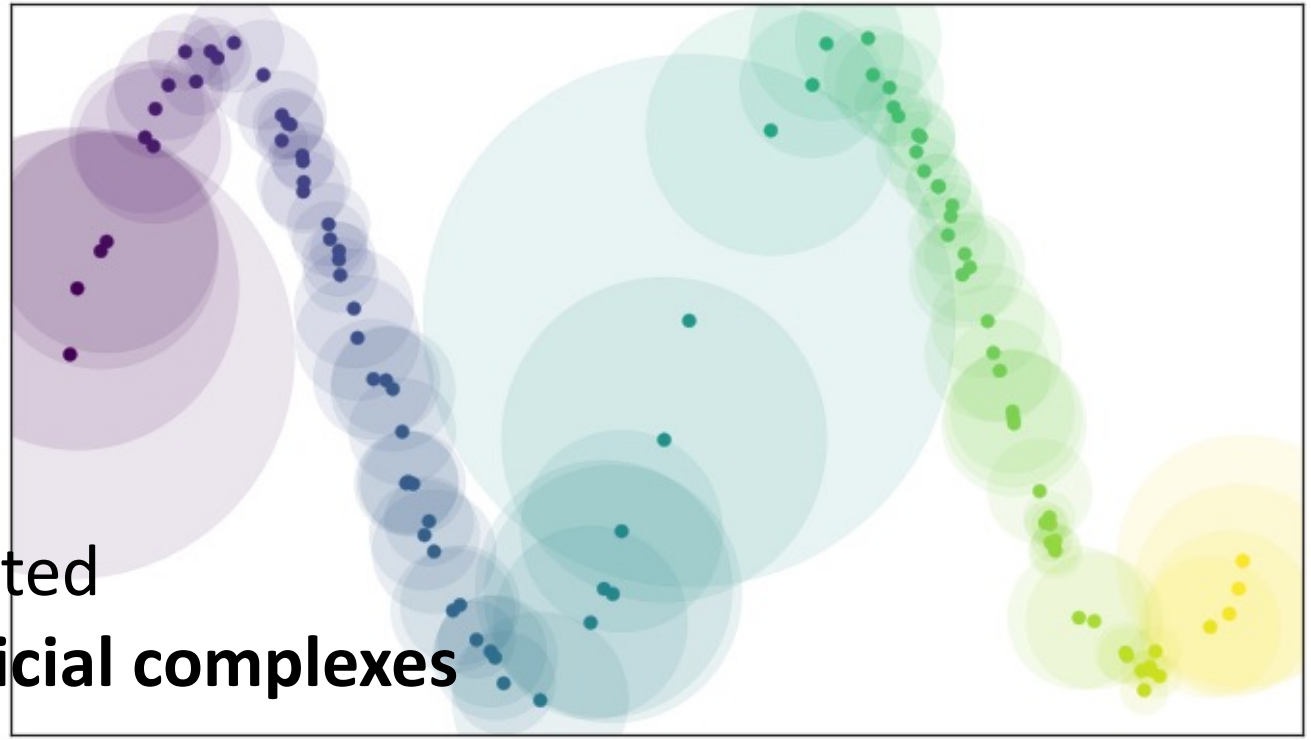
UMAP: Method in brief

- It improves tSNE by use of topology theory
- Topological construction leads to a cost function with different attractive/repulsive forces
- Key to this is to use distances rendering the points **uniform on the manifold**



UMAP : Method in brief

- It improves tSNE by use of topology theory
- Topological construction leads to a cost function with different attractive/repulsive forces
- Key to this is to use distances rendering the points **uniform on the manifold**
- The manifold is approximated by an **open cover of simplicial complexes**



UMAP : Flavours

- Standard umap (both python and R) is **unsupervised non-parametric**
- It can be used to project labelled data, and to assign a test dataset to each class afterwards (**supervised non-parametric**)
- An update optimizes the cost function with a neural network and the NN weights are used as a parametric function to assign test data (**supervised parametric**)

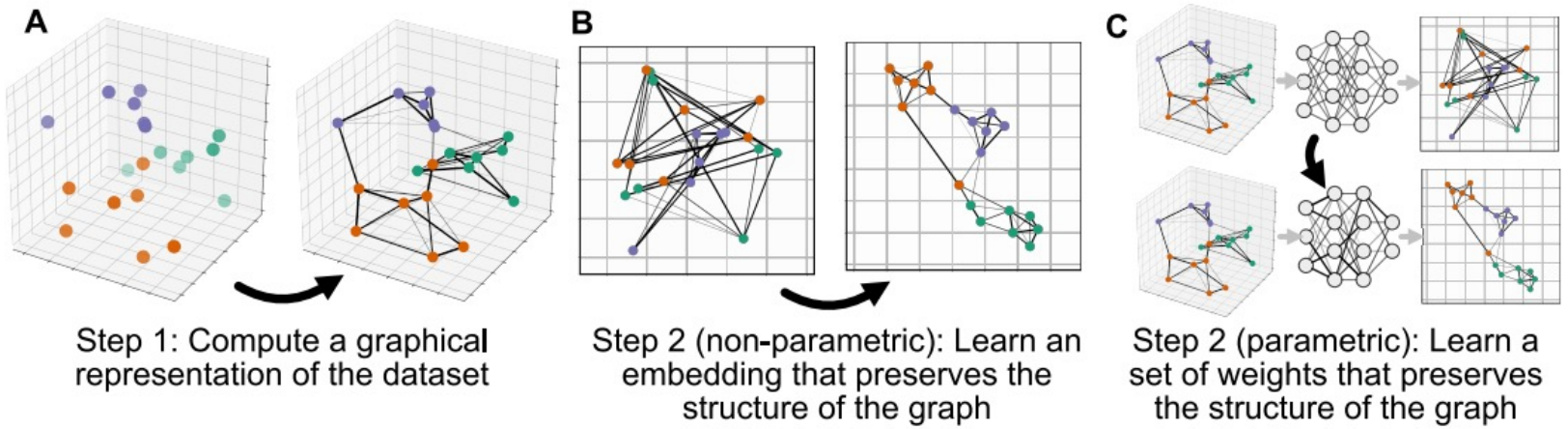
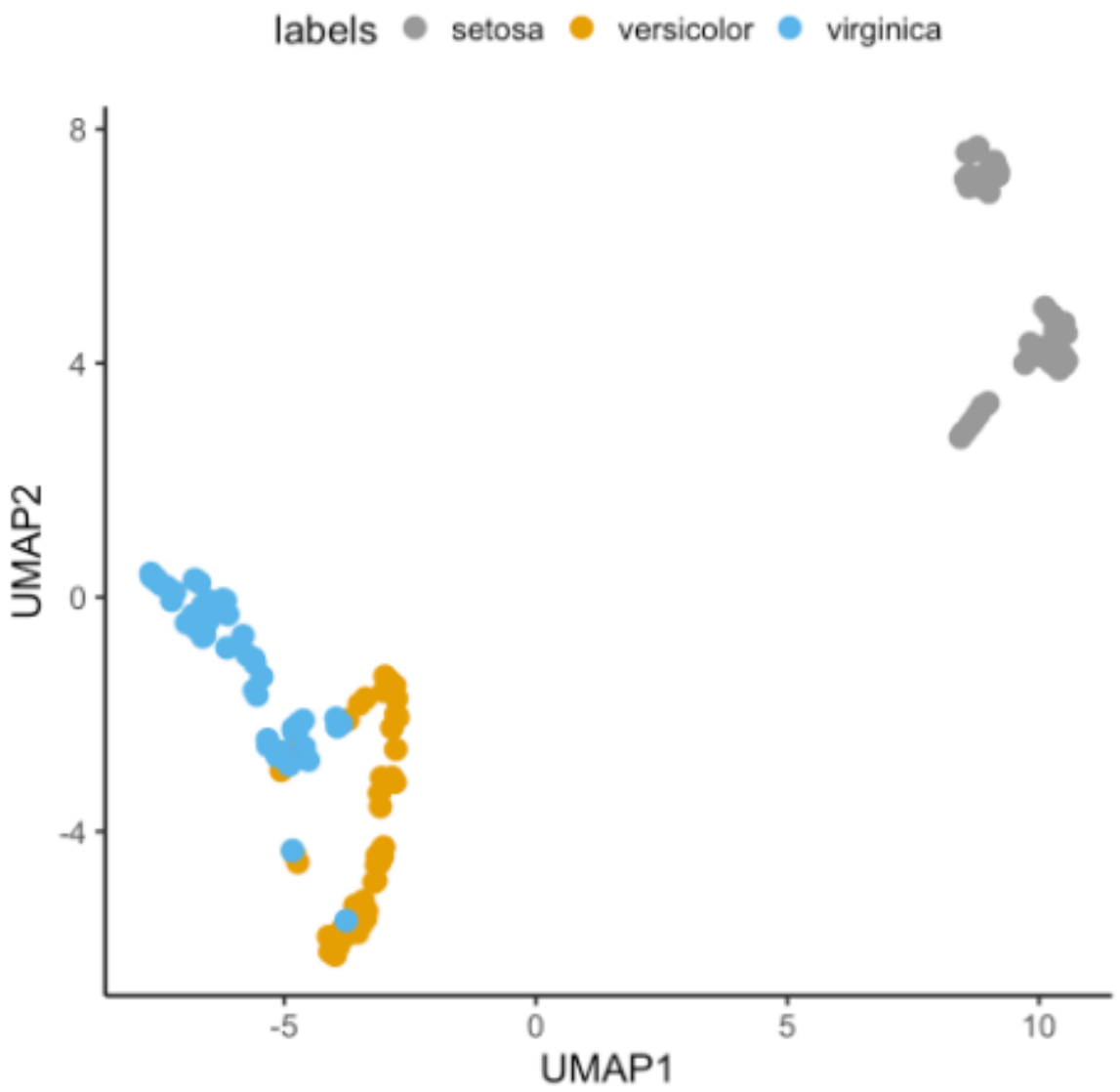


Figure 1: Overview of UMAP (A → B) and Parametric UMAP (A → C).

UMAP：Iris data



tSNE and UMAP: more resources

- UMAP for tSNE: webpage of the author of the R *uwot* package, with deep and detailed explanations about how tSNE and UMAP work. Worth reading.

- A page for the python package *openTSNE:* Here they explain some other improvements for tSNE, like interpolation and Barnes-Hut approximation.

- Barnes-Hut approximation. How does the Barnes-Hut approximation work? Here you find the explanation and a clear animation you can tune to see the results of the algorithm.

- Misread tSNE. Some simple interactive use cases to see how parameters of tSNE change an output.

- Nice theoretical paper on tSNE.

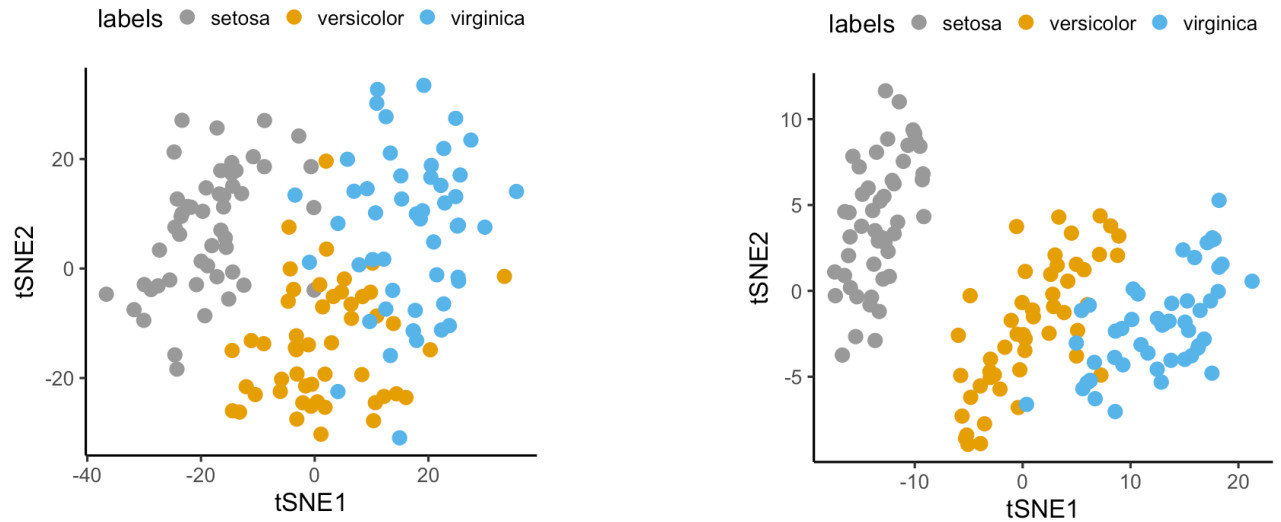
Break – Questions

And

Exercise

tSNE: projection improvements - early exaggeration

Solves the crowding problem (points in general very close with the risk of breaking clusters of similar points)



How: In the factor 4(P - Q) of the gradient, multiply P by a factor (e.g. 12), and in the gradient descent divide the learning rate by the same factor. Go back to normal after a few iterations.

tSNE: projection improvements – momentum

Local optima of the gradient descent can create poor solutions. We can add a term to the descent to avoid such optima.

How: At iteration *t+1* of gradient descent, add the two previous values of Y as follows

$$Y^{(t+1)} = Y^{(t)} - \rho\, 4(P - Q) - \alpha\, (Y^{(t-1)} - Y^{(t-2)})$$

α must be small for some iterations (e.g. between 0.1 and 0.5)
and close to 1 (e.g. 0.9) afterwards.

tSNE: speed improvements – early stop

You can avoid running all iterations of the gradient descent by evaluating if the relative change of KL divergence

$$|KL_{i+1} - KL_i\ |/|KL_i|$$

Is lower than a fixed value ϵ, for example $\epsilon = 1e - 5$:

$$|KL_{i+1} - KL_i\ | < \epsilon\ |KL_i|$$

tSNE: speed improvements – *kNN* and *dist*

Calculating distances in has quadratic cost.

- Use KNN from the package dbscan to calculate distances in the function *casl_tsne_p*

- Use a transformation of *dist(Y)* instead of the double *for* loop inside the *casl_tsne* function

```
for (inum in seq_len(iter)) {
    num <- matrix(0, nrow(X), nrow(X))
    for (j in seq_len(nrow(X))) {
        for (k in seq_len(nrow(X))) {
            num[j, k] = 1 / (1 + sum((Y[j,] - Y[k, ])^2))
        }
    }
```

One line of code

tSNE: misc improvements – inputs

- Provide an initialization from PCA or other algorithms

- If the data has 100s/1000s of dimensions, provide as proxy input a PCA or other projection with fewer dimensions (e.g. 100)

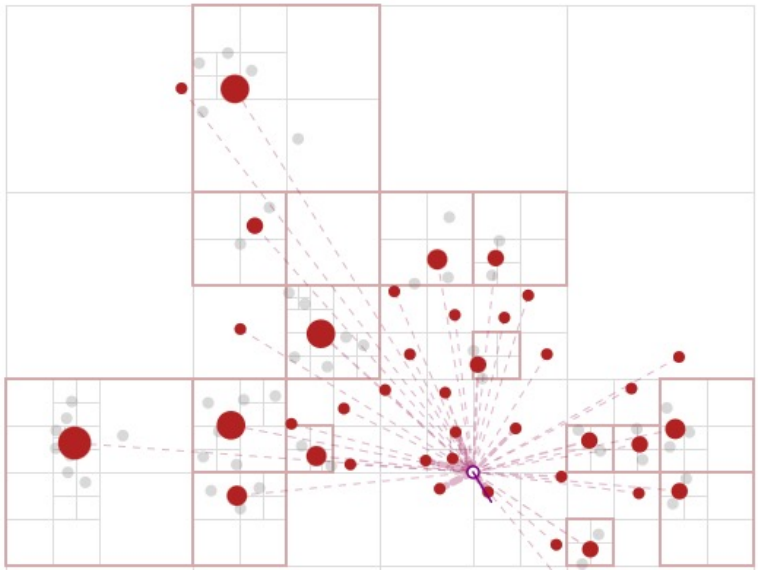
- Remember to center/standardize your data, e.g.

```
X_std = apply(X, 2, function(x){(x-mean(x))/sd(x)})
```

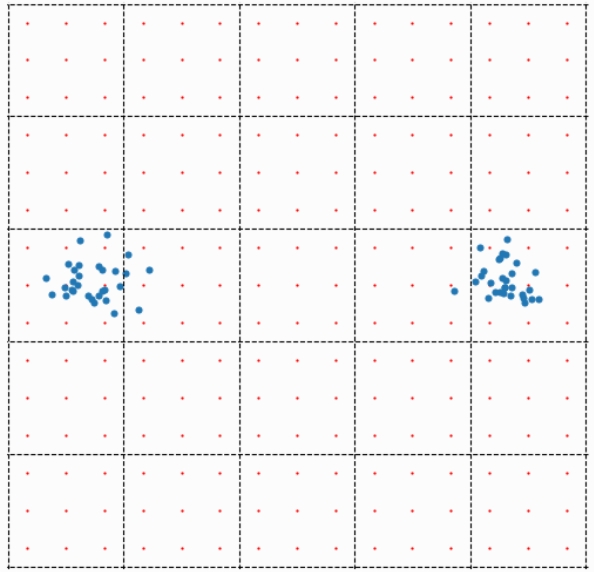
- Clean you NaNs, Inf, … values, transform large values (e.g. income to log(income))

tSNE: Other speed improvements (not for class)

- Barnes-Hut approximation: creates a grid where distances are from the average of the points in each square



- Interpolation: calculate distances on new points and combine them to extrapolate distances on your data



tSNE: benchmarking

```
MB <- microbenchmark("Improved" = { sam_tsne(X_std, init = PCA$ind$coord,
                                    momentum=list(init=.5, final=.9, iter=200L),
                                    early_ex=list(factor=12, iter=200L),
                                    perplexity = 10, rho=100) },
                     "CASL" =  { casl_tsne(X_std, init = PCA$ind$coord,
                                    perplexity = 20, rho=100) },
                      times=1 )
```

```
MB['time (s)'] = MB['time']/1e9
```

```
MB
```

A microbenchmark: 2 × 3

expr	time	time (s)
<fct>	<dbl>	<dbl>
Improved	5214781303	5.214781
CASL	63844027879	63.844028

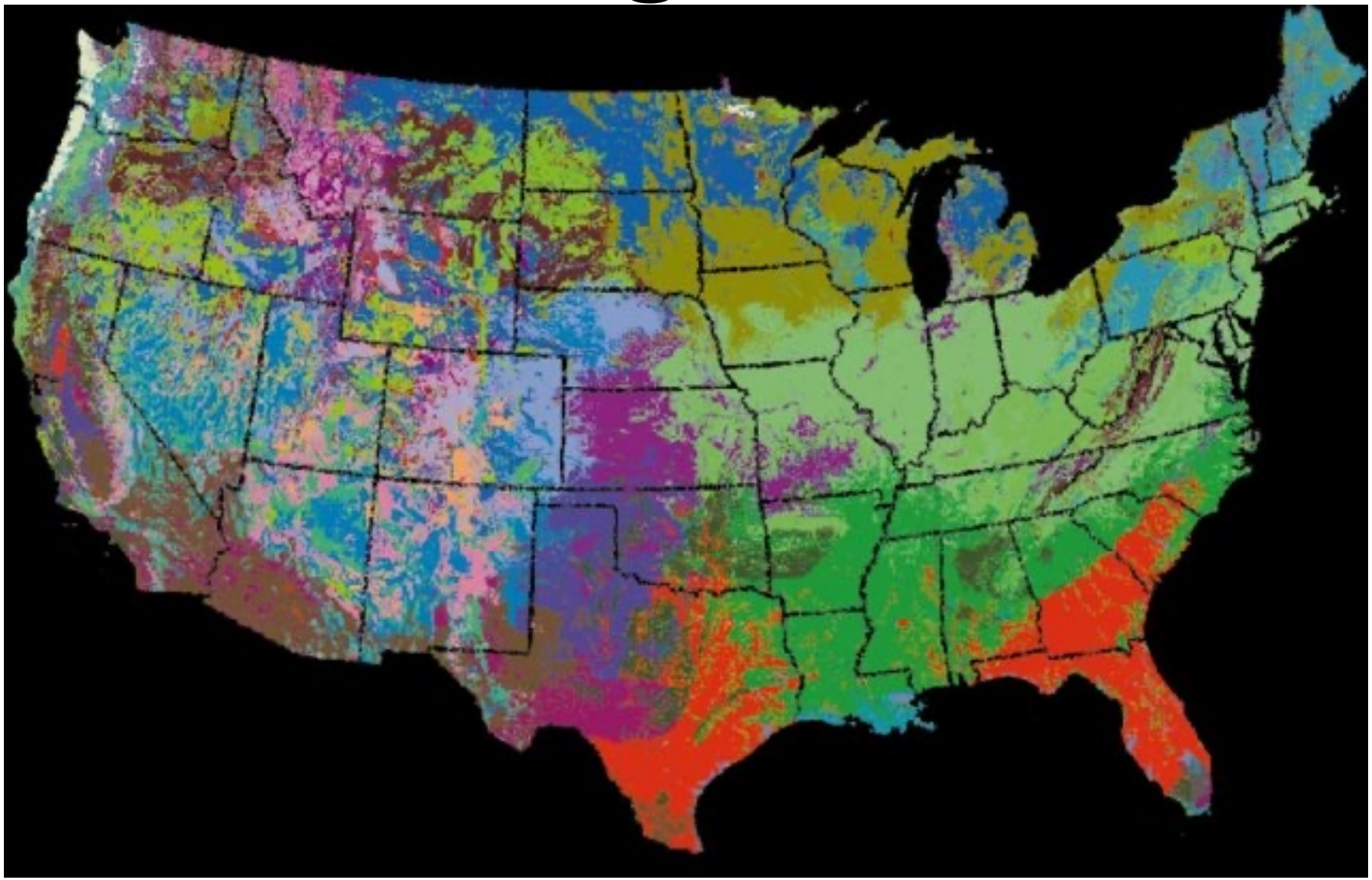
Questions

And

Exercise

Find exercise and code at the webpage
https://samuelesoraggi.github.io/Projection_and_clustering_tutorial/

Clustering methods



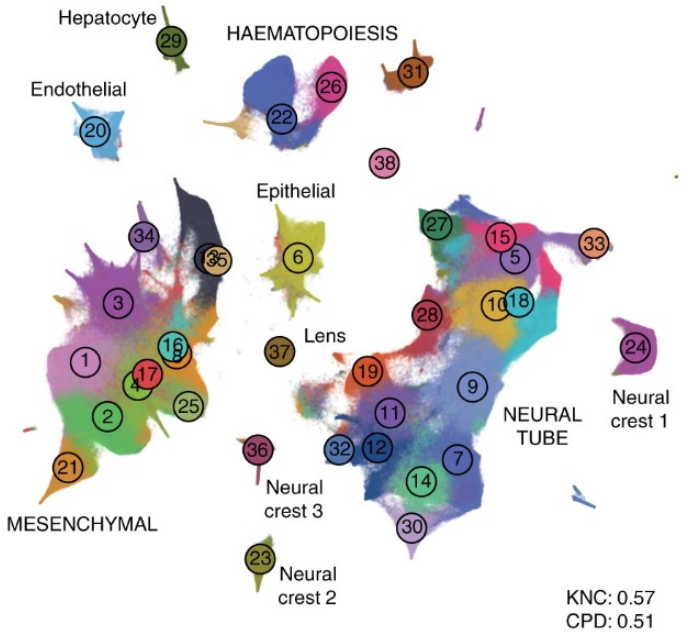
Clustering: definition

The unsupervised task where data points are grouped together according to a concept of being similar, e.g.

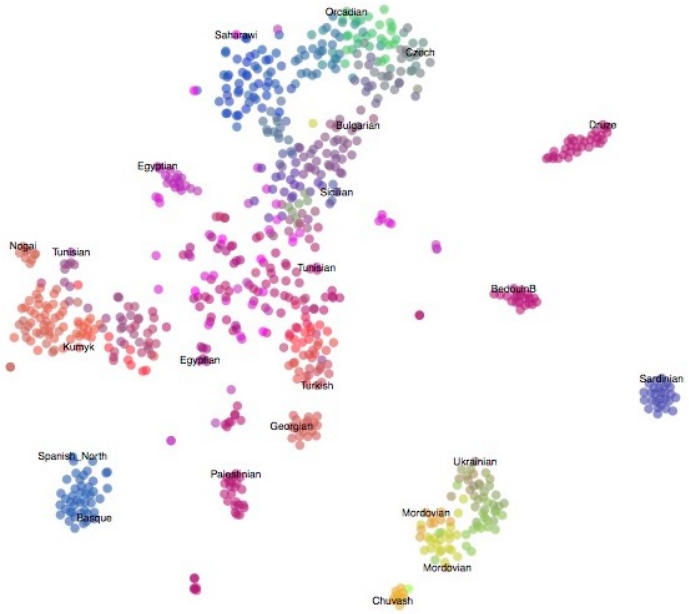
- Small distance between samples
- Samples following a distribution
- Samples in dense areas of space

Clustering: applications

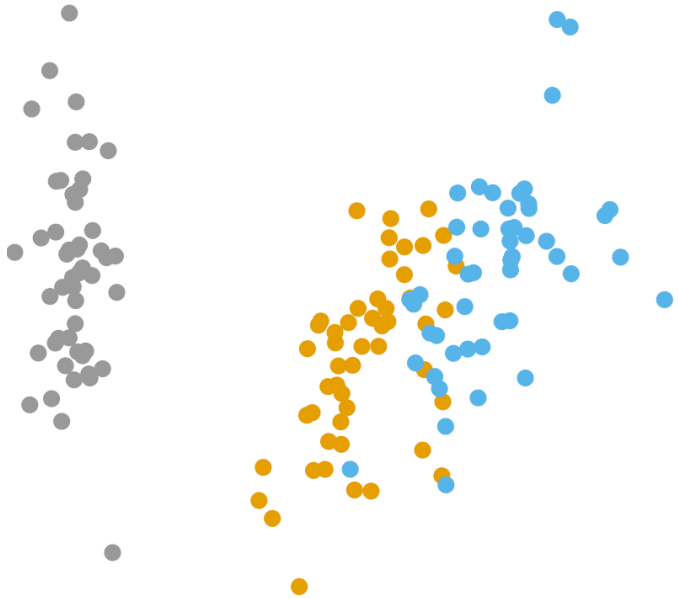
Applications from data projection and the exercises naturally fit into a clustering application



Transcriptomics of brain: finding cell types



Population genomics: identifying similar populations and subpopulations



Iris data exercise: separating flower species

Centroid-based methods

- K-means: probably the most known clustering method
- Each cluster is represented by a point (**centroid**)

Pseudocode (k-means):

Define k centroids for the dataset
While(iterate)
    Assign each point to the closest centroid
    Recalculate centroids as average of assigned points
    If(stopping criteria) then(iterate=False)

Centroid-based methods: k-means

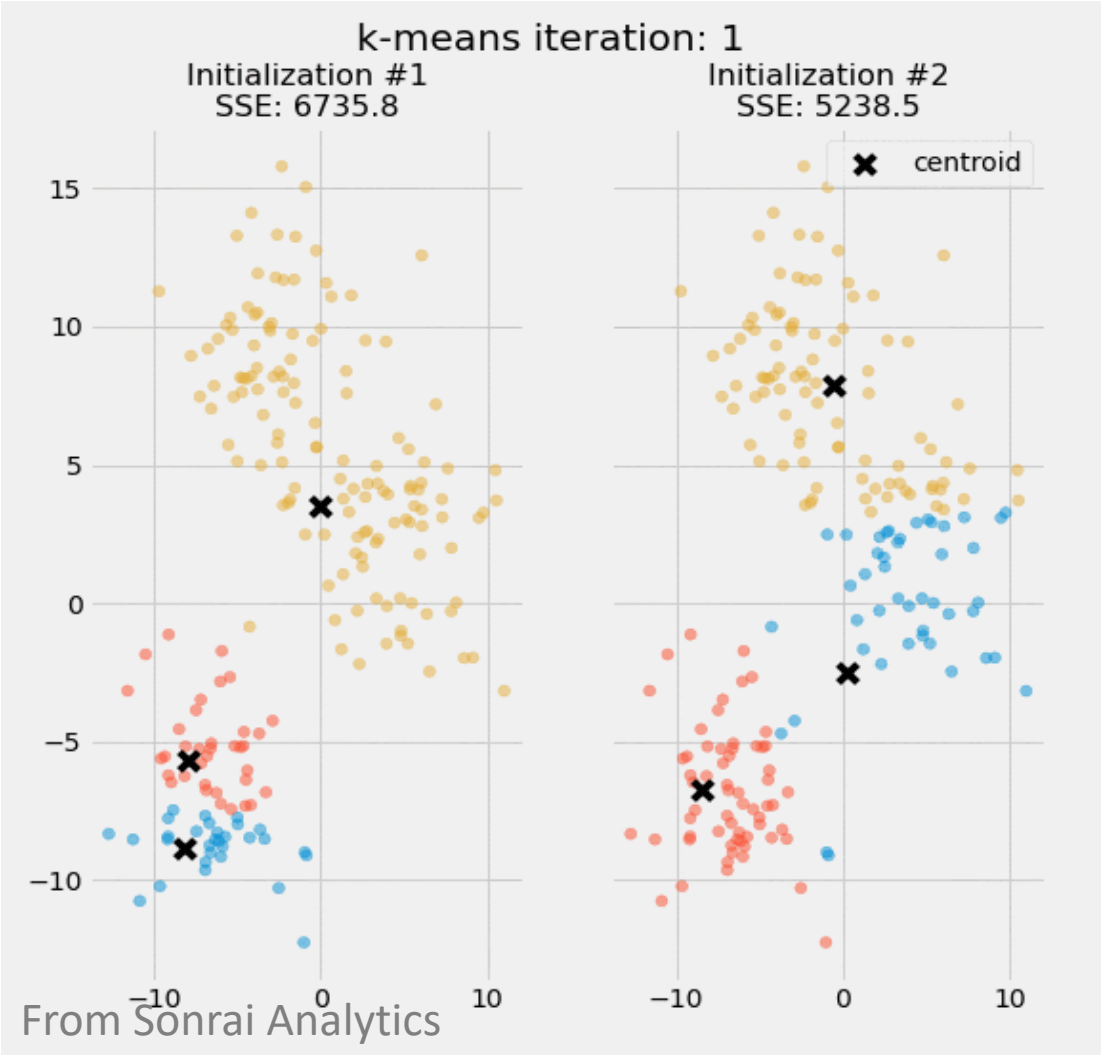
- K-means: probably the most known clustering method
- Each cluster is represented by a point (centroid)

- **Fast**: scales like Nk
- Takes usually **few iterations**
- Not for high-dimensionality: it's **distance based!**
- *k* must be chosen
- Dependent on initialization

Centroid-based methods: k-means

- K-means: probably the most known clustering method
- Each cluster is represented by a point (centroid)

- **Fast**: scales like Nk
- Takes usually **few iterations**
- Not for high-dimensionality: it's **distance based!**
- *k* must be chosen
- Dependent on random initialization



From Sonrai Analytics

Centroid-based methods: k-means

- K-means: probably the most known clustering method
- Each cluster is represented by a point (centroid)

- **Fast**: scales like Nk
- Takes usually **few iterations**
- Not for high-dimensionality: it's **distance based!**
- *k* must be chosen
  - A priori knowledge about *k* (not always true)
  - Hierarchical clustering on a subsample to find a good *k*
- Dependent on random initialization

Centroid-based methods: k-means

- K-means: probably the most known clustering method
- Each cluster is represented by a point (centroid)

- **Fast**: scales like Nk
- Takes usually **few iterations**
- Not for high-dimensionality: it's **distance based!**
- *k* must be chosen
  - A priori knowledge about *k* (not always true)
  - Hierarchical clustering on a subsample to find a good *k*
- Dependent on random initialization
  - Multiple random initializations and choose *"the best result"*
  - Try to assign initial centroids not randomly

Beyond k-means: CLARANS and AP

CLARANS (originating from PAM and CLARA)

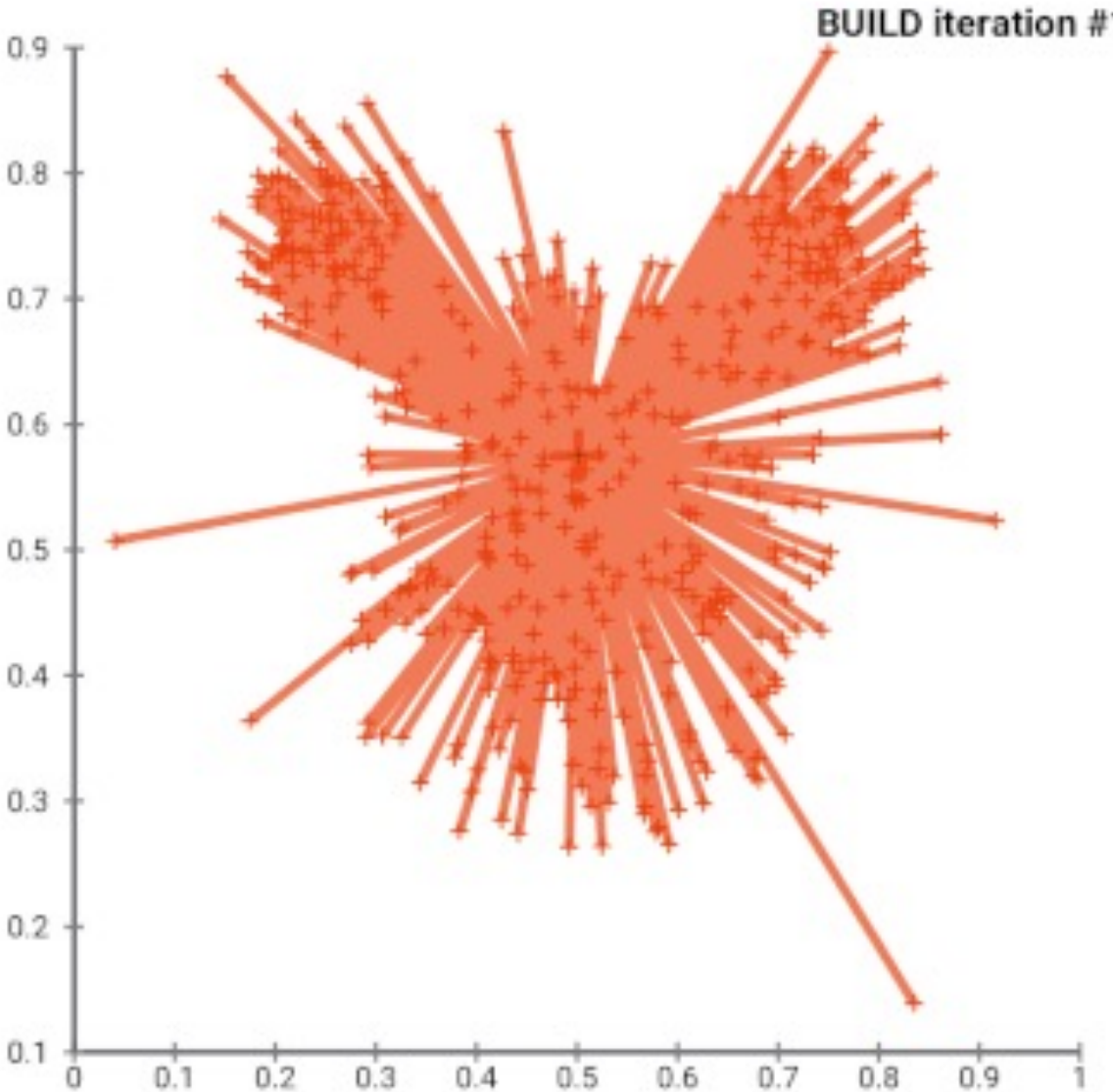
- Works on batches of data (reduced memory usage)
- Centroids are datapoints of the dataset
- Number of clusters not chosen

Affinity Propagation (AP)

- Any point can be a cluster representative
- Based on message passing between datapoints
- Consensus of clustering based on sent-received information
- **No initialization of clusters or choice of *k***
- Tuning of a *subclustering preference* parameter

Beyond k-means: CLARANS and AP

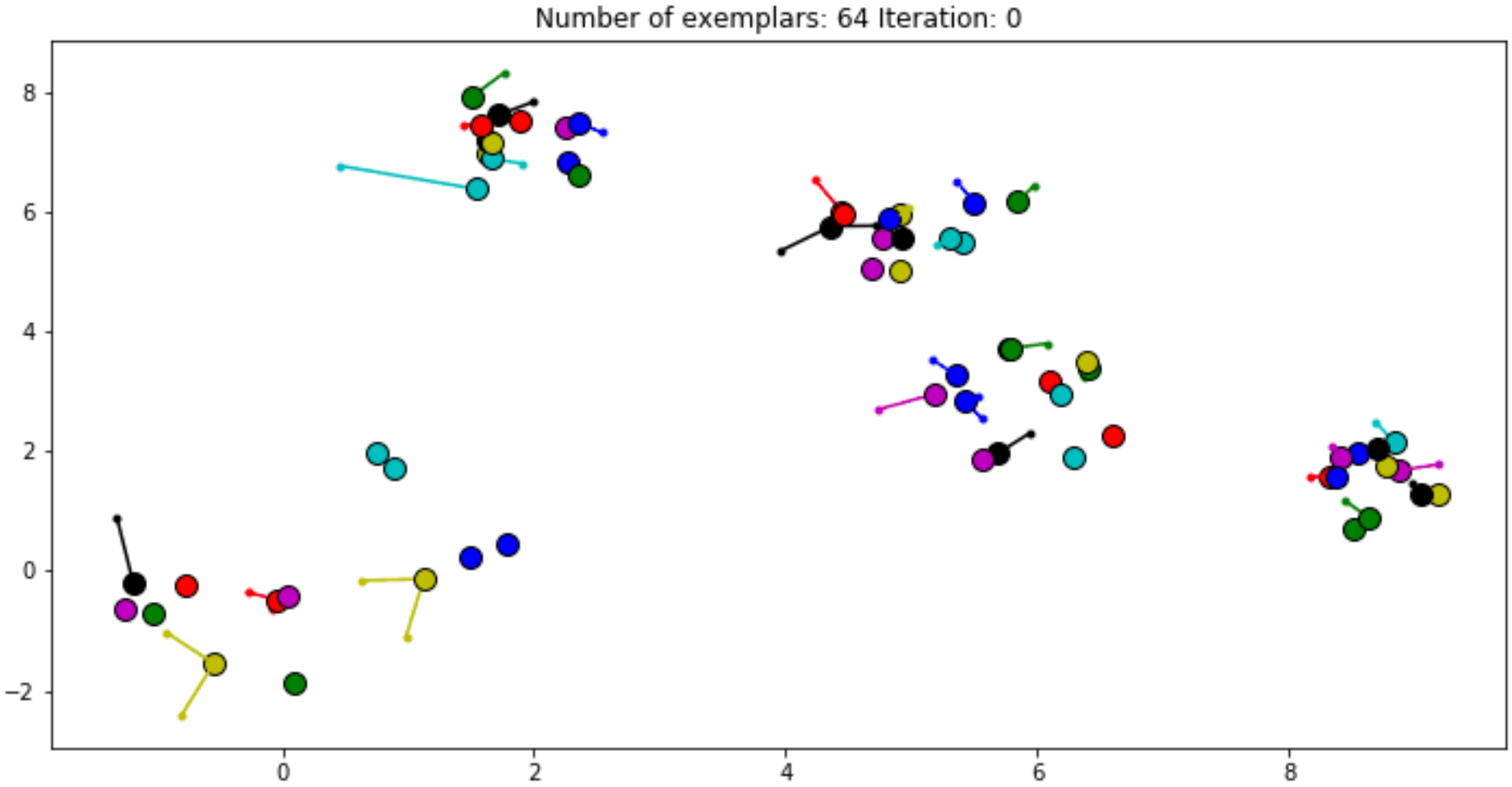
CLARANS



From Wikiwand

Beyond k-means: CLARANS and AP

[Affinity Propagation (AP)]

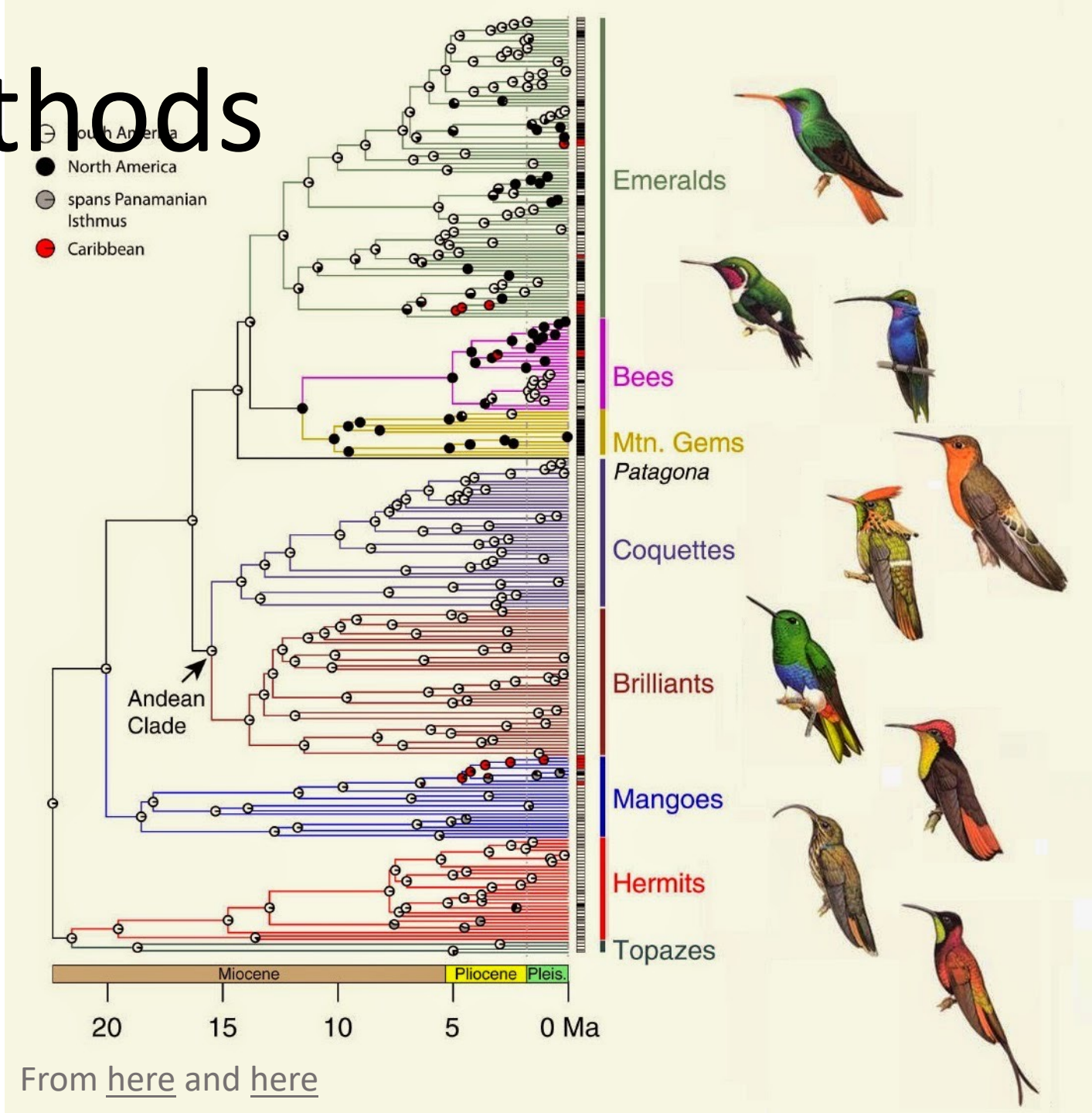


Hierarchical methods

- Develop a hierarchy of datapoints by grouping/splitting them step by step
- Top-down (divisive) or bottom-up (agglomerative)
- Criterias for hierarchy (Linkage) are many and change the output
- Very expensive (N^3 in N datapoints), but best implementations are down to N^2

Hierarchical methods

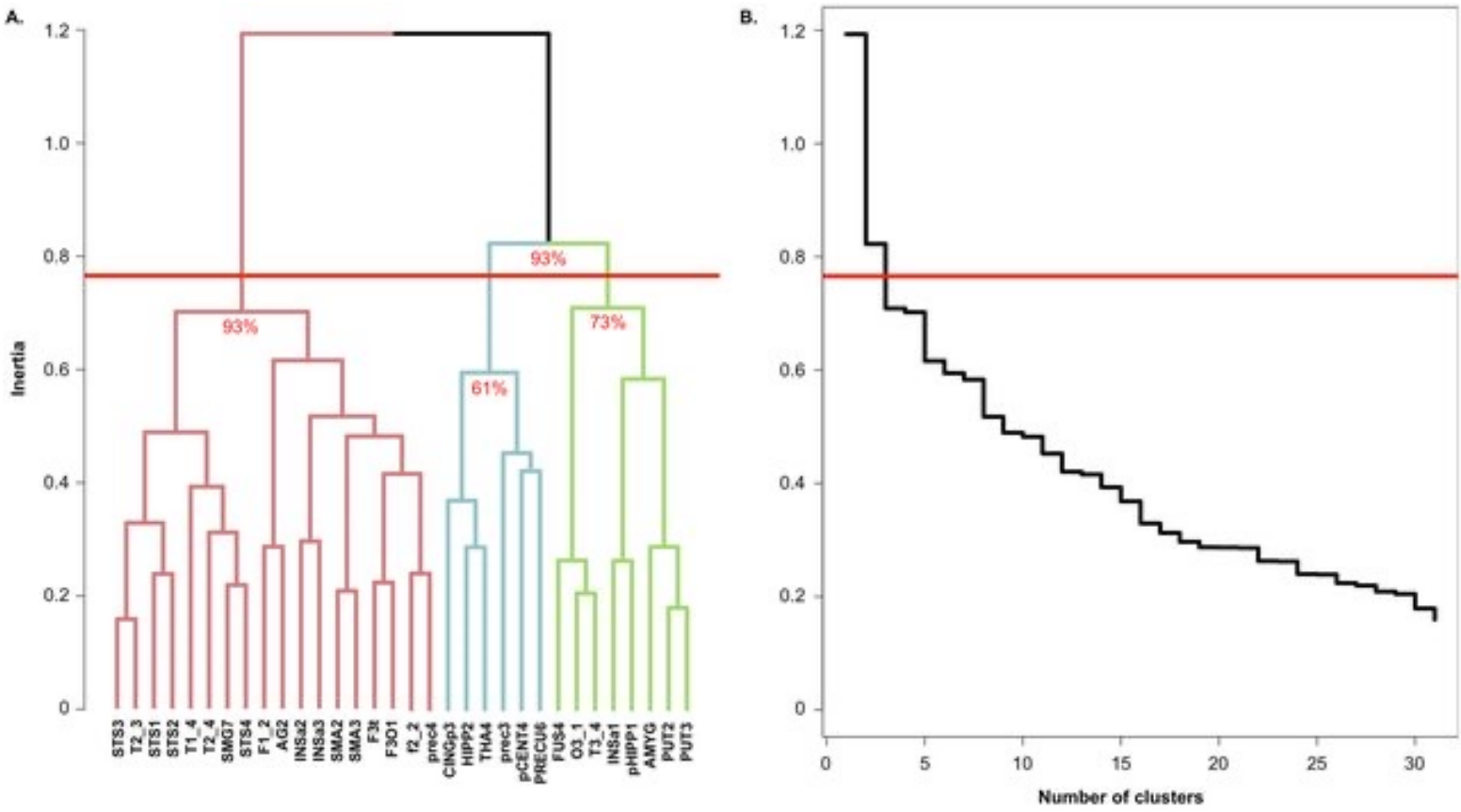
- Develop a hierarchy of datapoints by grouping/splitting them step by step
- Top-down (divisive) or bottom-up (agglomerative)
- Criterias for hierarchy (Linkage) are
  many and change the output
- Very expensive (N^3 in N datapoints),
  but best implementations are down to N^2



From here and here

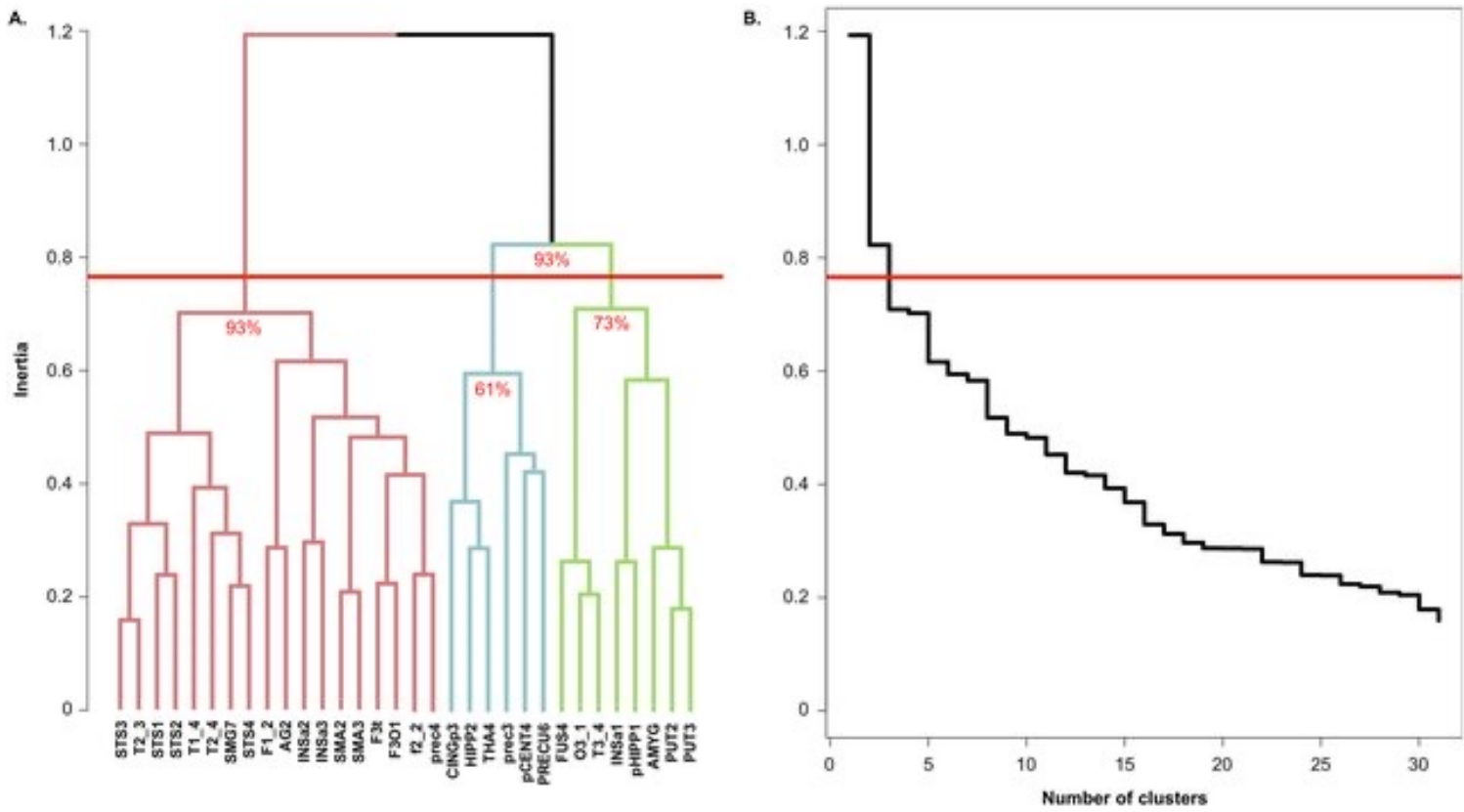
Hierarchical methods

- Develop a hierarchy of datapoints by grouping/splitting them step by step
- Top-down (divisive) or bottom-up (agglomerative)
- Criterias for hierarchy (Linkage) are many and change the outpu
- Very expensive (N^3 in N datapoints),
  but best implementations are down to N^2



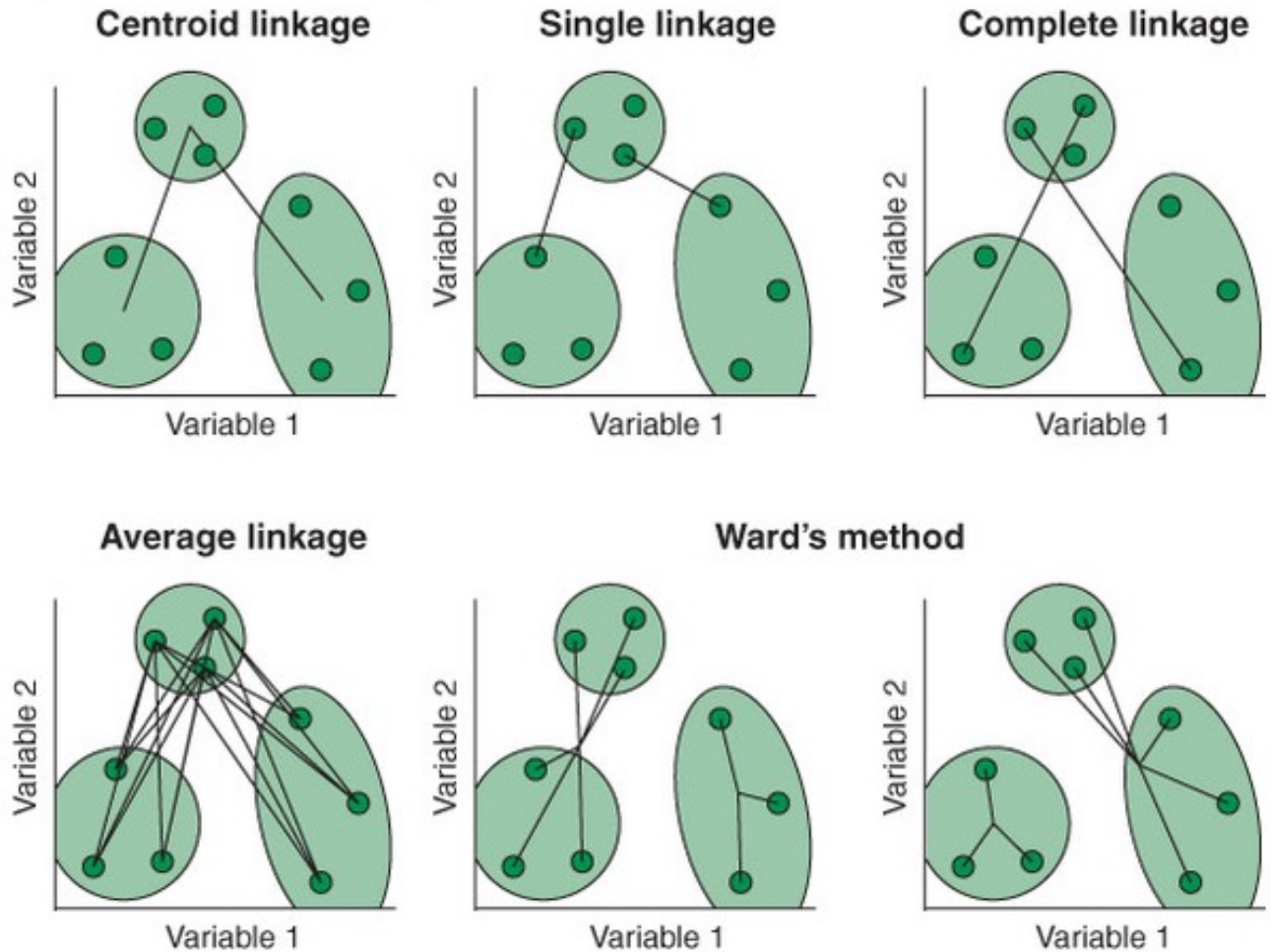
Hierarchical methods: cluster choice

- Inertia is the sum of distances^2 between each point and its cluster centroid
- You can use the bottom-up view to choose a number *k* of clusters for other methods, using a subsample of the data
- You want to do it before small changes in inertia make *k* change a lot



Hierarchical methods: linkage

Figure 17.3. Different linkage methods to define the distance between clusters. Centroid linkage calculates the distance between cluster centroids. Single linkage calculates the smallest distance between clusters. Complete linkage calculates the largest distance between clusters. Average linkage calculates all pairwise distances between cases in two clusters and finds the mean. Ward's method calculates the within-cluster sum of squares for each candidate merge and chooses the one with the smallest value.



Hierarchical methods

	Maribo	Roskilde	Odense	Aarhus
Maribo		126	183	324
Roskilde			130	158
Odense				145
Aarhus				

(M) (R) (O) (A)

Hierarchical methods

	Maribo	Roskilde	Odense	Aarhus
Maribo		126	183	324
Roskilde			130	158
Odense				145
Aarhus				

(M) (R) (O) (A)

(M,R) (O) (A)

Hierarchical methods

	Maribo	Roskilde	Odense	Aarhus
Maribo		126	183	324
Roskilde			130	158
Odense				145
Aarhus				

(M) (R) (O) (A)

(M,R) (O) (A)

Single Linkage

(M,R,O) (A)

Hierarchical methods

	Maribo	Roskilde	Odense	Aarhus
Maribo		126	183	324
Roskilde			$\bar{Y}$ = 156.5	$\bar{Y}$ = 241
			130	158
Odense				145
Aarhus				

(M) (R) (O) (A)

(M,R) (O) (A)

Single Linkage
(M,R,O) (A)

Average Linkage
(M,R) (O,A)

Hierarchical methods

	Maribo	Roskilde	Odense	Aarhus
Maribo		126	183	324
Roskilde			130	158
Odense				145
Aarhus				

(M) (R) (O) (A)

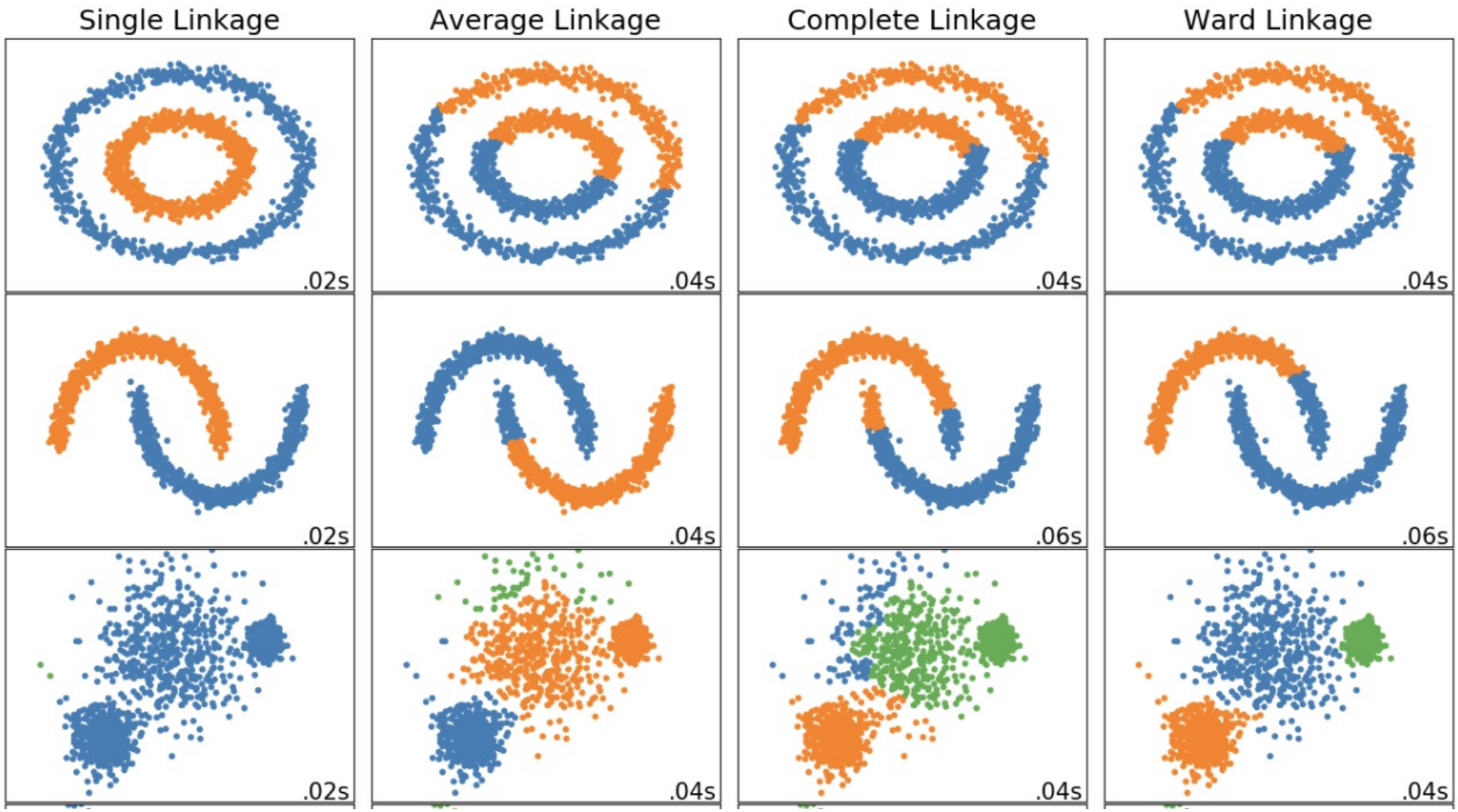
(M,R) (O) (A)

Single Linkage
(M,R,O) (A)

Average Linkage
(M,R) (O,A)

Complete Linkage
(M,R,O) (A)

Hierarchical methods: linkage effect



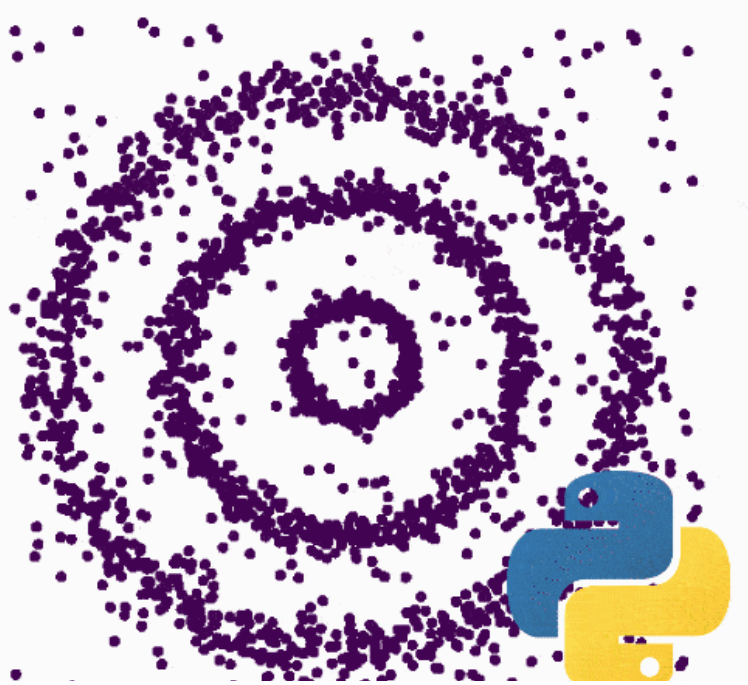
Scikit-learn

Density methods

- Group points in dense areas of space
- Makes use of distances or density kernels (Gaussian, t-Student, …)
- Suffer in low density areas
- Capture arbitrary cluster shapes
- Quadratic cost

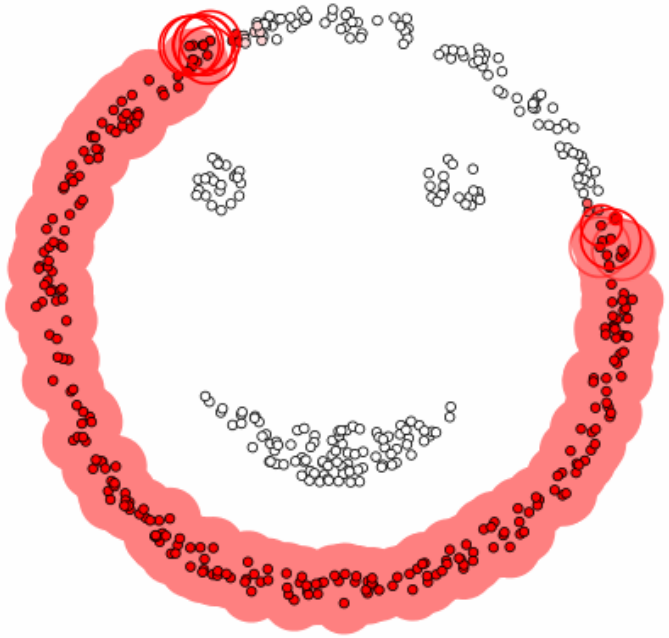
- DBSCAN: classical distance-based
- HDBSCAN: hierarchical based on densities
- FN-DBSCAN: probabilistic neighborhood
- DENCLUE 2: based on density kernel

DBSCAN



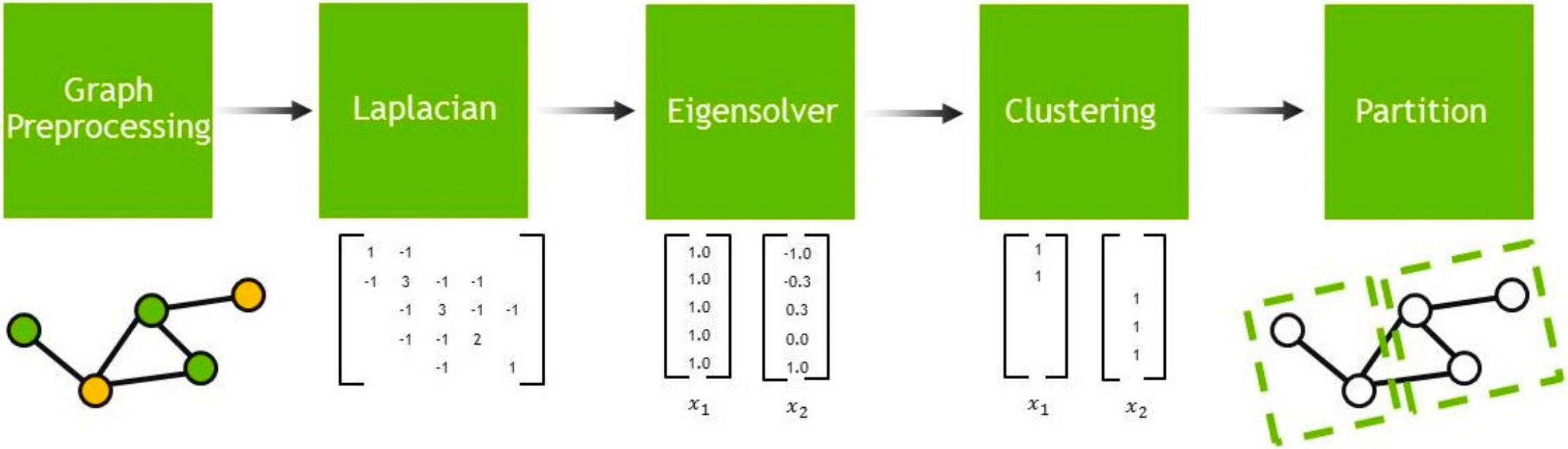
Pueble Rino

HDBSCAN



Kavja Gajjar

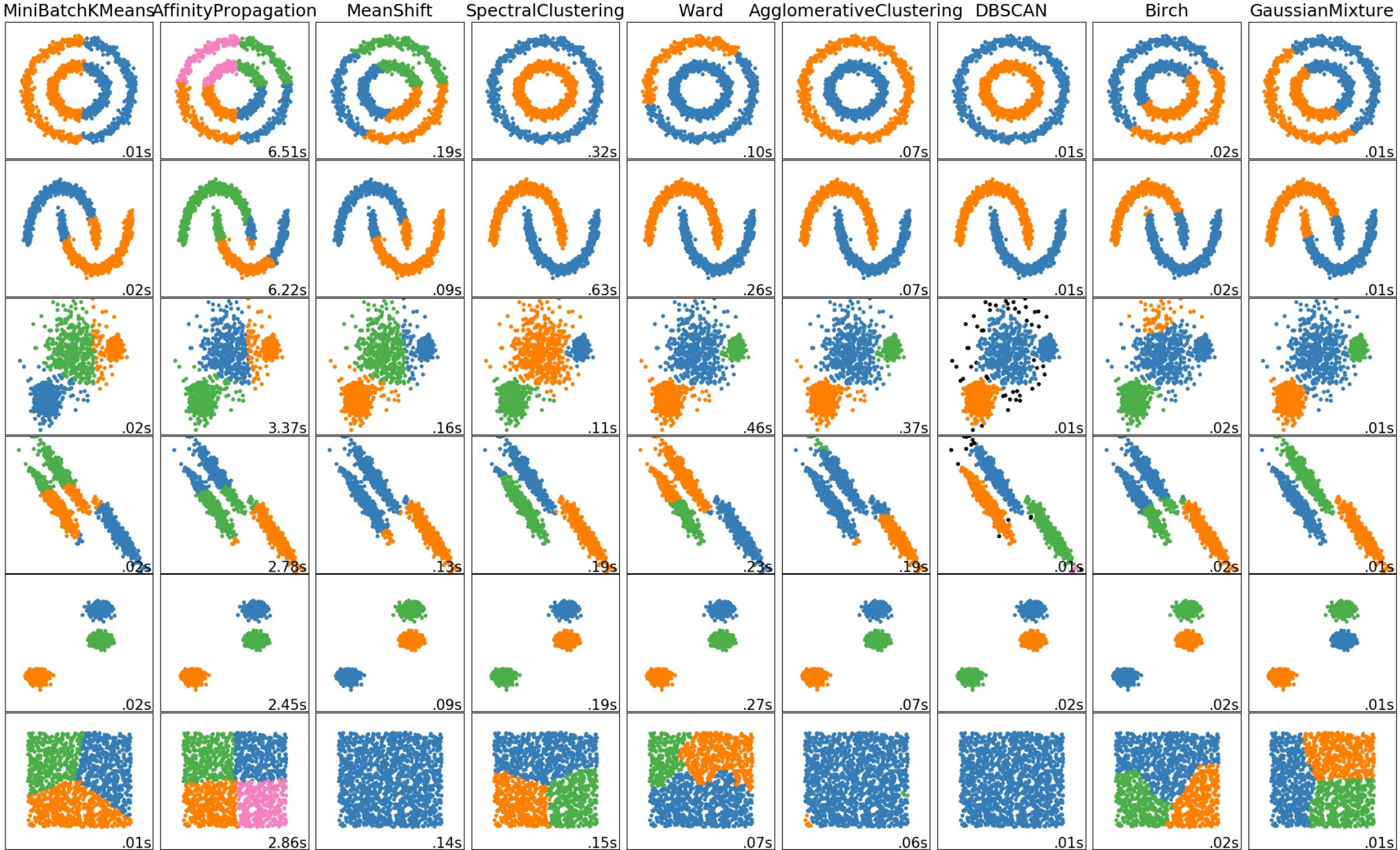
Bonus: spectral clustering



Nvidia

- Represents the data as a graph through a Laplacian matrix *L*
- Applies SVD to *L,* and uses one of the previous algorithms on the features (eigenvectors)

- L often ill-conditioned
- Calculation of svd costs Nk (k features)
- Graph similarities calculated through kernel has quadratic cost, reduced to N with nearest neighbors


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
AgglomerativeClustering
DBSCAN
Birch
GaussianMixture
.01s
6.51s
.19s
.32s
.10s
.07s
.01s
.02s
.01s
.02s
6.22s
.09s
.63s
.26s
.07s
.01s
.02s
.01s
.02s
3.37s
.16s
.11s
.46s
.37s
.01s
.02s
.01s
.02s
2.78s
.13s
.19s
.23s
.19s
.01s
.02s
.01s
.02s
2.45s
.09s
.19s
.27s
.07s
.02s
.02s
.01s
.01s
2.86s
.14s
.15s
.07s
.06s
.01s
.02s
.01s
Scikit-learn

Questions

And

Exercise

Find exercise and code at the webpage
https://samuelesoraggi.github.io/Projection_and_clustering_tutorial/