

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



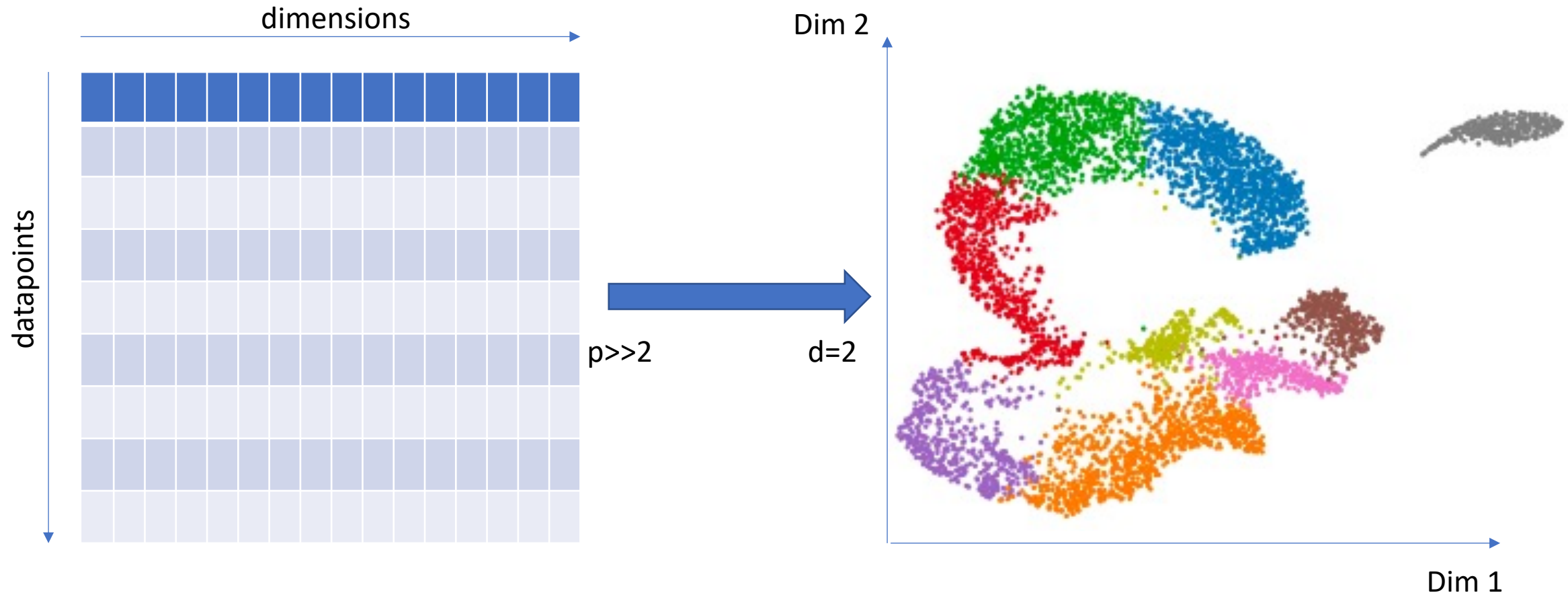
- Recognize types of dim.red. methods
- Master principles of how they work
- Some examples and practical issues

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

- Conclusions
- Break
- Exercise illustration and setup

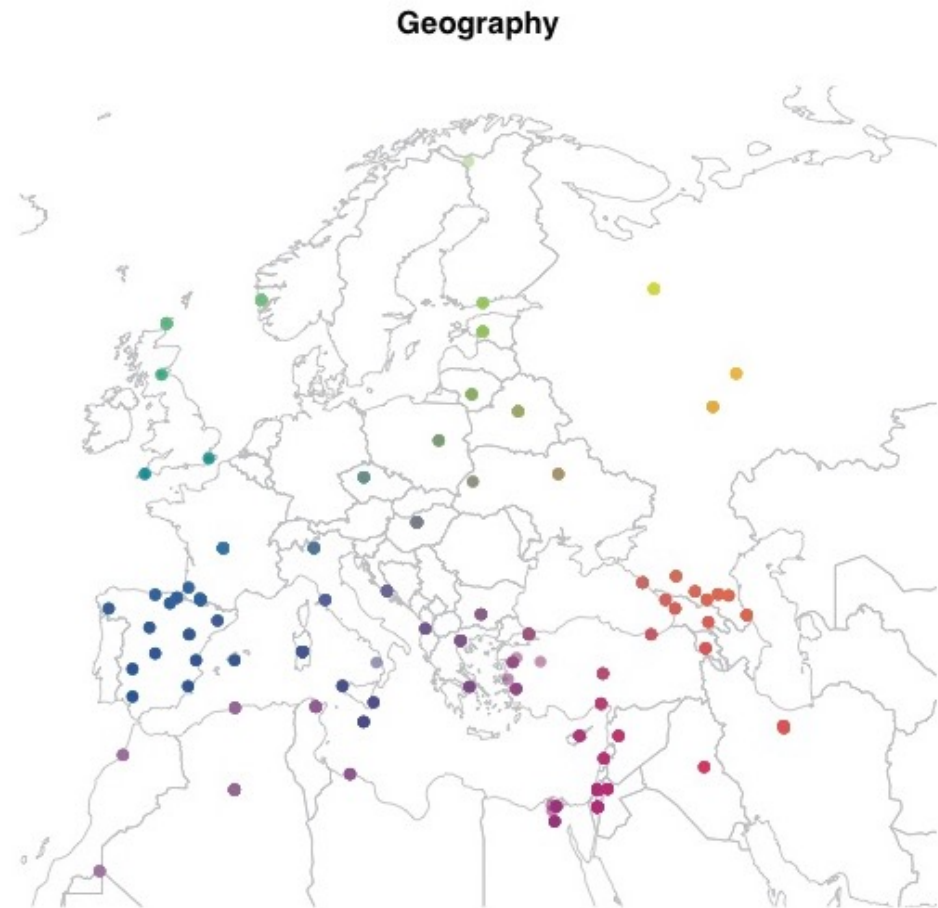
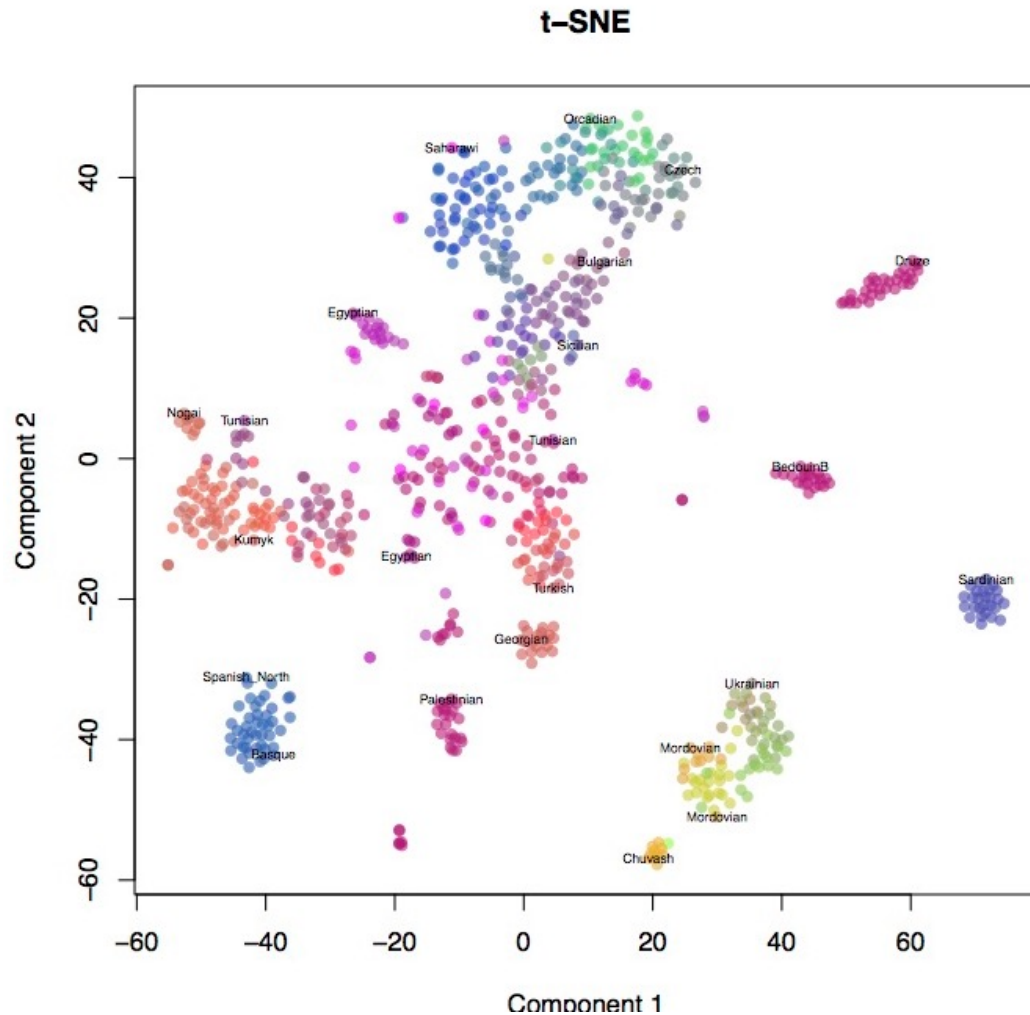
Dimensionality reduction

- Mapping data from a space of dimension p to a space of dimension $d \ll p$



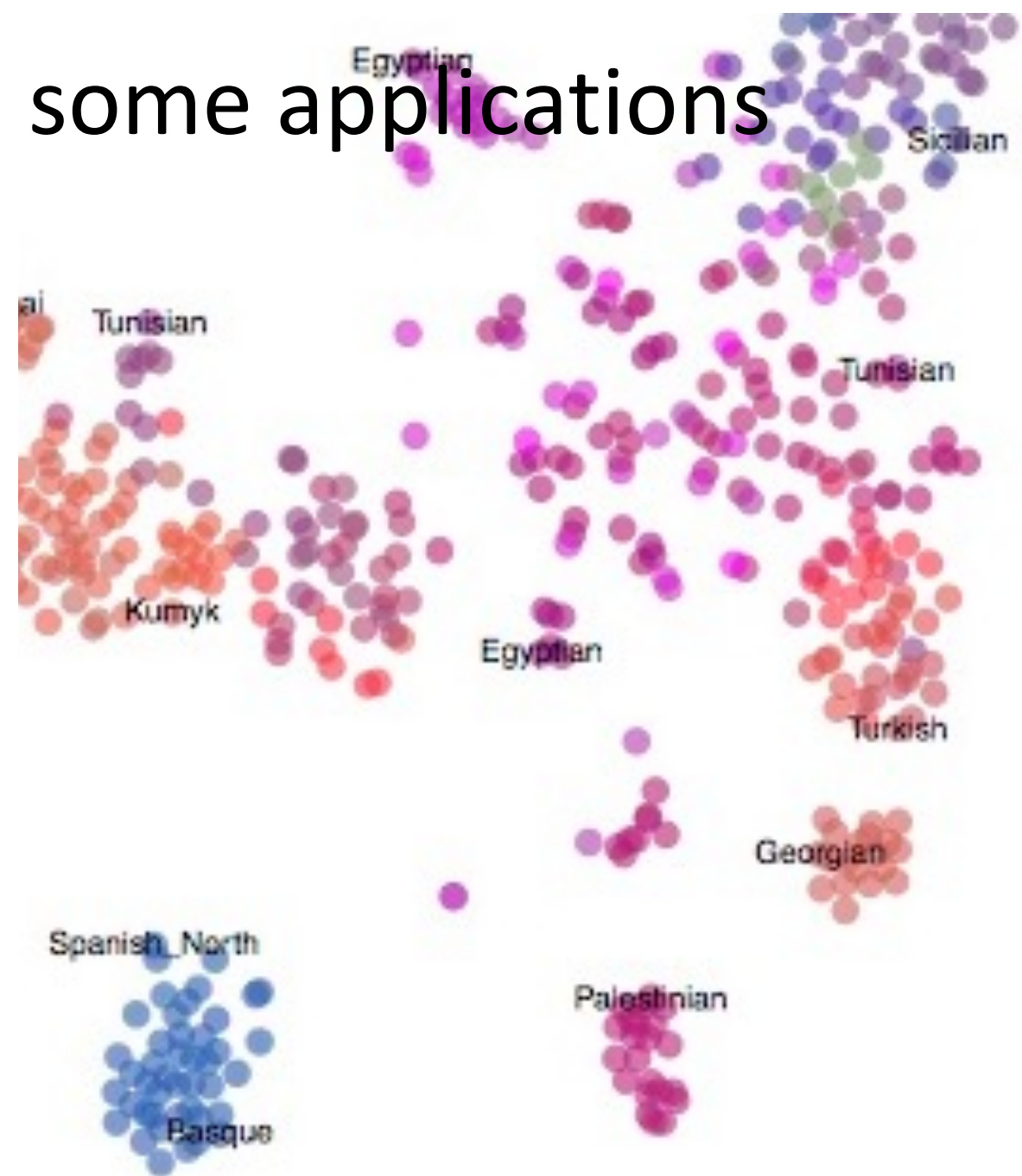
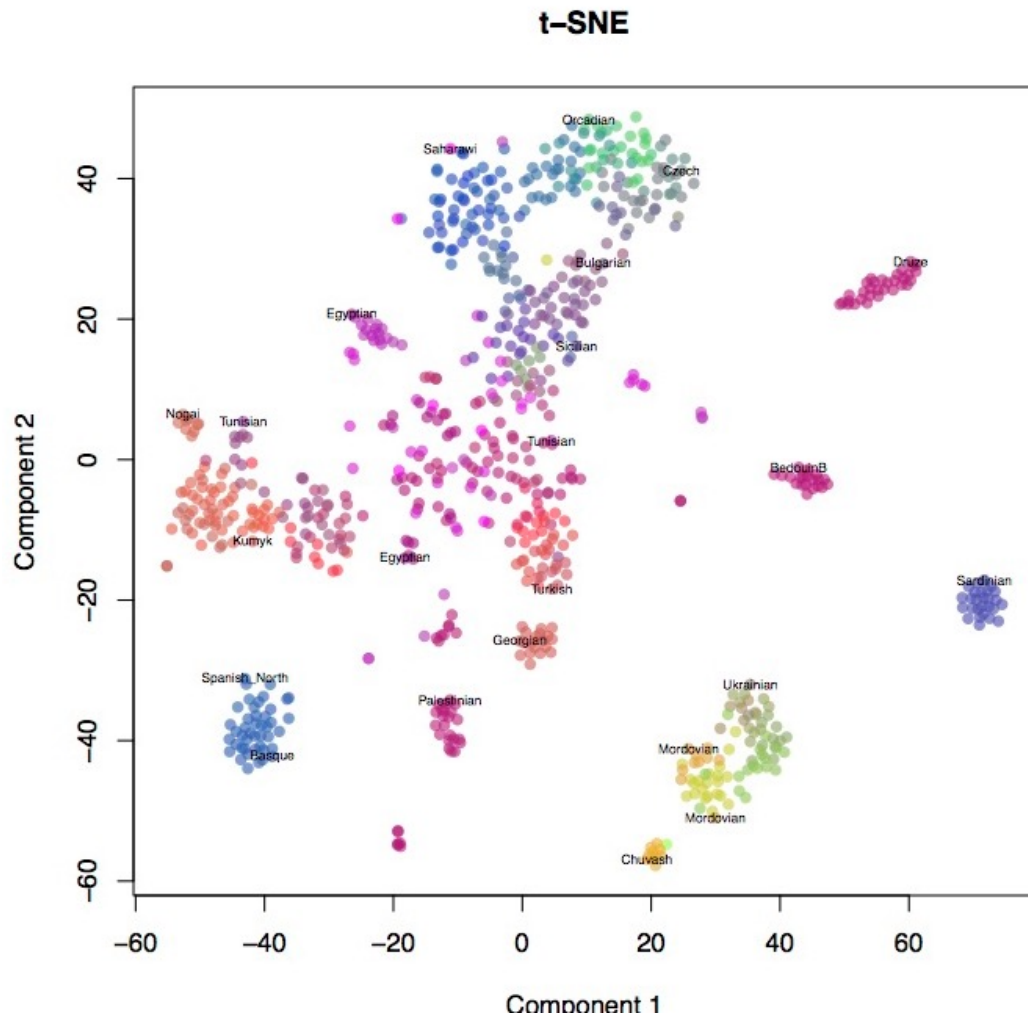
Dimensionality reduction: some applications

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



Dimensionality reduction : some applications

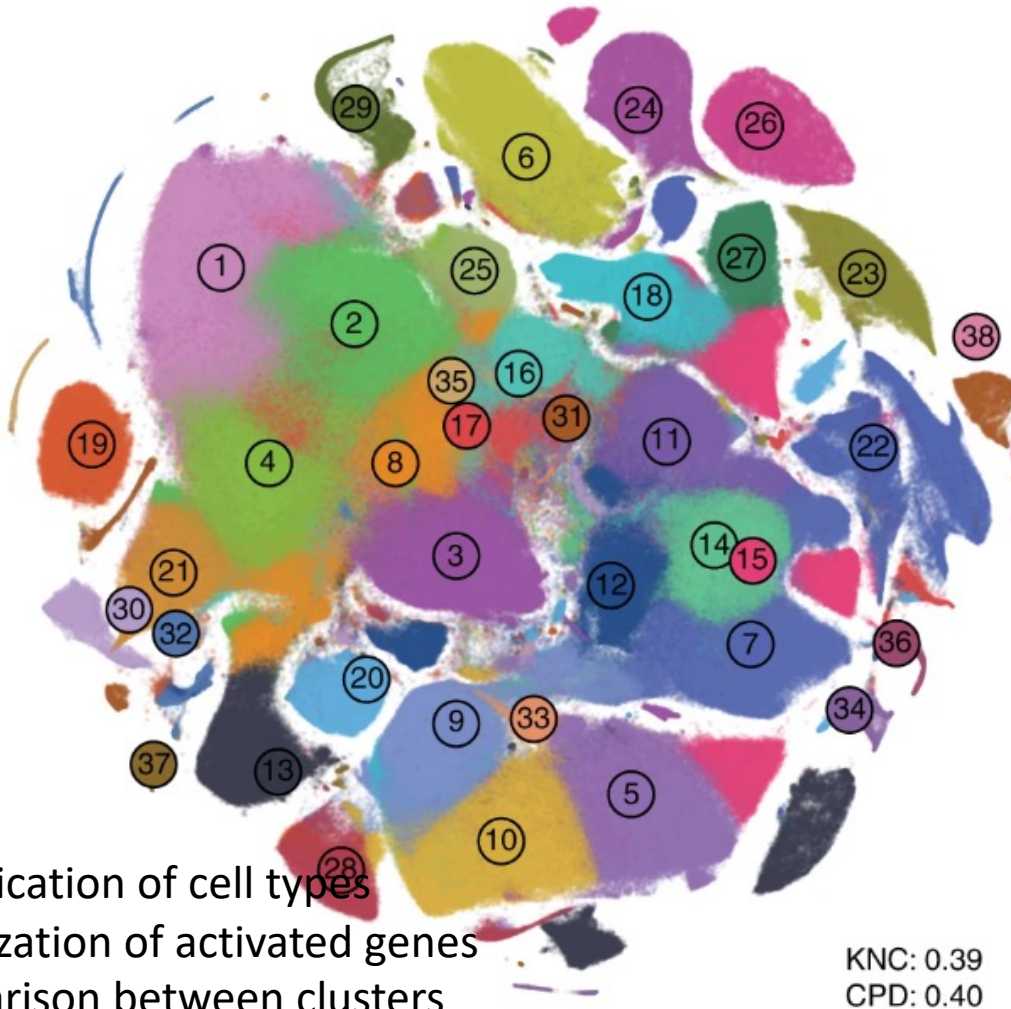
- SNPs (columns) from individual genomes (rows) reflect geographical structure



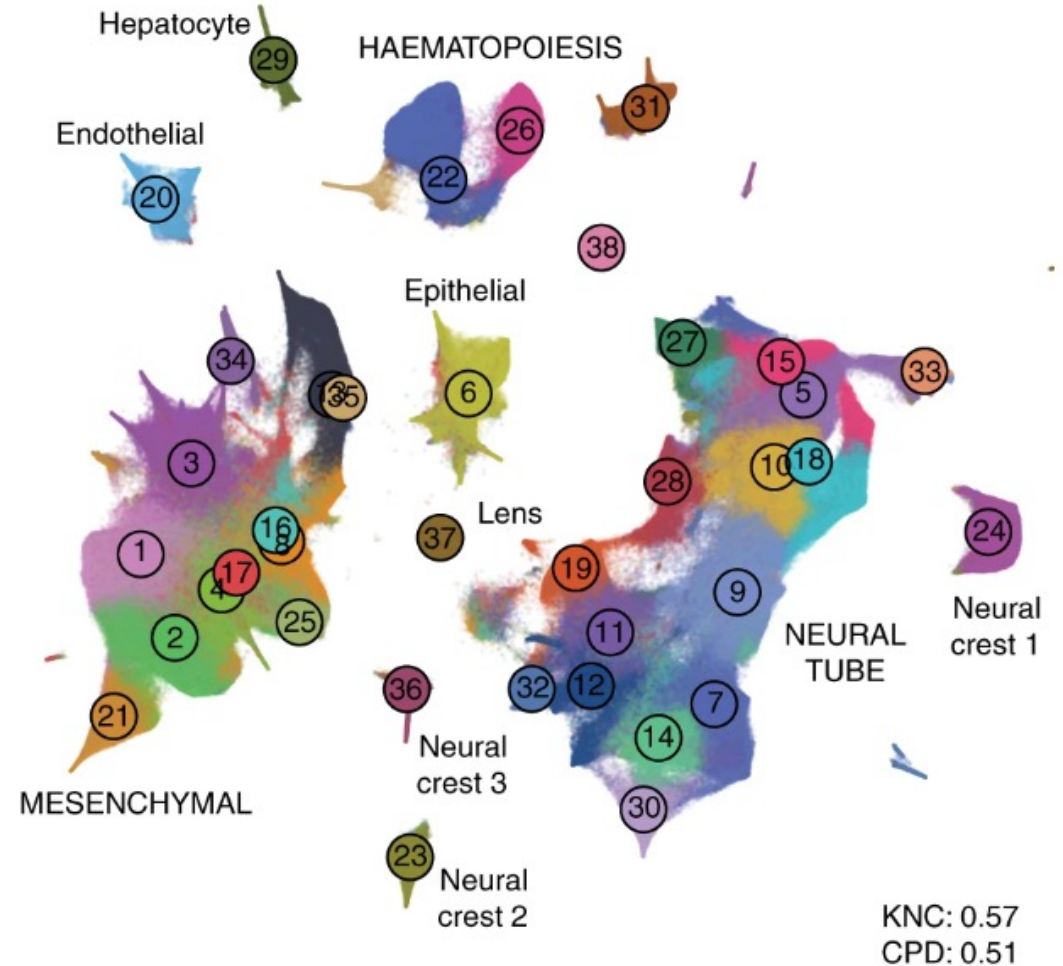
Dimensionality reduction: some applications

- Transcriptomics: 1.3 Million cells (rows) and ~30 000 genes (columns)

a Cao et al.⁸, $n = 2,058,652$. Original t-SNE



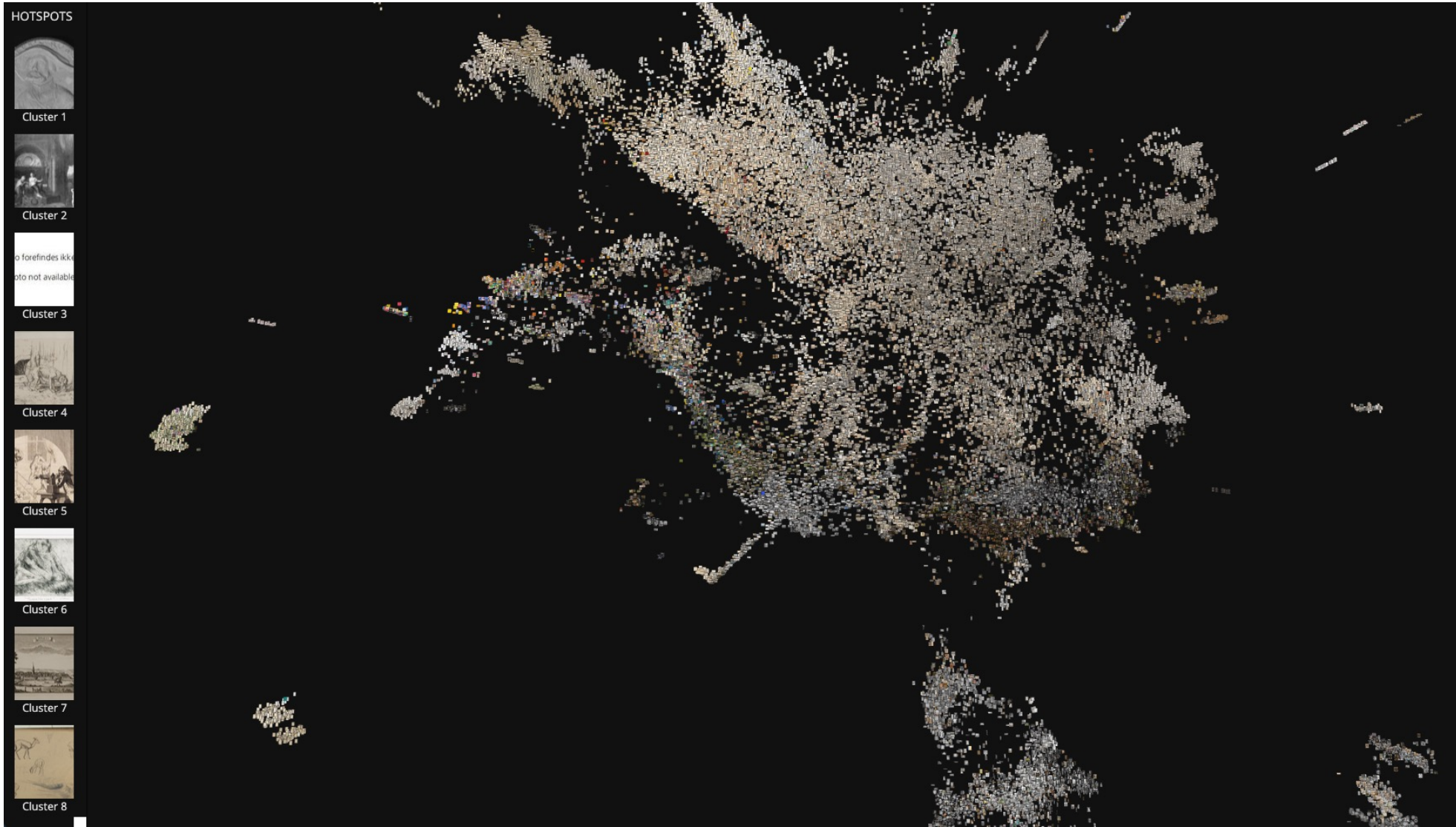
b Our t-SNE



- 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	

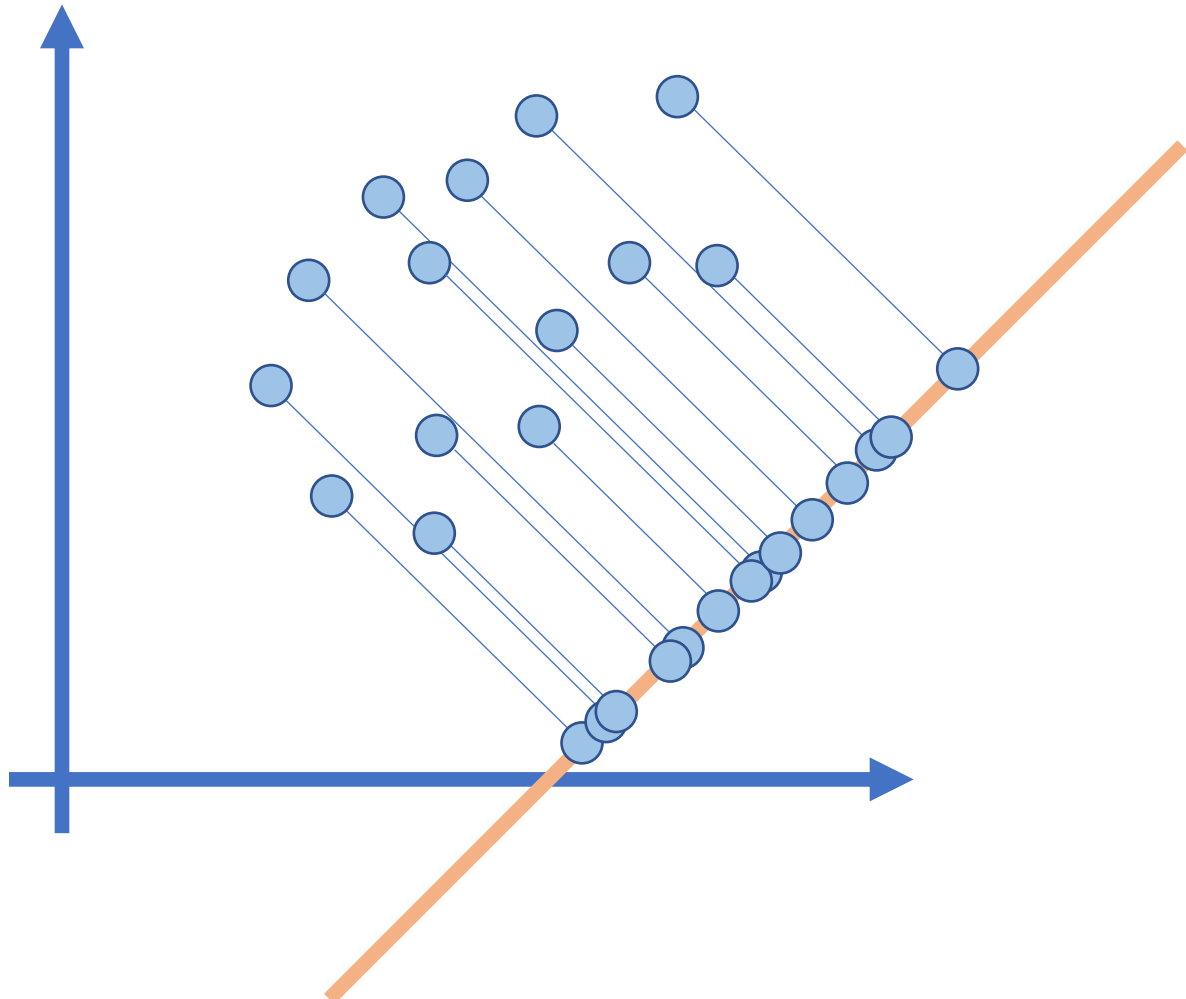
Dimensionality reduction: methods types

	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

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

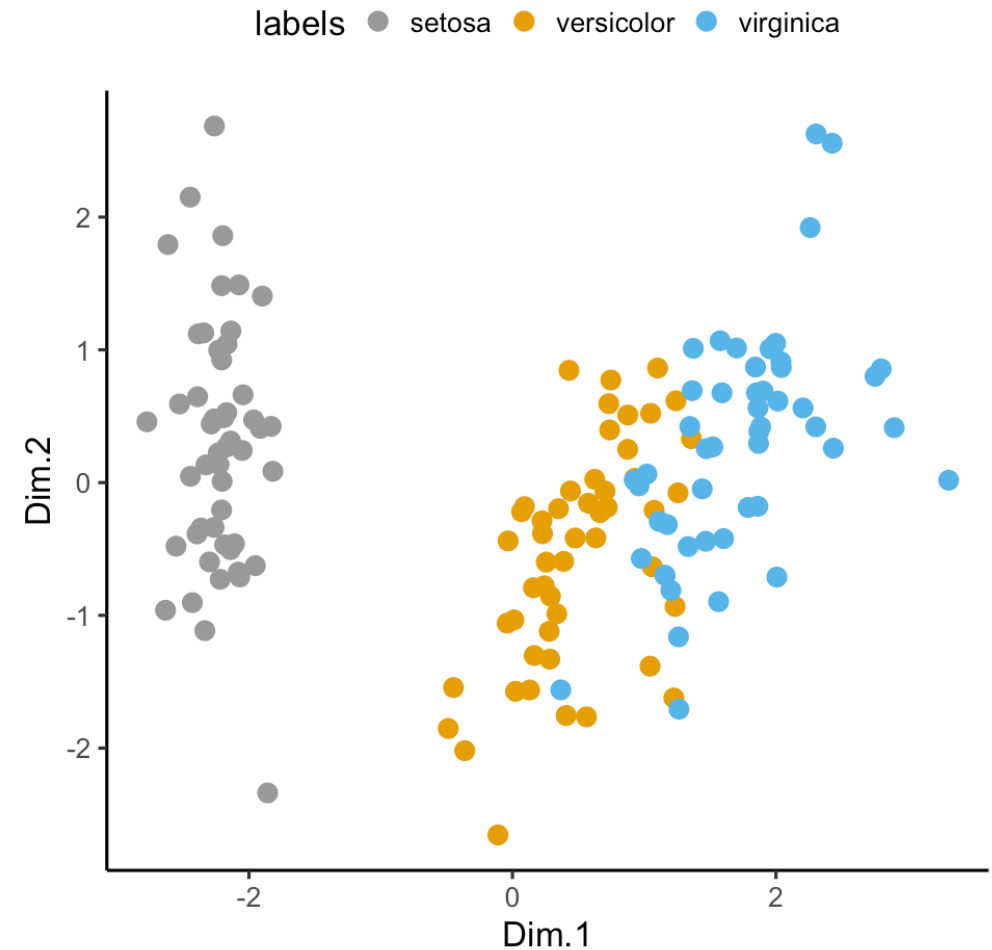
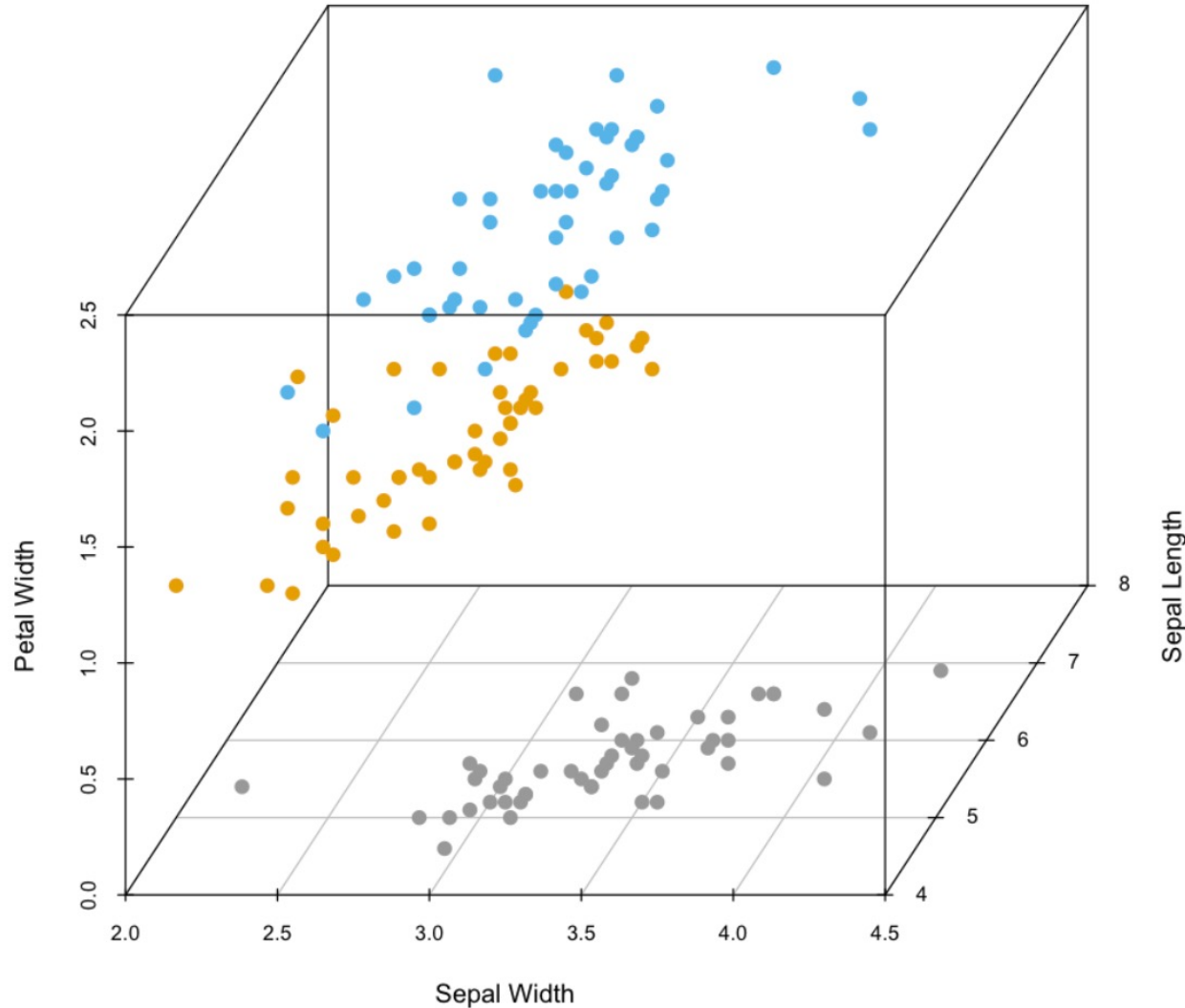
$$Y = XU = U\Sigma$$

minimizing reconstruction error

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

Dimensionality reduction: limits of linearity

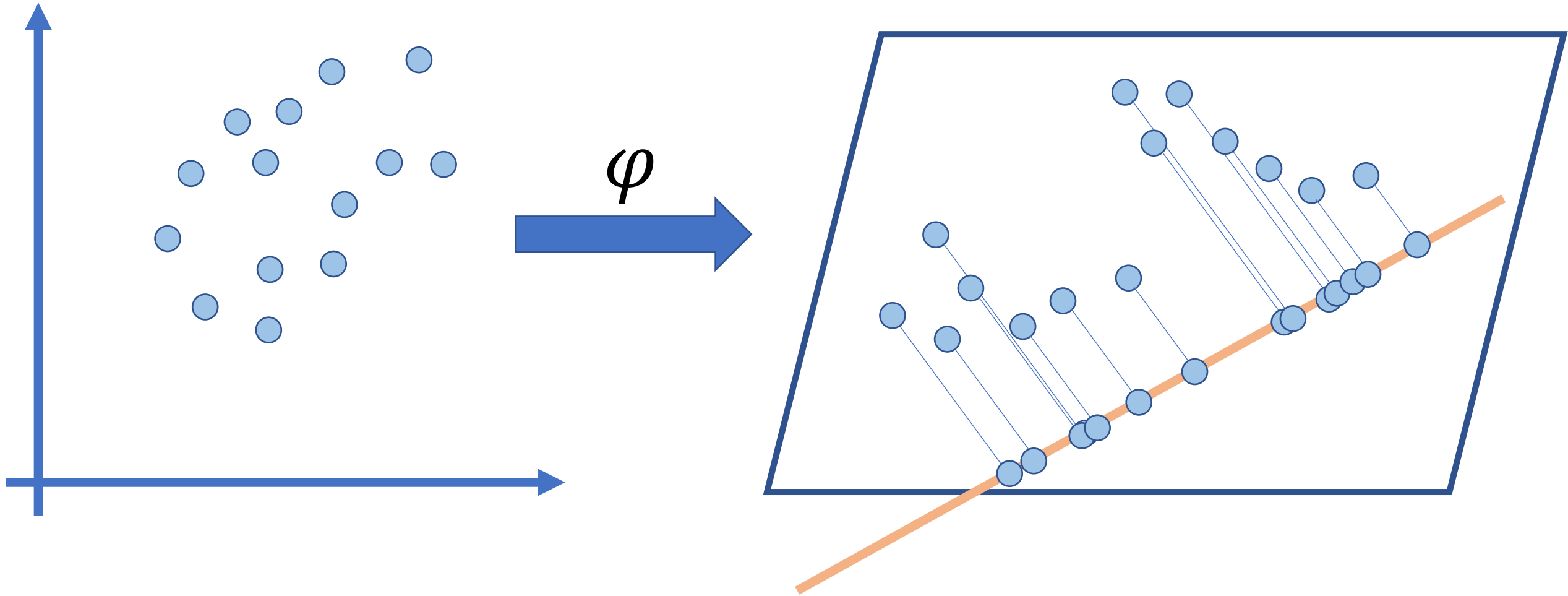
	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA



Dimensionality reduction: kernelPCA

	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

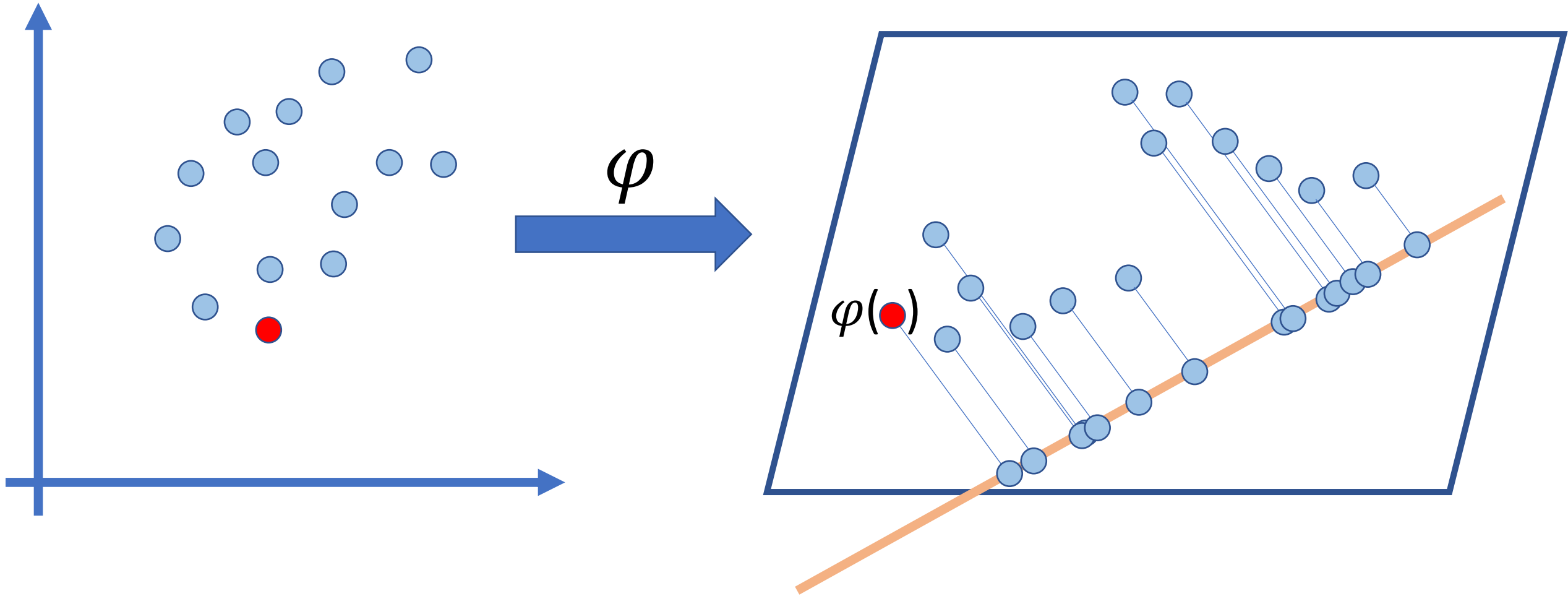
$$p=2 \rightarrow p' \rightarrow d=1$$



Dimensionality reduction: kernelPCA

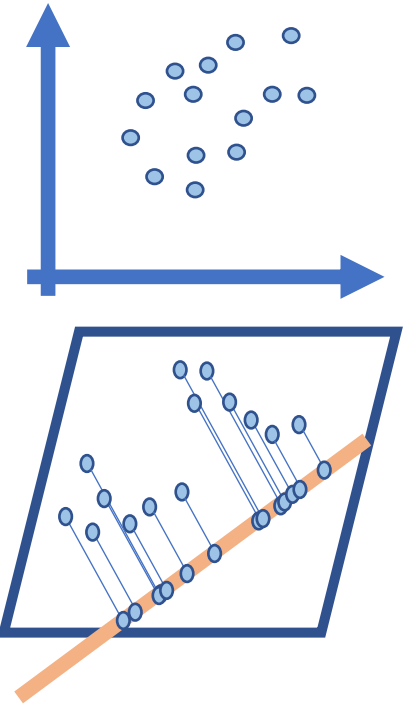
	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

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



Dimensionality reduction: kernelPCA

	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA



Analogous to PCA

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

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

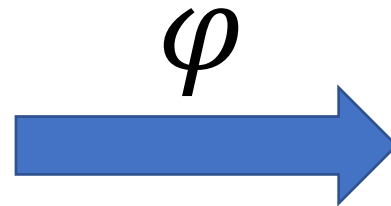
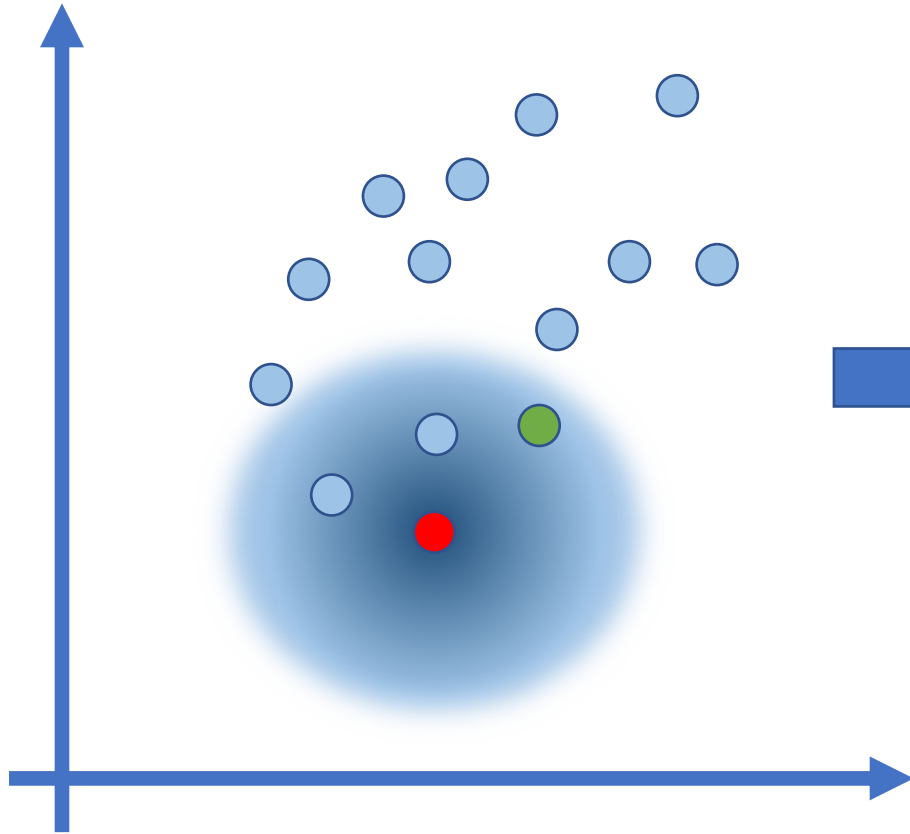
by assuming centered kernel φ and a centered kernel matrix

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

Dimensionality reduction: kernel trick

	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

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



$$[M]_{ij} = [\varphi(\mathbf{x}_i, \mathbf{x}_j)]$$

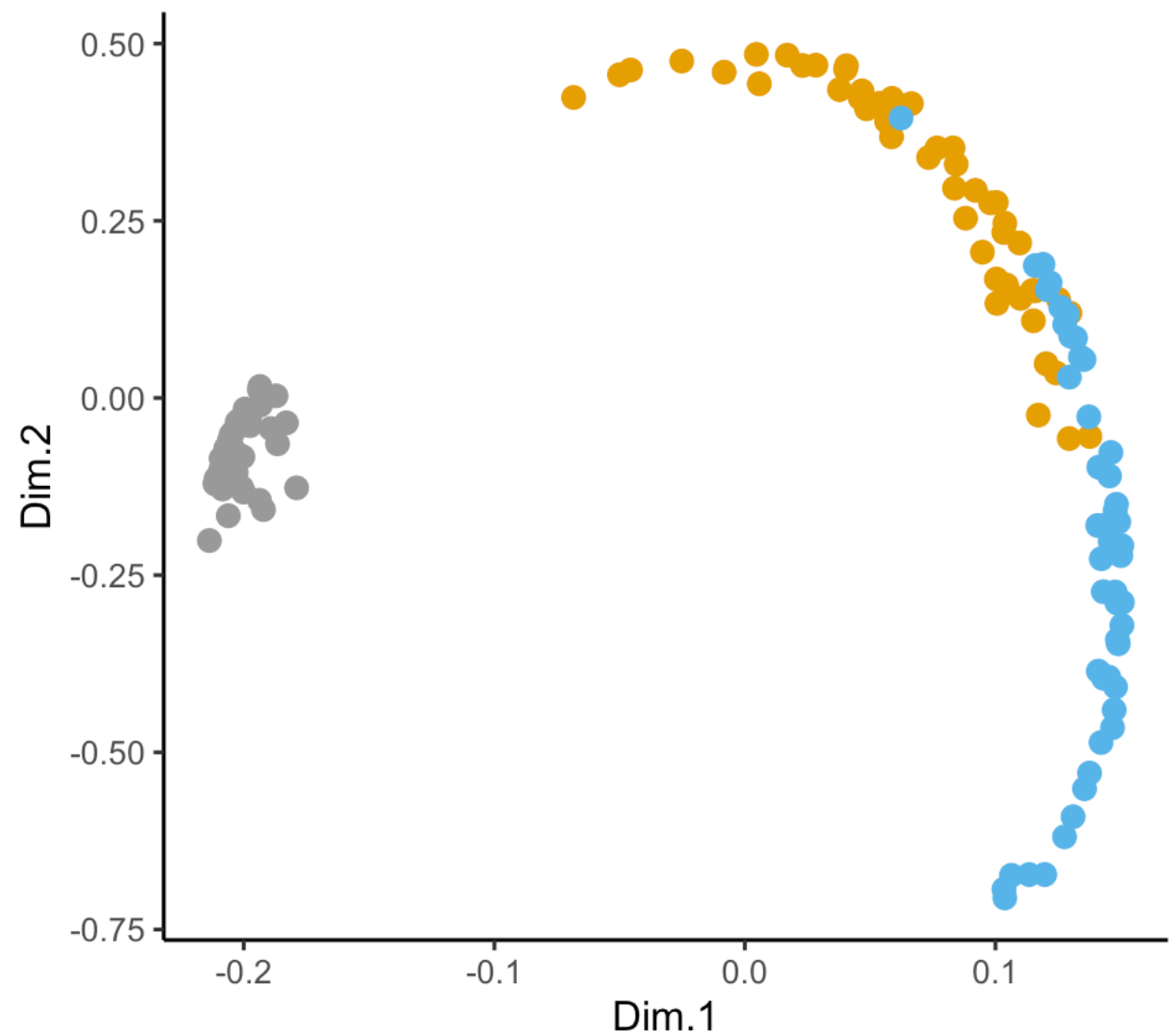
Avoids dot-products

Gaussian/RBF $\varphi(x_i, x_j) = \exp(-\|x_i - x_j\|^2 / 2\sigma^2)$ $\sigma \geq 0$
Polynomial $\varphi(x_i, x_j) = (x_i^T \cdot x_j + c)^d$ $c \geq 0, d \geq 1$
Linear $\varphi(x_i, x_j) = x_i^T \cdot x_j$
Laplacian $\varphi(x_i, x_j) = \exp(-\alpha \|x_i - x_j\|)$ $\alpha \geq 0$

.....

Dimensionality reduction: kernelPCA

labels ● setosa ● versicolor ● virginica

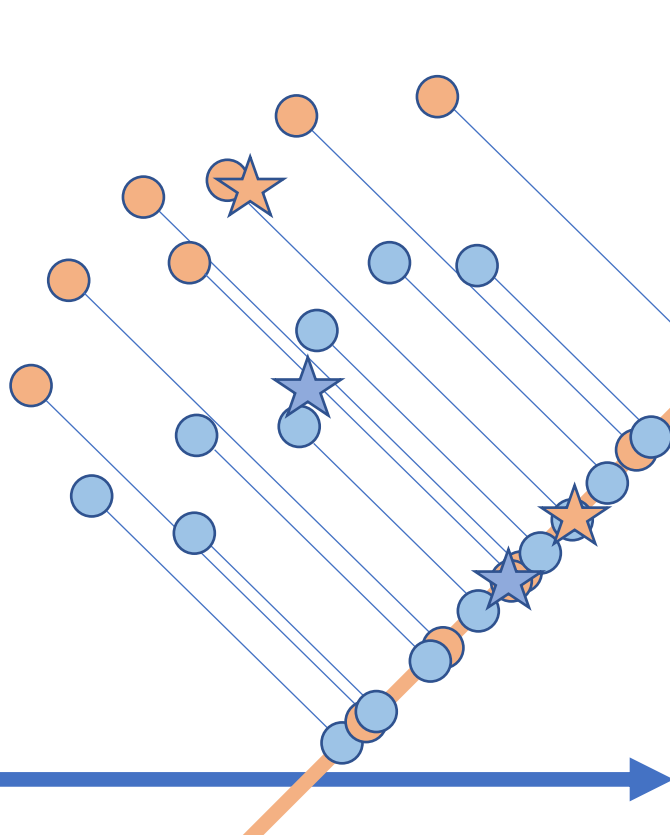


	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

Dimensionality reduction: LDA

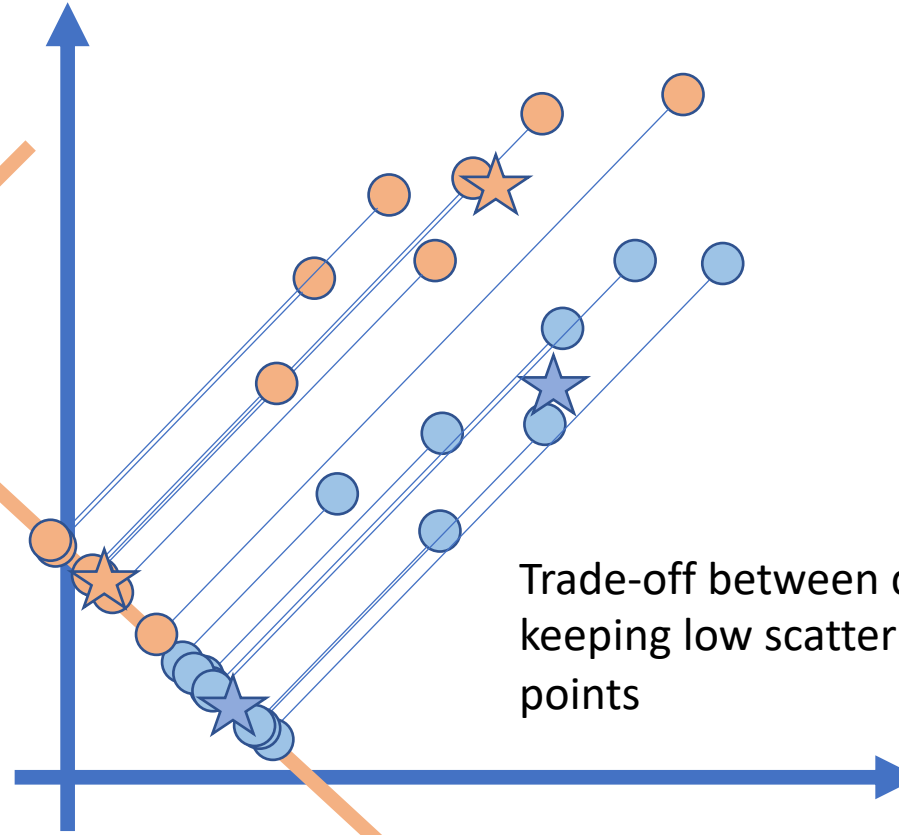
	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

$p=2 \rightarrow d=1$



(Unsupervised) PCA

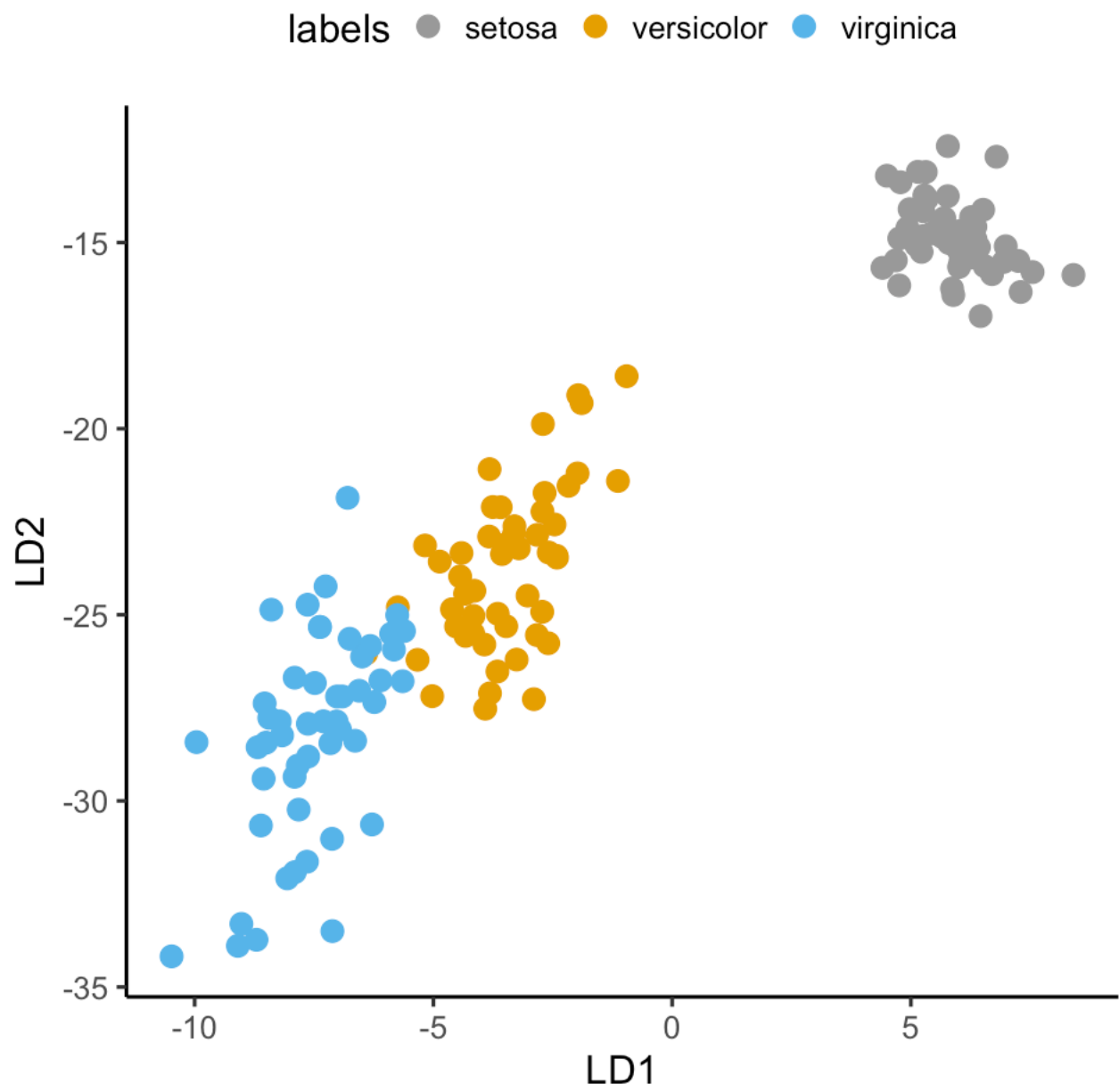
$p=2 \rightarrow d=1$



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

(Supervised) LDA

Dimensionality reduction: LDA

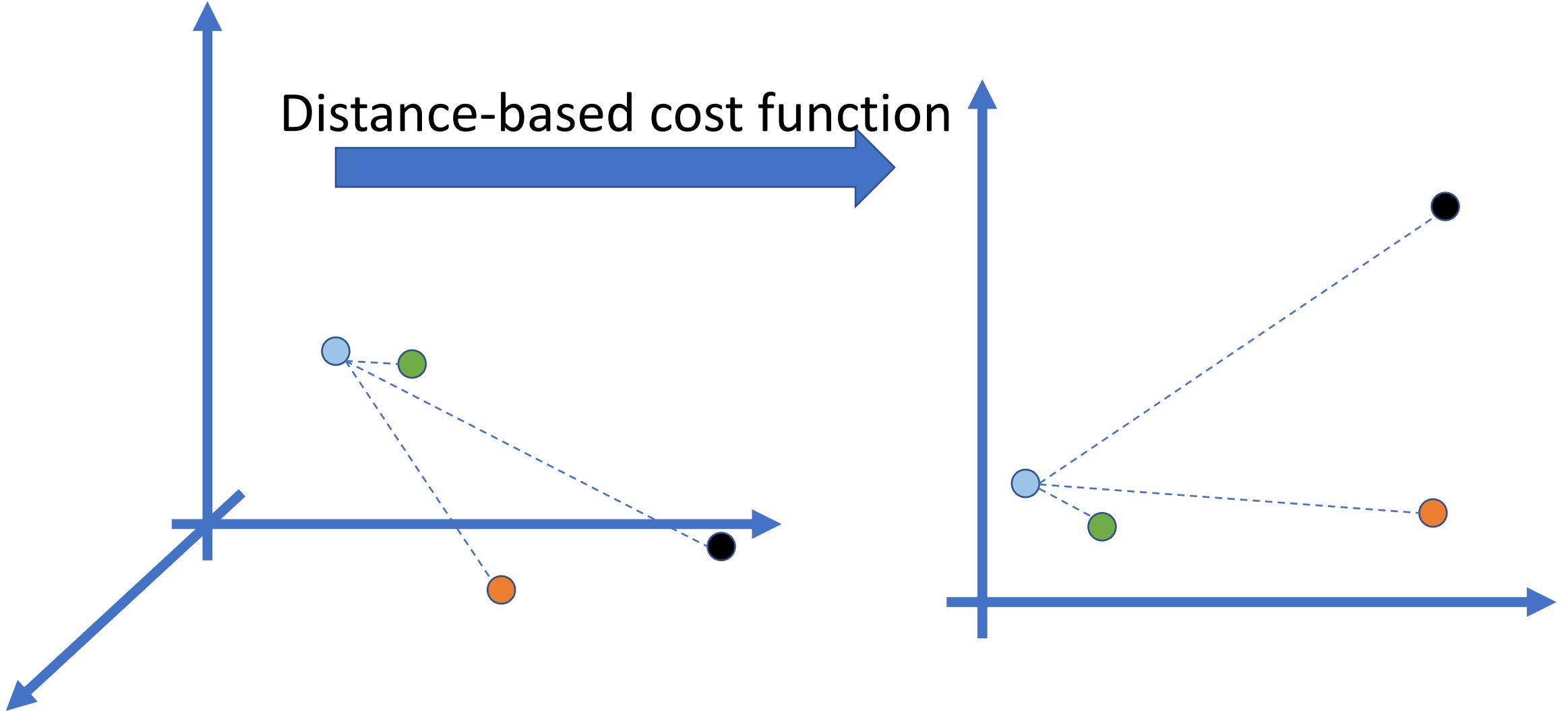


	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

Dimensionality reduction:

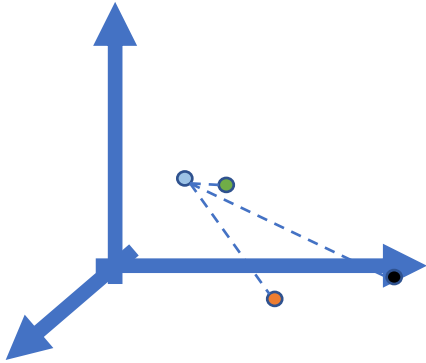
Non-parametric methods → MDS

Distance-based cost function

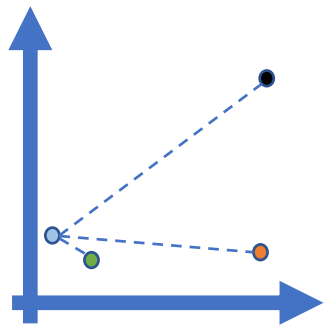


Dimensionality reduction:

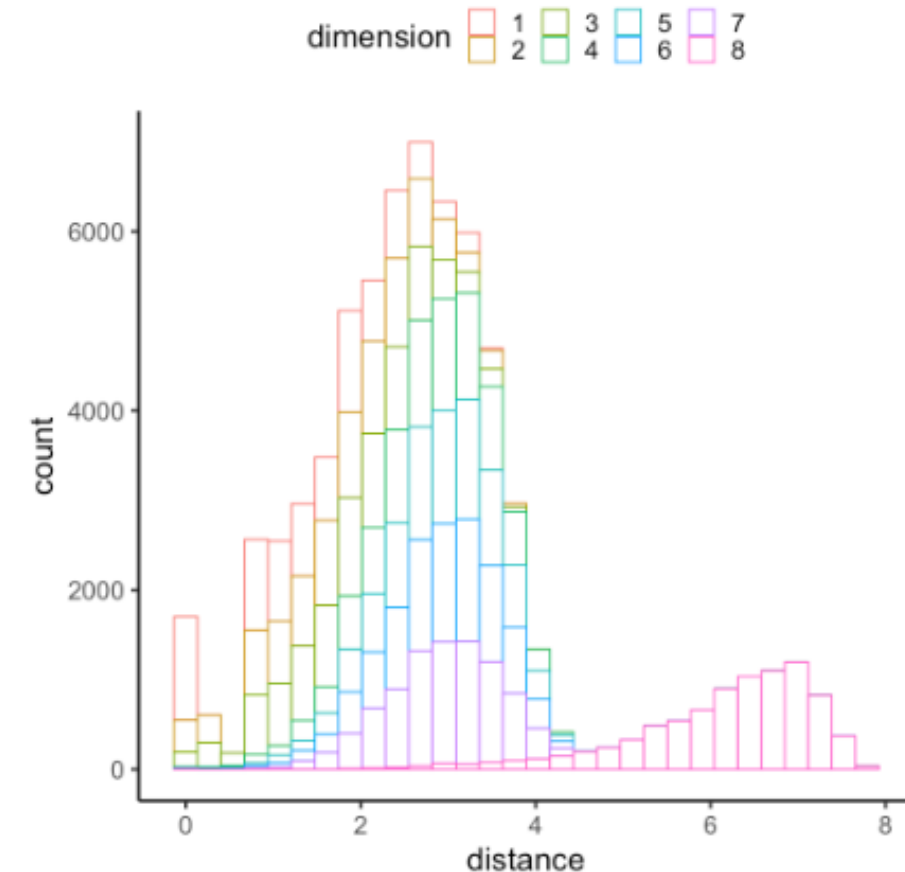
Non-parametric methods → MDS



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



	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA



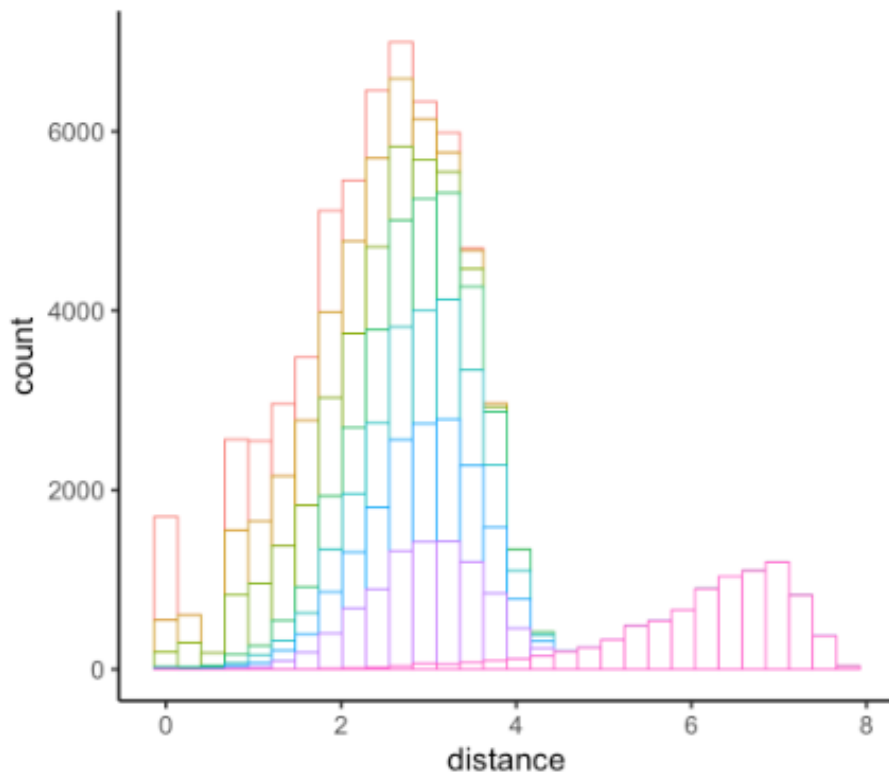
Dimensionality reduction:

Distances: curse of dimensionality

	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

dimension

1	3	5	7
2	4	6	8



Distances increase in higher dimensions.

Actually,

$$\forall \varepsilon > 0 : \lim_{d \rightarrow \infty} \mathbb{P} \left[\text{dist}_d \left(\frac{D_{\max_d} - D_{\min_d}}{D_{\min_d}}, 0 \right) \leq \varepsilon \right] = 1$$

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

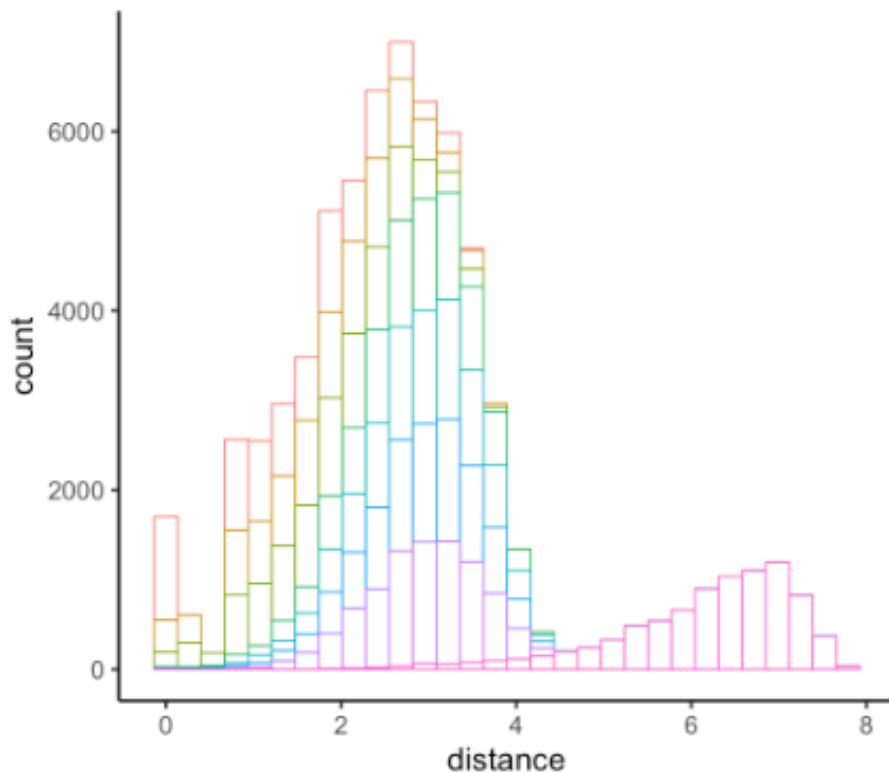
Dimensionality reduction:

Distances: curse of dimensionality

	L	Non-L
Unsup	PCA	kPCA
Sup	LDA	kLDA

dimension

1	3	5	7
2	4	6	8



Distances increase in higher dimensions.

Actually,

$$\forall \varepsilon > 0 : \lim_{d \rightarrow \infty} \mathbb{P} \left[\text{dist}_d \left(\frac{D_{\max_d} - D_{\min_d}}{D_{\min_d}}, 0 \right) \leq \varepsilon \right] = 1$$

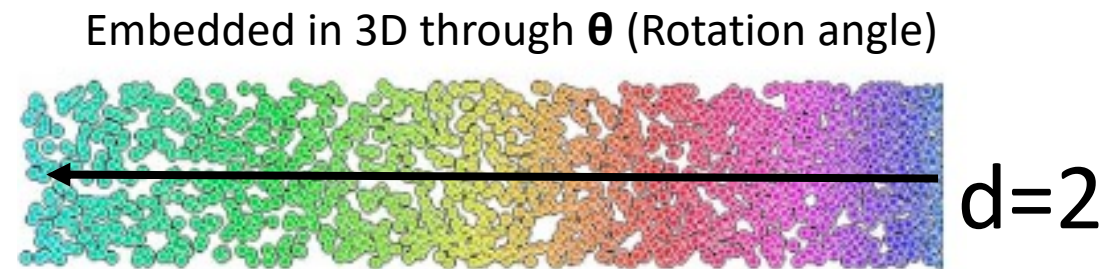
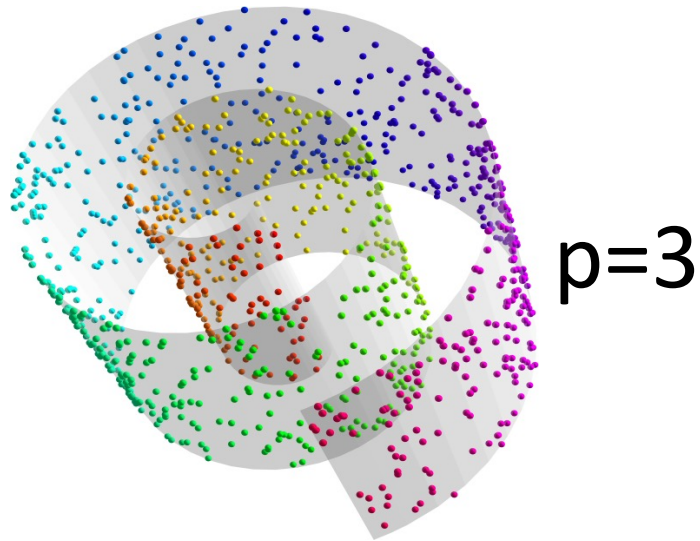
So distances **increase** and we have **no smaller distances** to preserve

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



Dimensionality reduction:

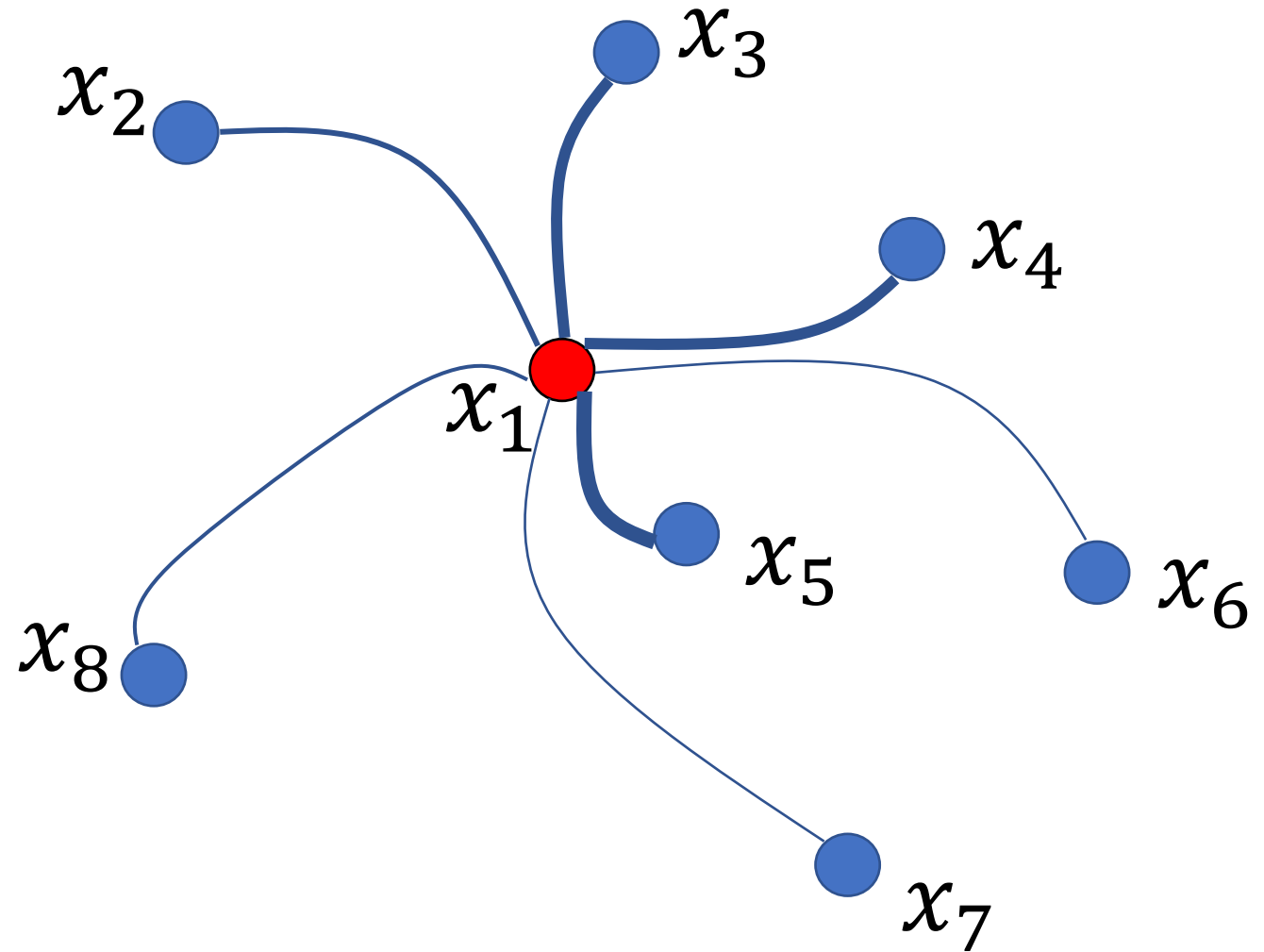
Distance preserving → Manifold learning

	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

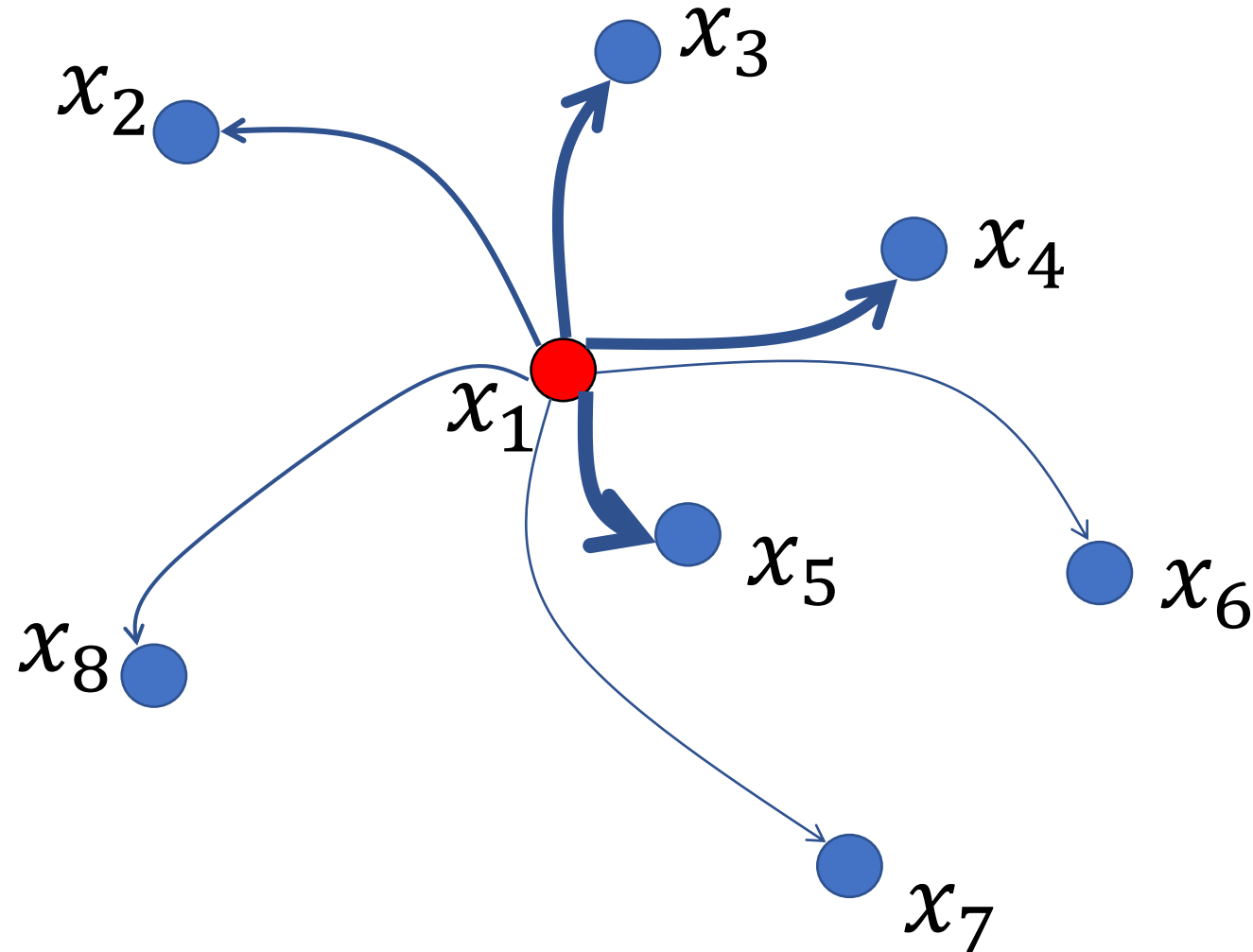
- Datapoints as **vertices** of a graph
- Pairwise **edges** are weighted by a distance function (local flatness)
- Graph algorithms are **well-known and implemented**



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: $k=7$.

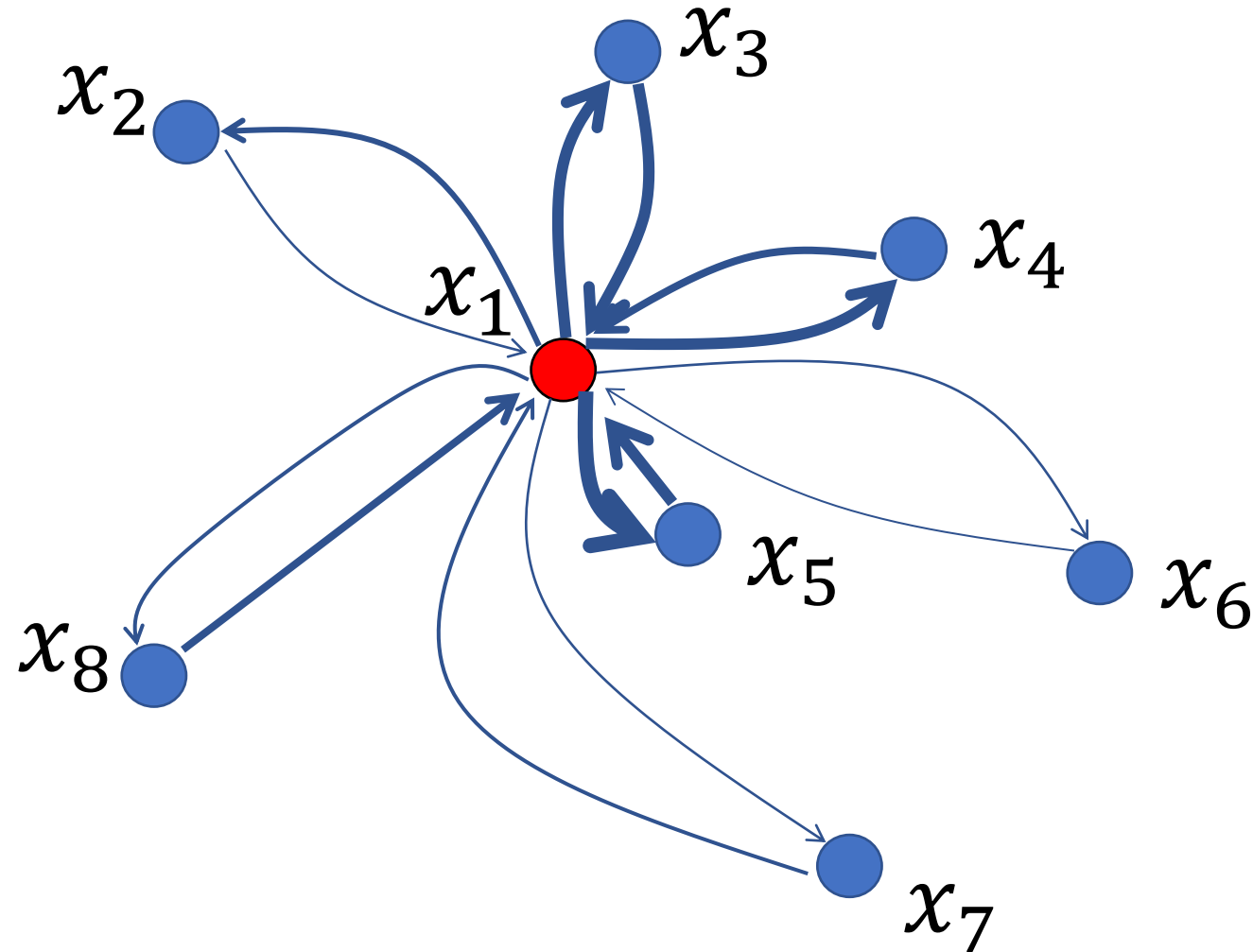
tSNE: Manifold description by graph

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



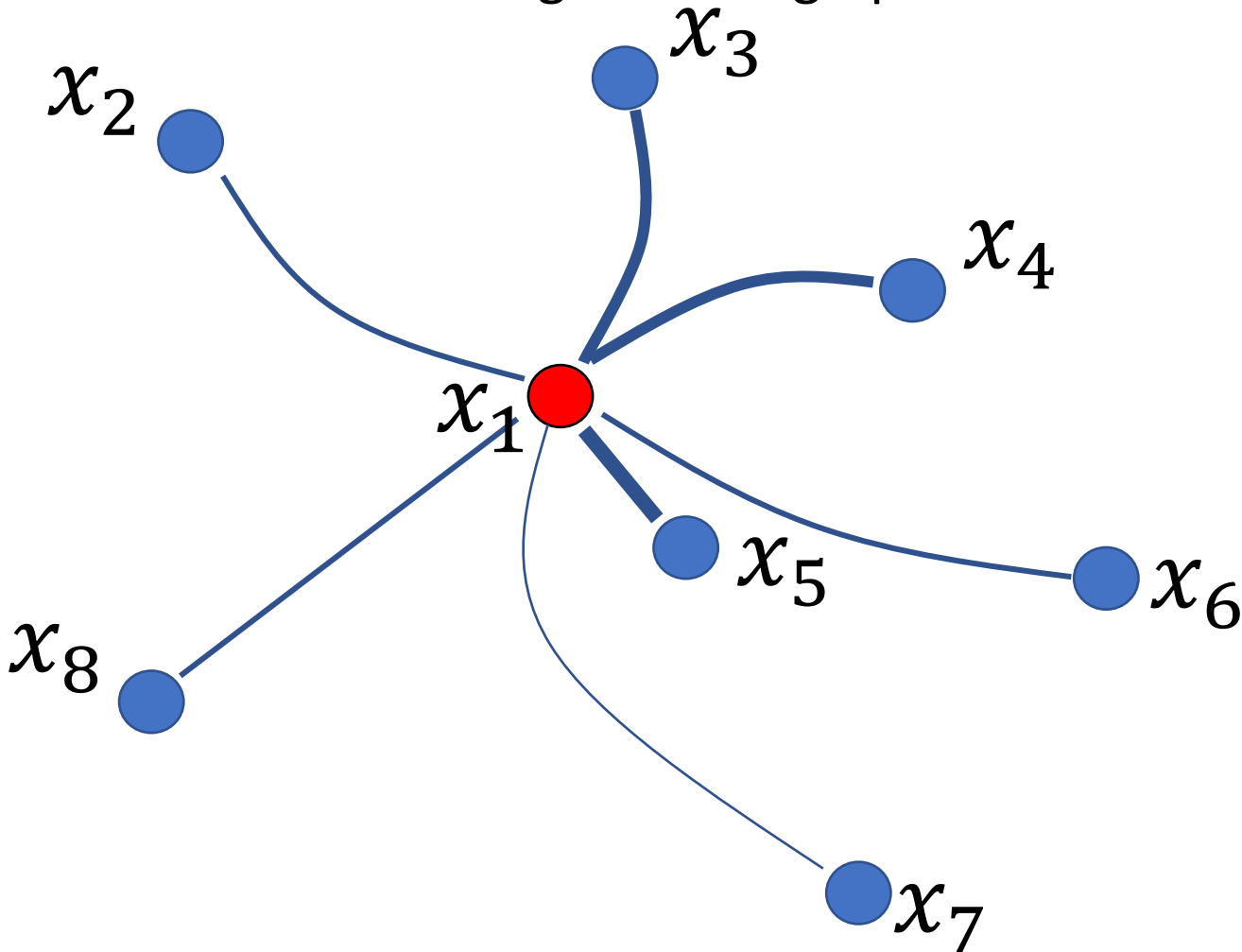
tSNE: Manifold description by graph

- 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



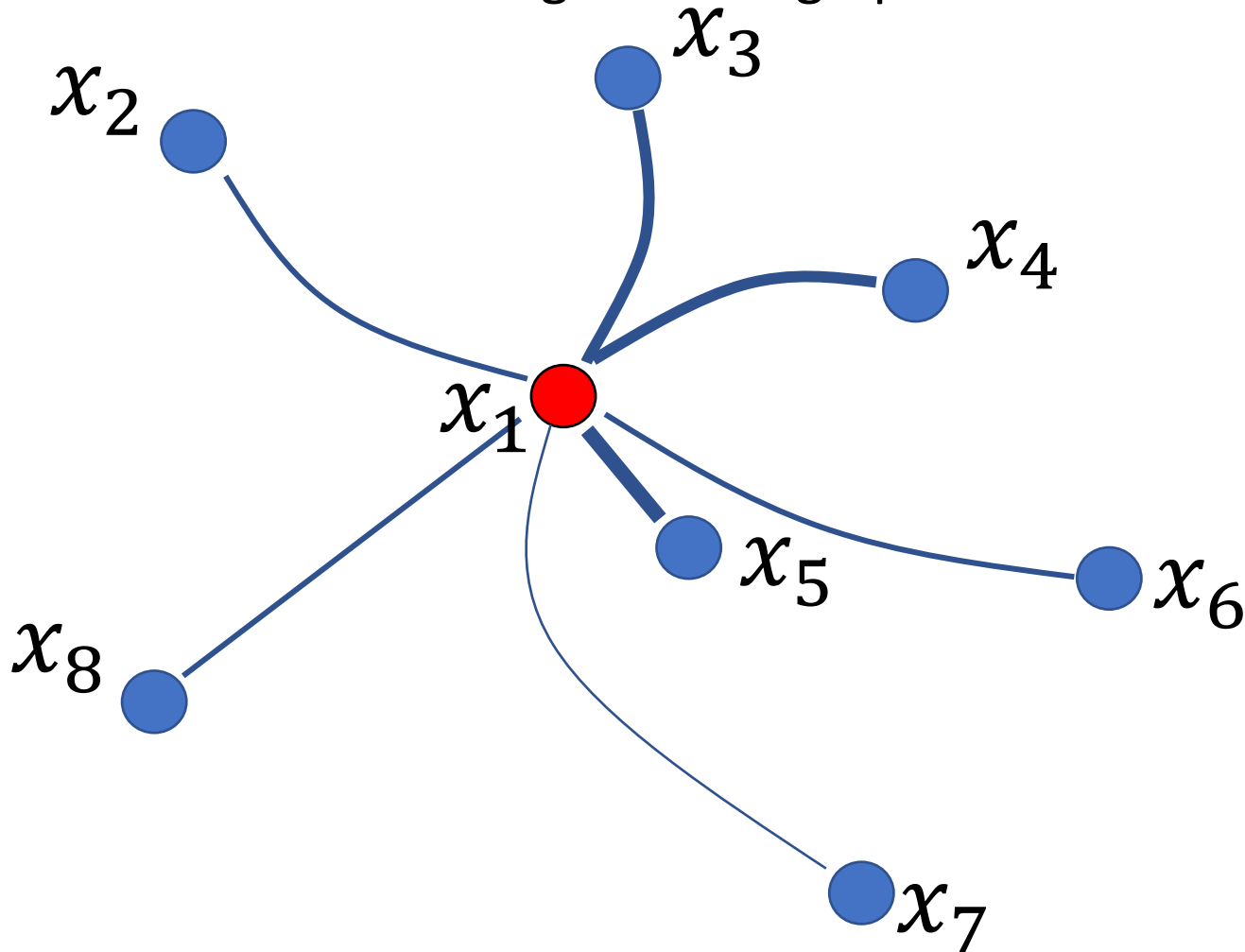
It corresponds to the formula

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

A procedure of the same type is done in the low-dimension ($d \ll D$) 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

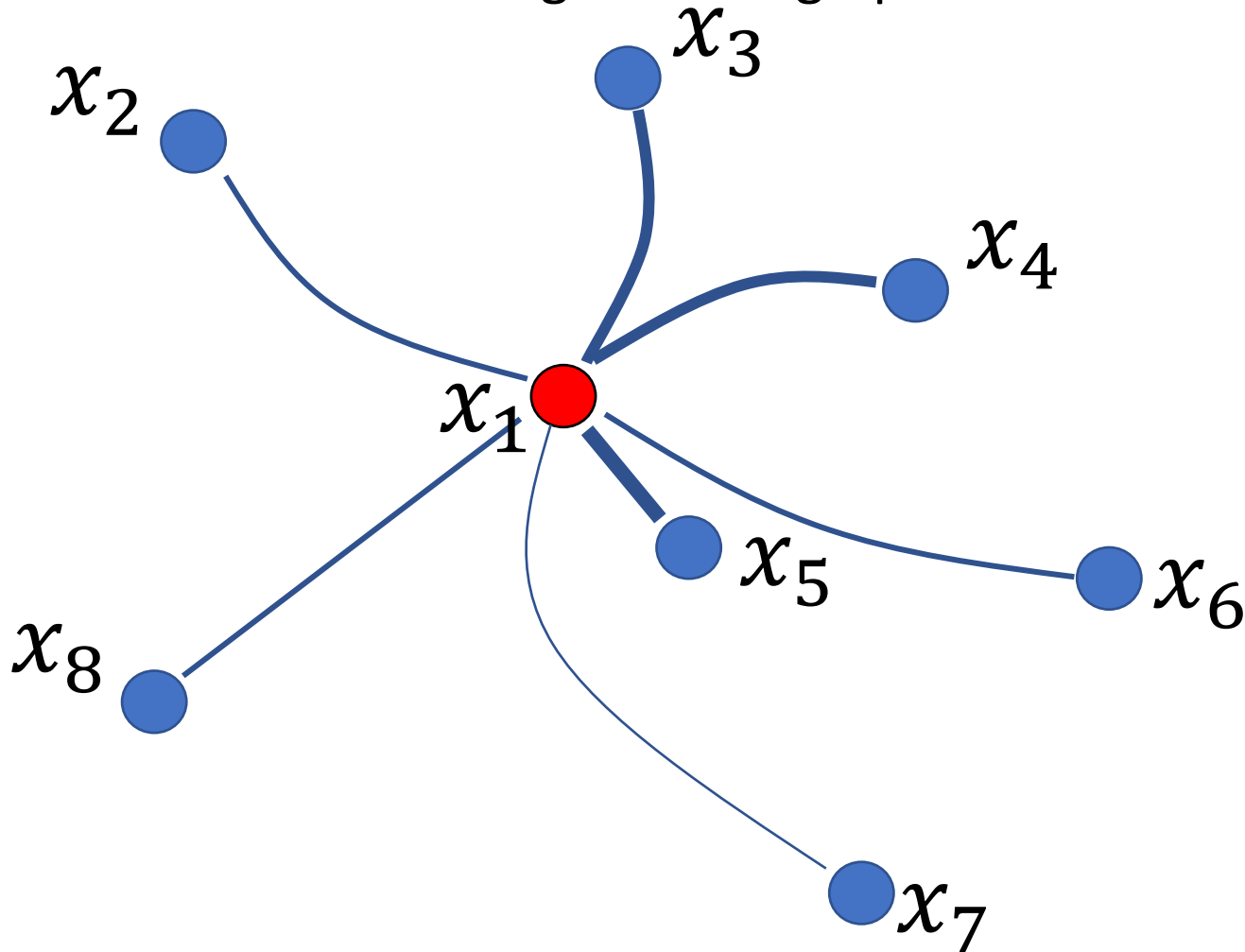
Mean of two
normalized edges
Between i and j

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

A procedure of the same type is done in the low-dimension ($d \ll p$) 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 ($d \ll p$) 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))
```

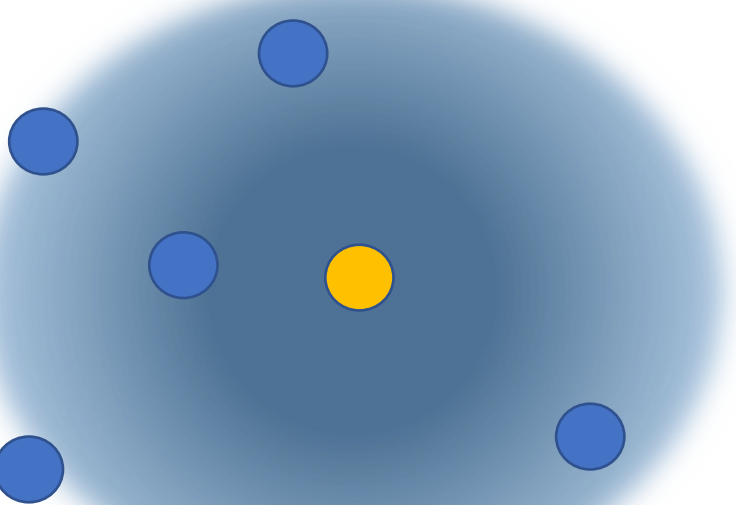
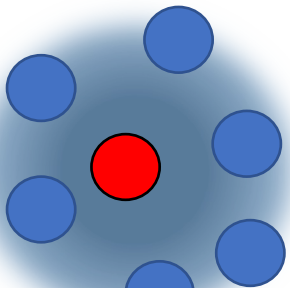
```
function(X, perplexity=15)  
{  
  
  D <- as.matrix(dist(X))^2  
  P <- matrix(0, nrow(X), nrow(X))  
  svals <- rep(1, nrow(X))  
  
  for (i in seq_along(svals))  
  {  
    srange <- c(0, 100)  
    tries <- 0  
  
    for(j in seq_len(50))  
    {  
      Pji <- exp(-D[i, -i] / (2 * svals[i]))  
      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

$H \leftarrow -1 * P_{ji} \%*\% \log(P_{ji}, 2)$

If H is less than the log-perplexity,
increase sigma, otherwise reduce it
if ($H < \log(\text{perplexity}, 2)$){
..... }



```
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

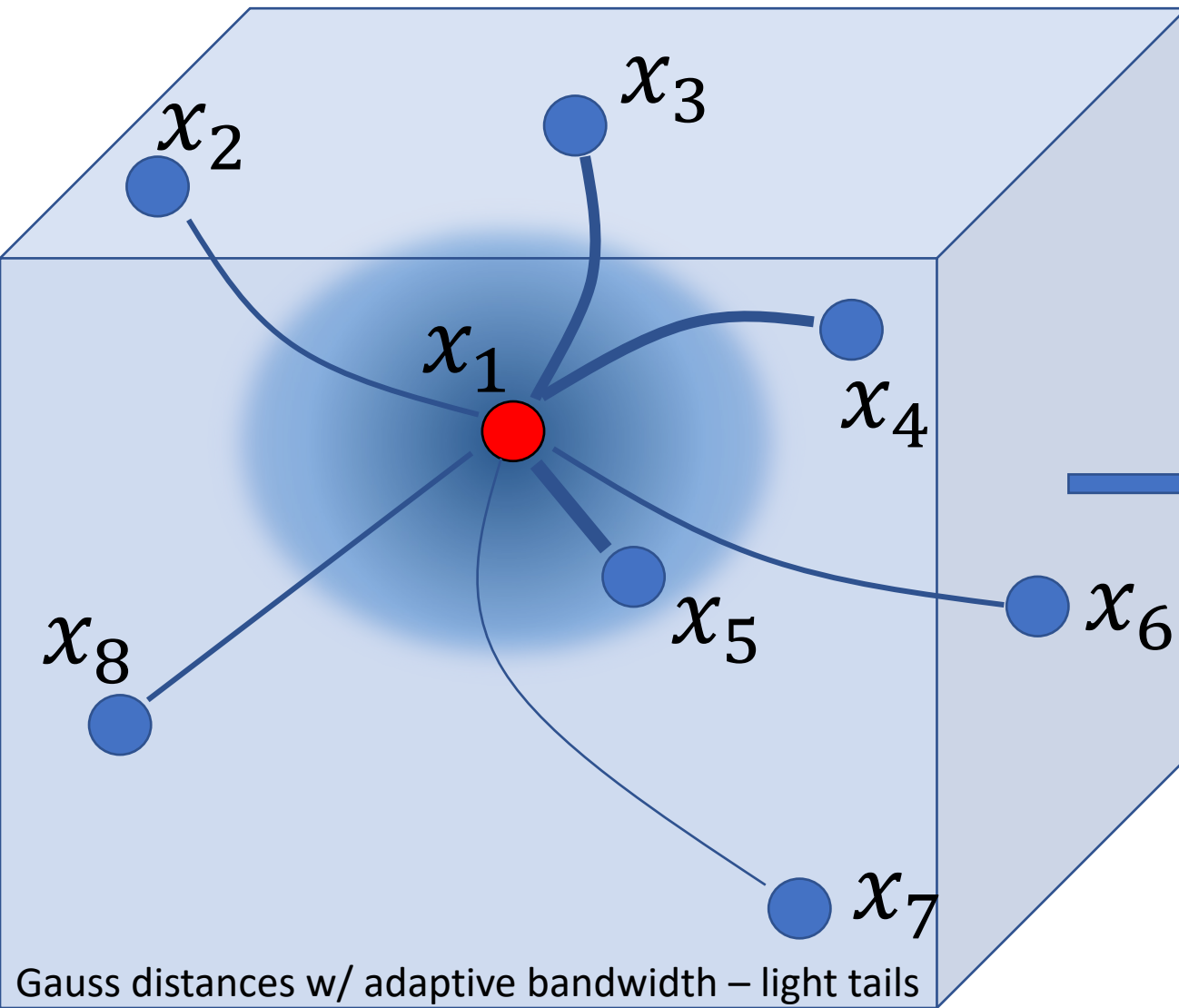
    for(j in seq_len(50))
    {
      Pji <- exp(-D[i, -i] / (2 * svals[i]))
      Pji <- Pji / sum(Pji)
      H <- -1 * Pji \%*\% log(Pji, 2)

      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
  }

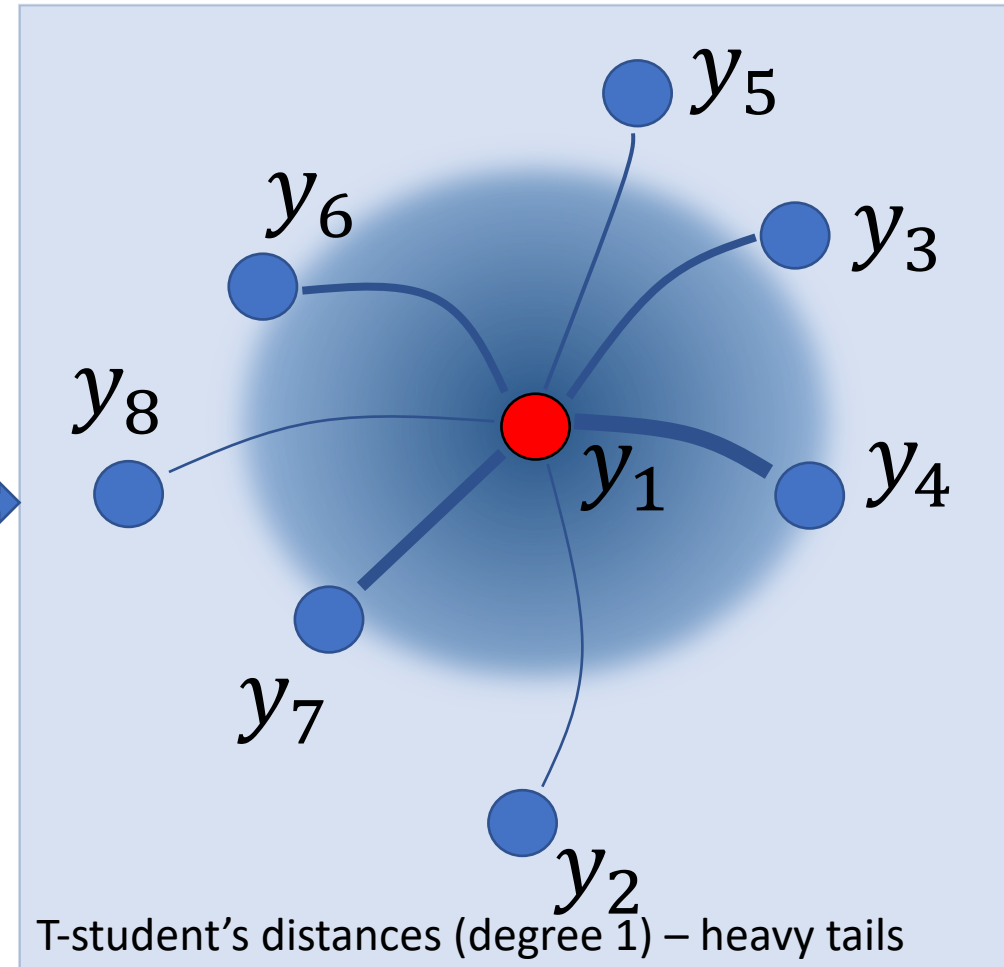
  return(0.5 * (P + t(P)) / sum(P))
}
```

tSNE: optimization

In dimension D the edges are **fixed**



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

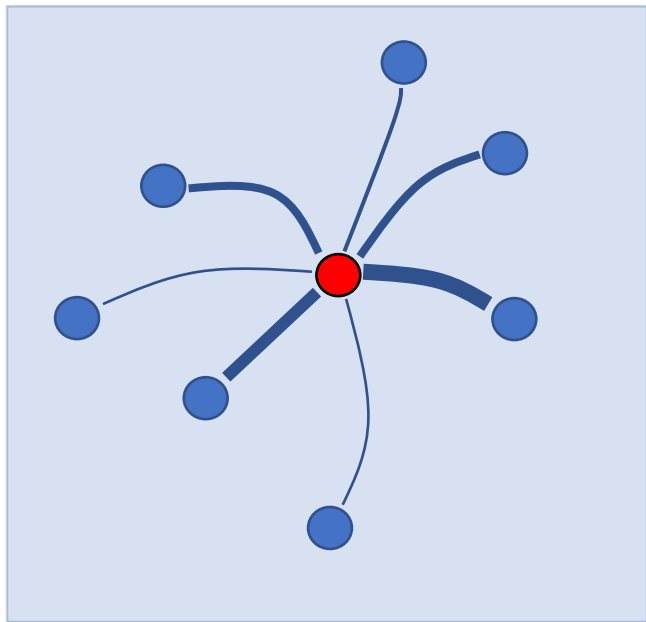
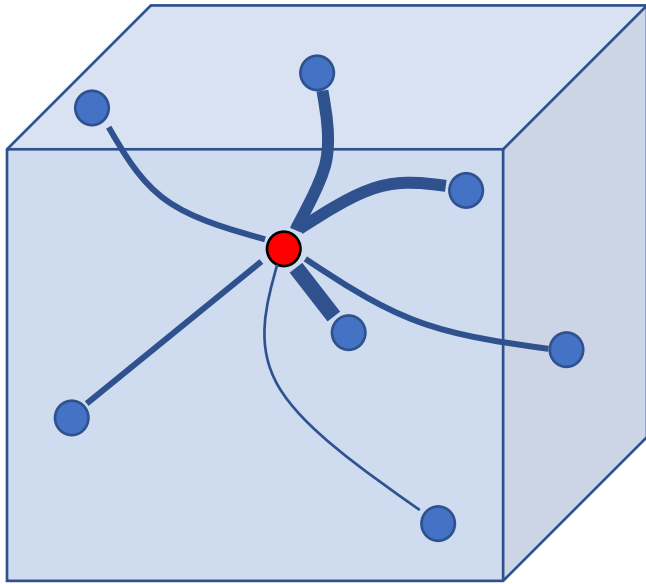


tSNE: optimization

Cost function (KL divergence)

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

- 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

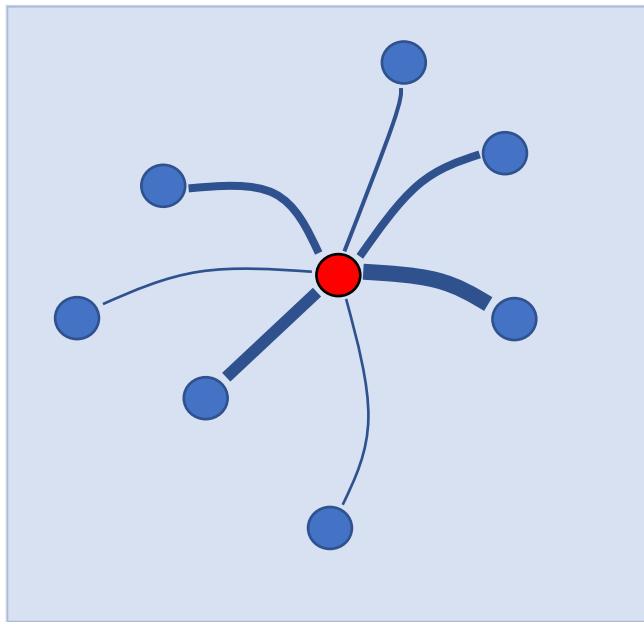
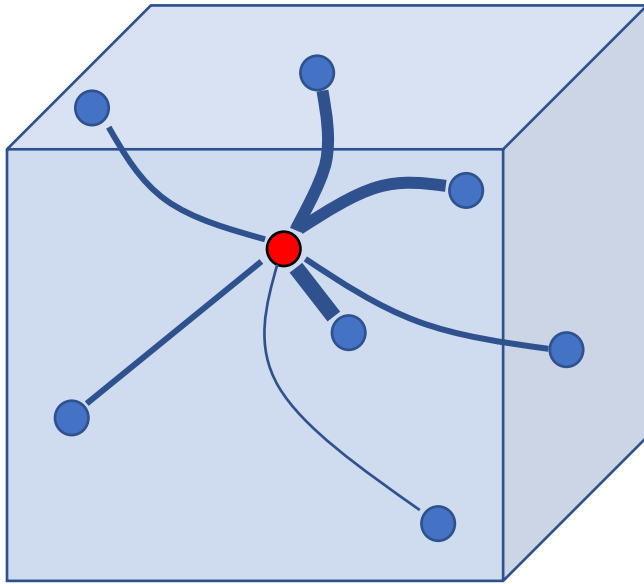


tSNE: optimization

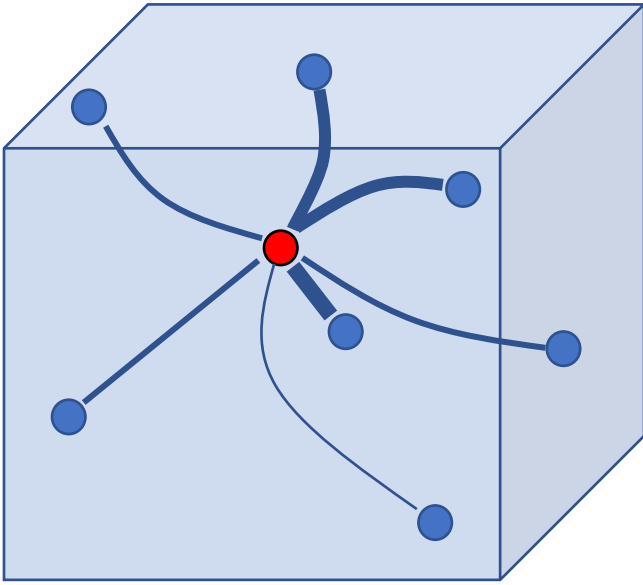
Cost function (KL divergence)

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

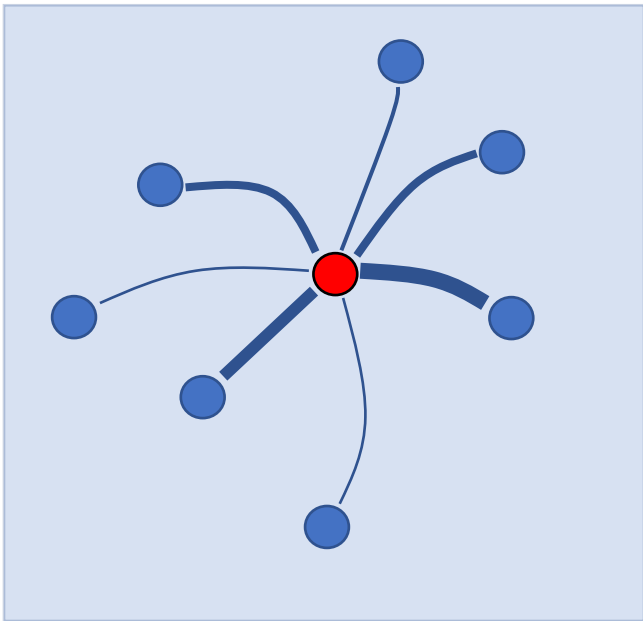
- The p 's are all fixed (D-dimensional space)
- To make C small (ideally zero), you want q 's similar to p 's
- Normalization helps avoiding clusters of packed points
- tSNE mostly cares about how points are distributed locally even though it uses a t-student distribution



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



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

tSNE: optimization

The gradient of C is

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

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

$$4 \left(\sum_{j \neq i} p_{ij} q_{ij} Z (y_i - y_j) - \sum_{j \neq i} q_{ij}^2 Z (y_i - y_j) \right)$$

$$Z = \sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}$$

Why?

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

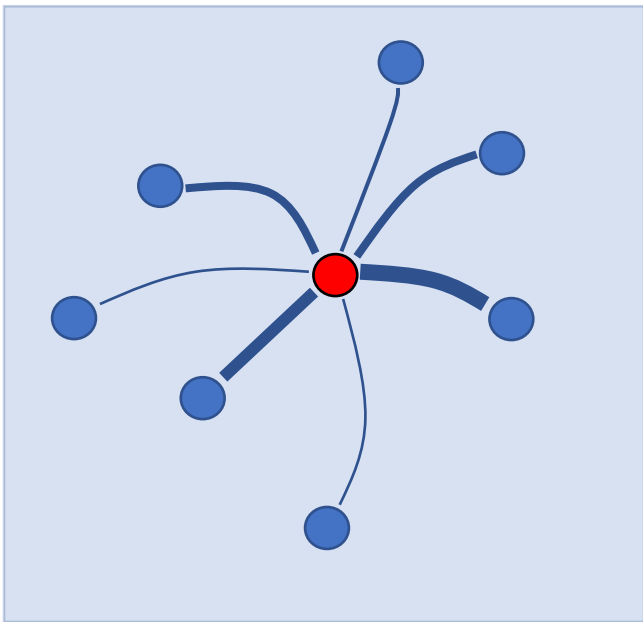
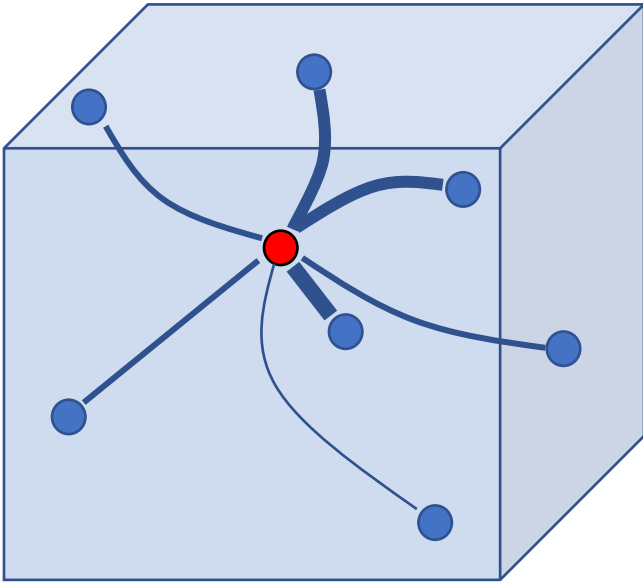
With $d_{ij} = \|y_i - y_j\|_2^2$
helps in doing the derivatives.

[Look here.](#)

tSNE: optimization

Optimization done with gradient descent. The gradient of C is

$$4 \left(\underbrace{\sum_{j \neq i} p_{ij} q_{ij} Z(\mathbf{y}_i - \mathbf{y}_j)}_{\text{Attractive forces P: keep close points together}} - \underbrace{\sum_{j \neq i} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)}_{\text{Repulsive forces Q: push points away from each other}} \right)$$



tSNE: optimization

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

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

- 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  
}
```

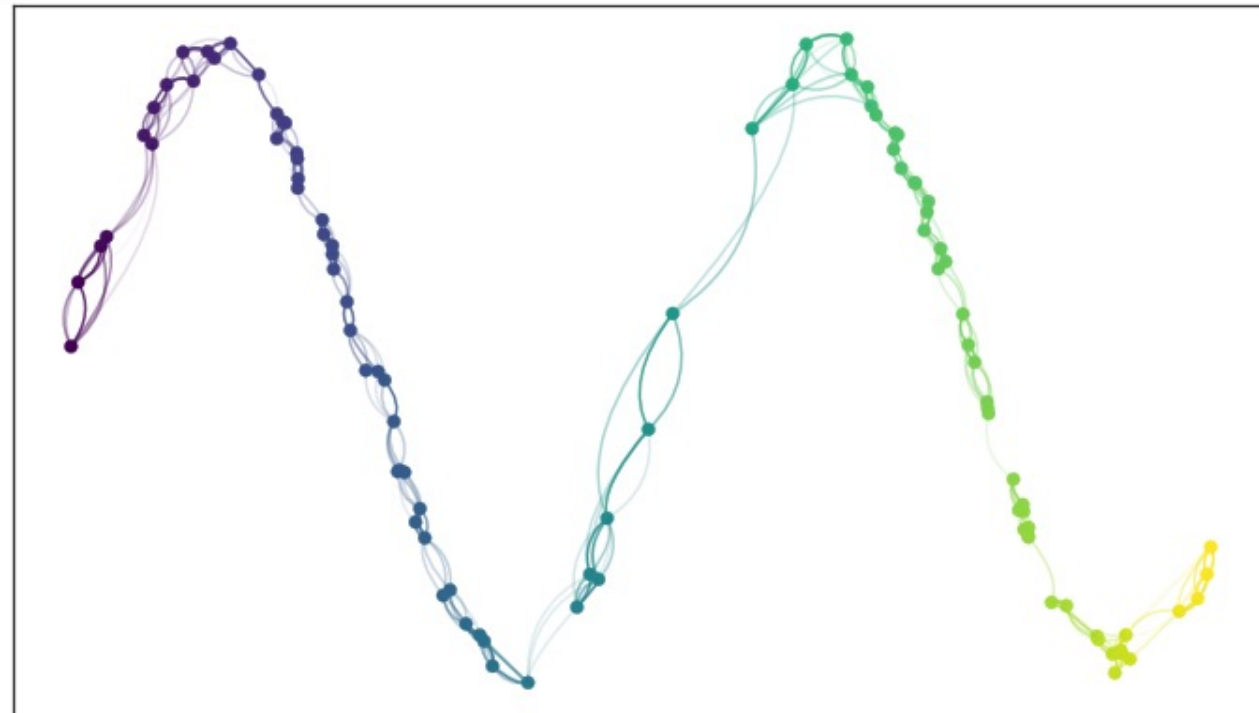

tSNE: optimization

- **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**

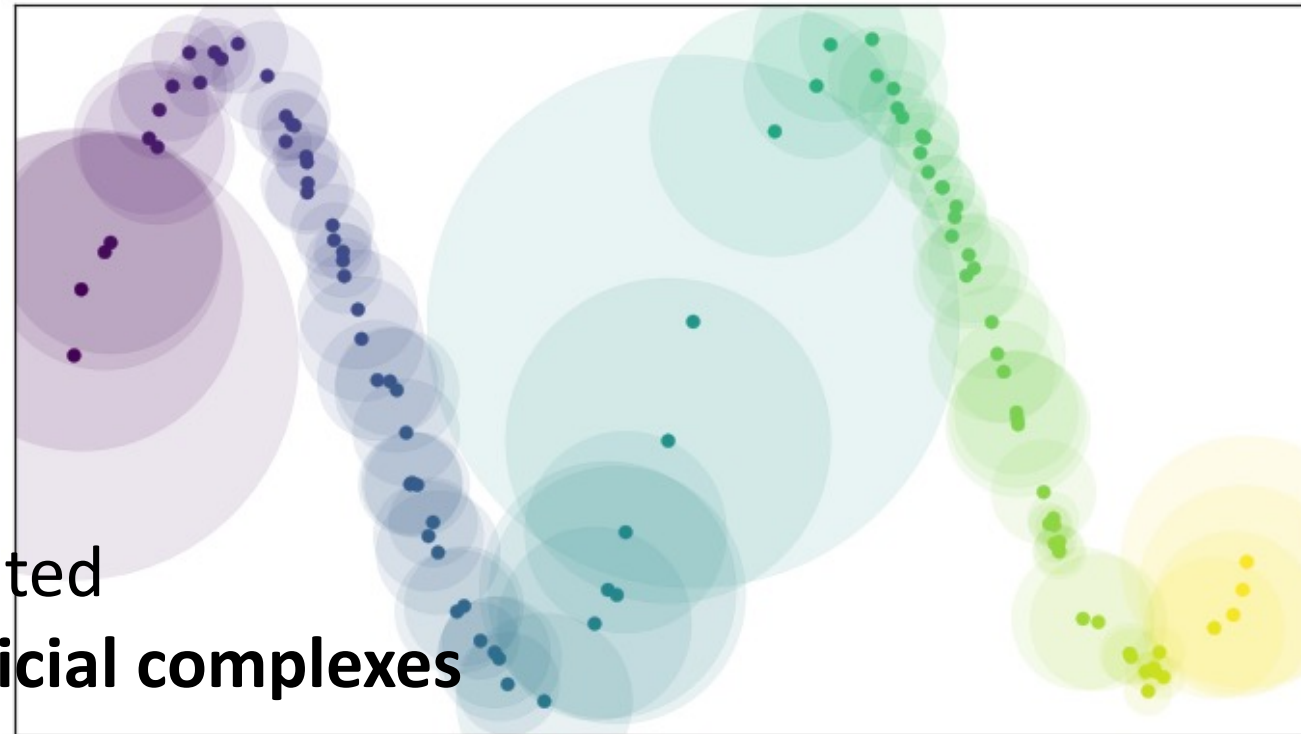
UMAP: Method in brief

- [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**)

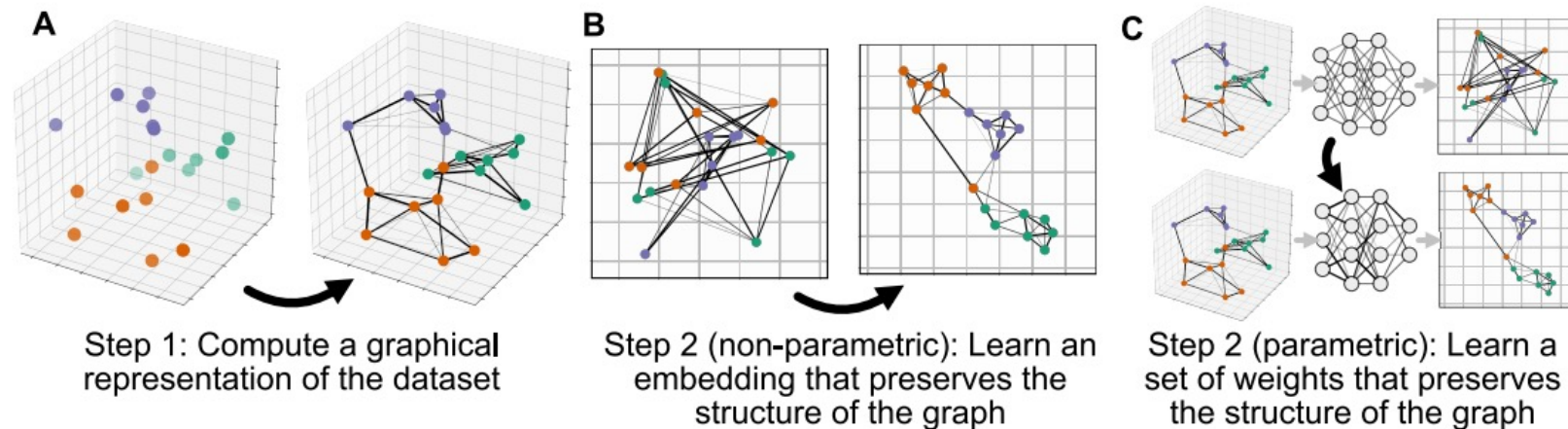
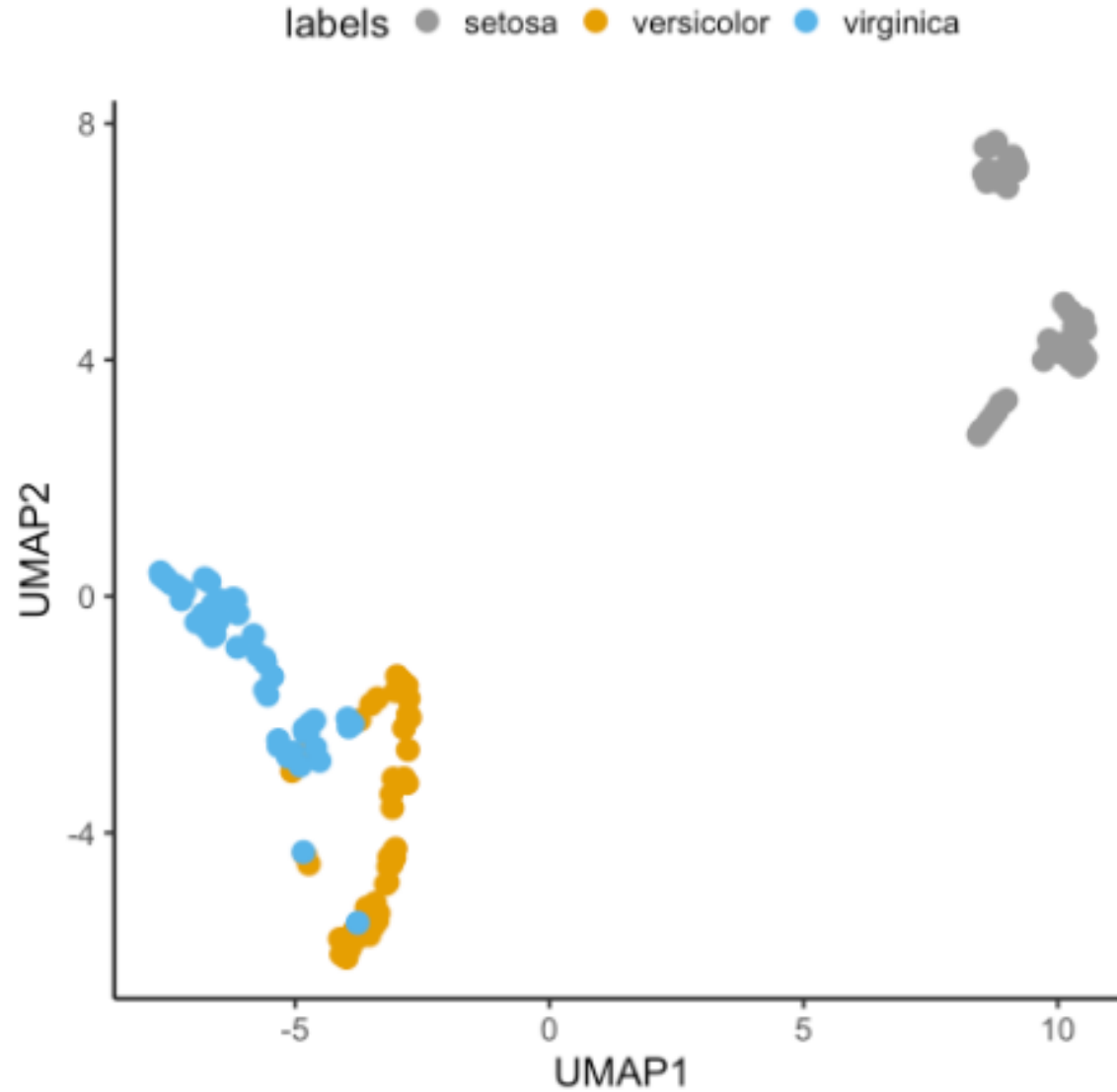


Figure 1: Overview of UMAP (A \rightarrow B) and Parametric UMAP (A \rightarrow C).

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.

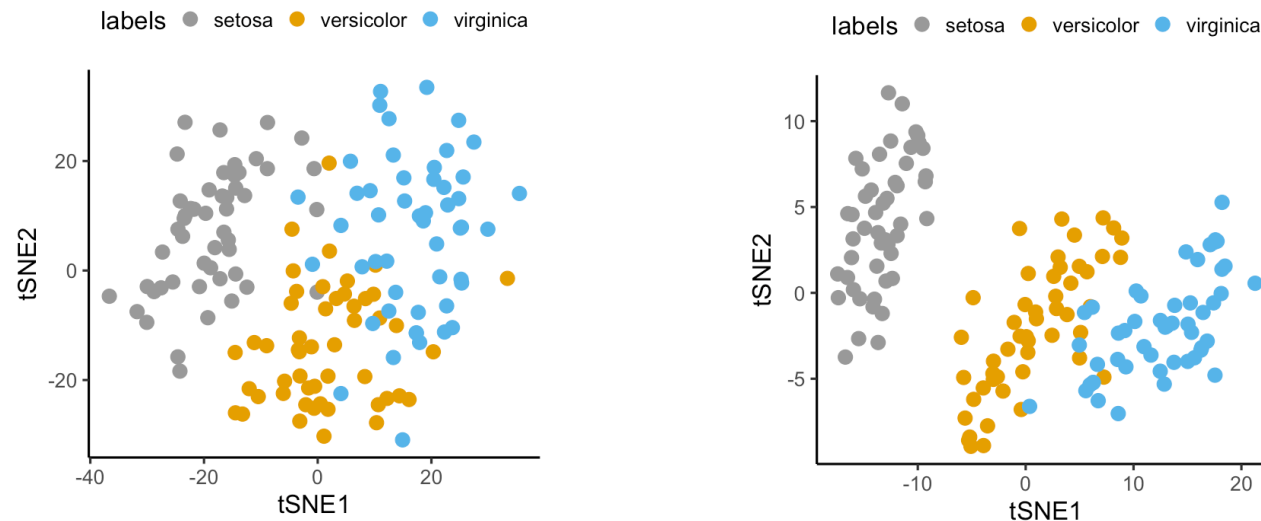
Break – Questions

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

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

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

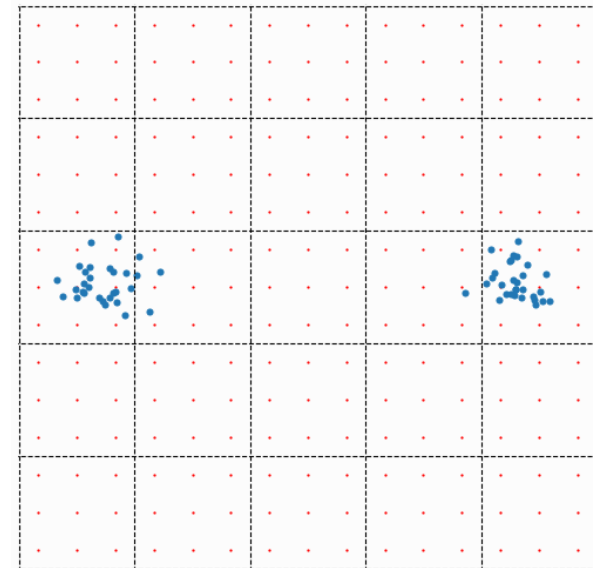
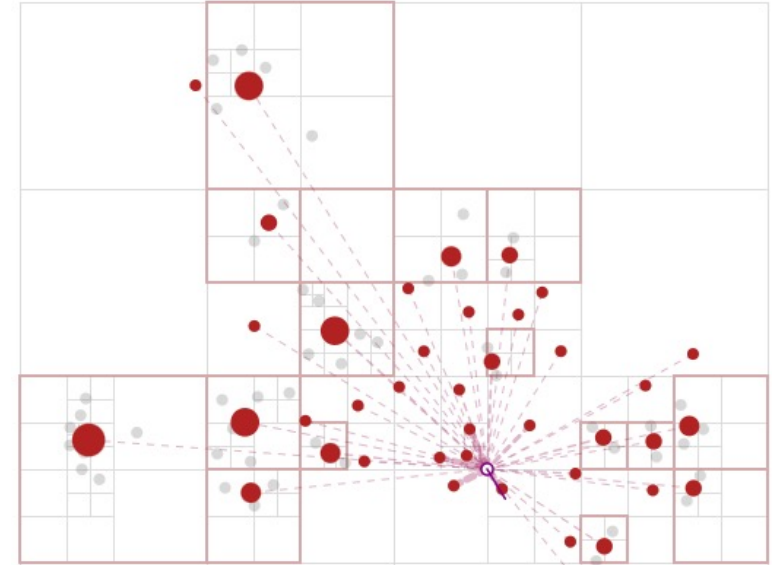
- [Provide an initialization from PCA or other algorithms](#)
- If the data has 100s/1000s of dimensions, provide as proxy input a PCA or other projection with fewer dimensions (e.g. 100)
- Remember to center/standardize your data, e.g.

```
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/