Manifold Learning

Max Turgeon

STAT 4690–Applied Multivariate Analysis

Dimension reduction redux i

- Recall Pearson's approach to PCA: **best approximation of the data by a linear manifold**.
- Let's unpack this definition:
  - We're looking for a linear subspace of $\mathbb{R}^p$ of dimension k.
  - For a fixed k, we want to minimise the error when projecting onto the linear subspace.
  - We can also identify that subspace with $\mathbb{R}^k$ (e.g. for visualisation).

Dimension reduction redux ii

- **Manifold learning** is a nonlinear approach to dimension reduction, where:
  - We assume the data lies on (or close to) a nonlinear manifold of dimension k in $\mathbb{R}^p$.
  - We project the data from the manifold to $\mathbb{R}^k$.
- There are two main classes of methods:
  - Distance preserving (e.g. Isomap);
  - Topology preserving (e.g. Locally linear embedding)

Manifolds–Definition

- Roughly speaking, **manifolds** of dimension k are geometric objects that locally look like $\mathbb{R}^k$.
  - Every point on the manifold has an open neighbourhood that is equivalent to an open ball in $\mathbb{R}^k$.
- Examples in $\mathbb{R}^p$ include any curve, the $(p-1)$-dimensional sphere, or any linear subspace.
- Some manifolds have boundaries (e.g. a cylinder) or corners (e.g. a cube).

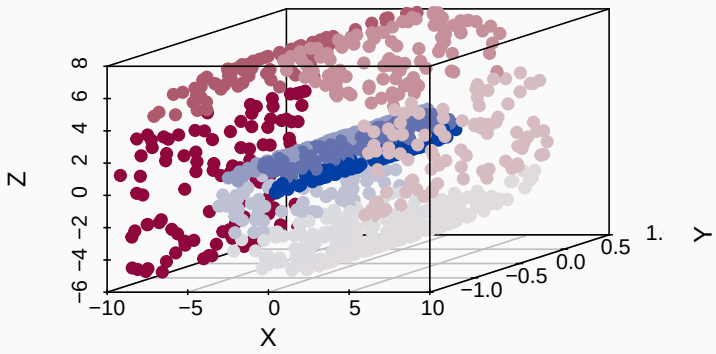
Swiss roll i

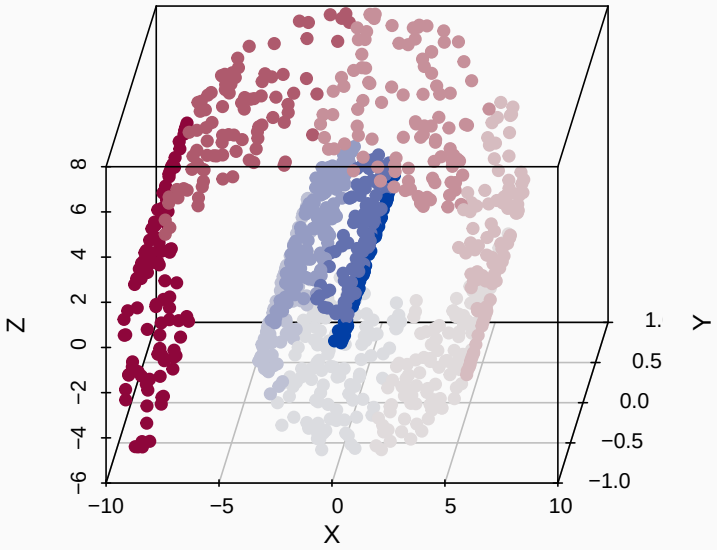
```r
n <- 1000
F1 <- runif(n, 0, 10)
F2 <- runif(n, -1, 1)

X <- F1 * cos(F1)
Y <- F2
Z <- F1 * sin(F1)
```

Swiss roll ii

```
library(scatterplot3d)
library(colorspace)
colours <- cut(F1, breaks = seq(0, 10),
                labels = diverging_hcl(10))
par(mfrow = c(1, 2))
scatterplot3d(X, Y, Z, pch = 19, asp = 1,
              color = colours)
scatterplot3d(X, Y, Z, pch = 19, asp = 1,
              color = colours, angle = 80)
```

Swiss roll iii

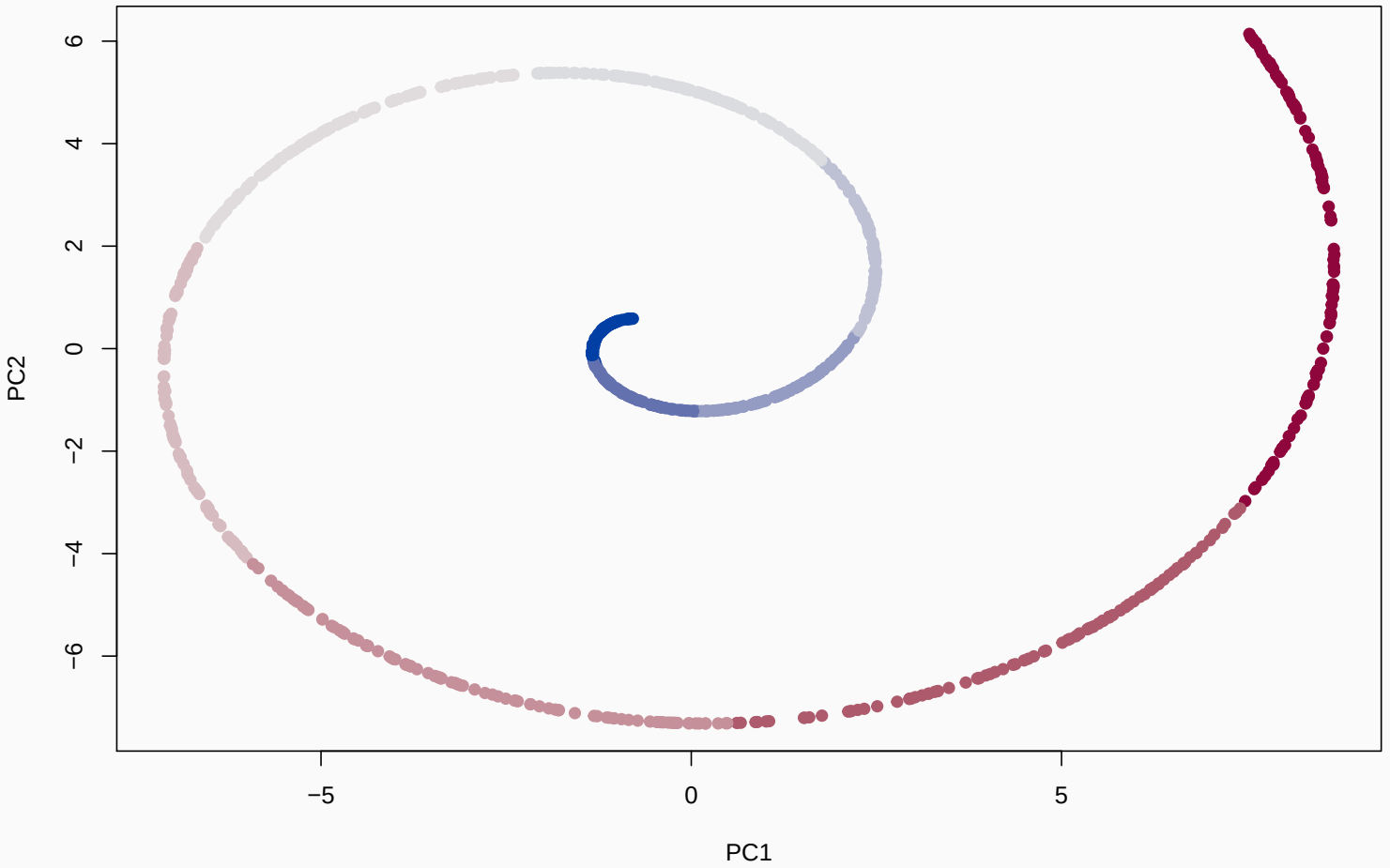




Swiss roll iv

```
# Let's see if PCA can unroll the Swiss roll
decomp <- prcomp(cbind(X, Y, Z))

plot(decomp$x[,1:2],
     col = as.character(colours), pch = 19)
```

Swiss roll v



MNIST data revisited

- To study the nonlinear dimension reduction methods in this lecture, we will restrict our attention to the digit 2 in the MNIST dataset.
- The reason: we can think of the different shapes of 2 as "smooth deformations" of one another.
    - This would work for other digits too (e.g. 6, 9, 8), but not all (e.g. 4, 7).

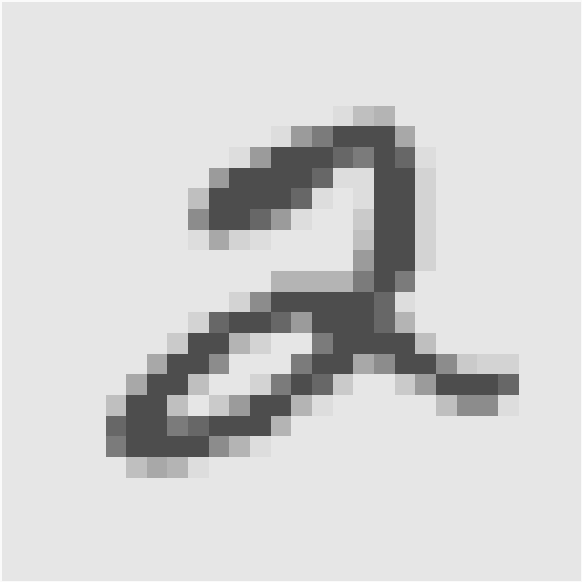
Example i

```
library(dslabs)
library(tidyverse)

mnist <- read_mnist()

data <- mnist$train$images[mnist$train$labels == 2, ]
```

Example ii

```
par(mfrow = c(1, 2))
# With crossing
matrix(data[1,], ncol = 28)[ , 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, asp = 1)
# Without crossing
matrix(data[4,], ncol = 28)[ , 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, asp = 1)
```

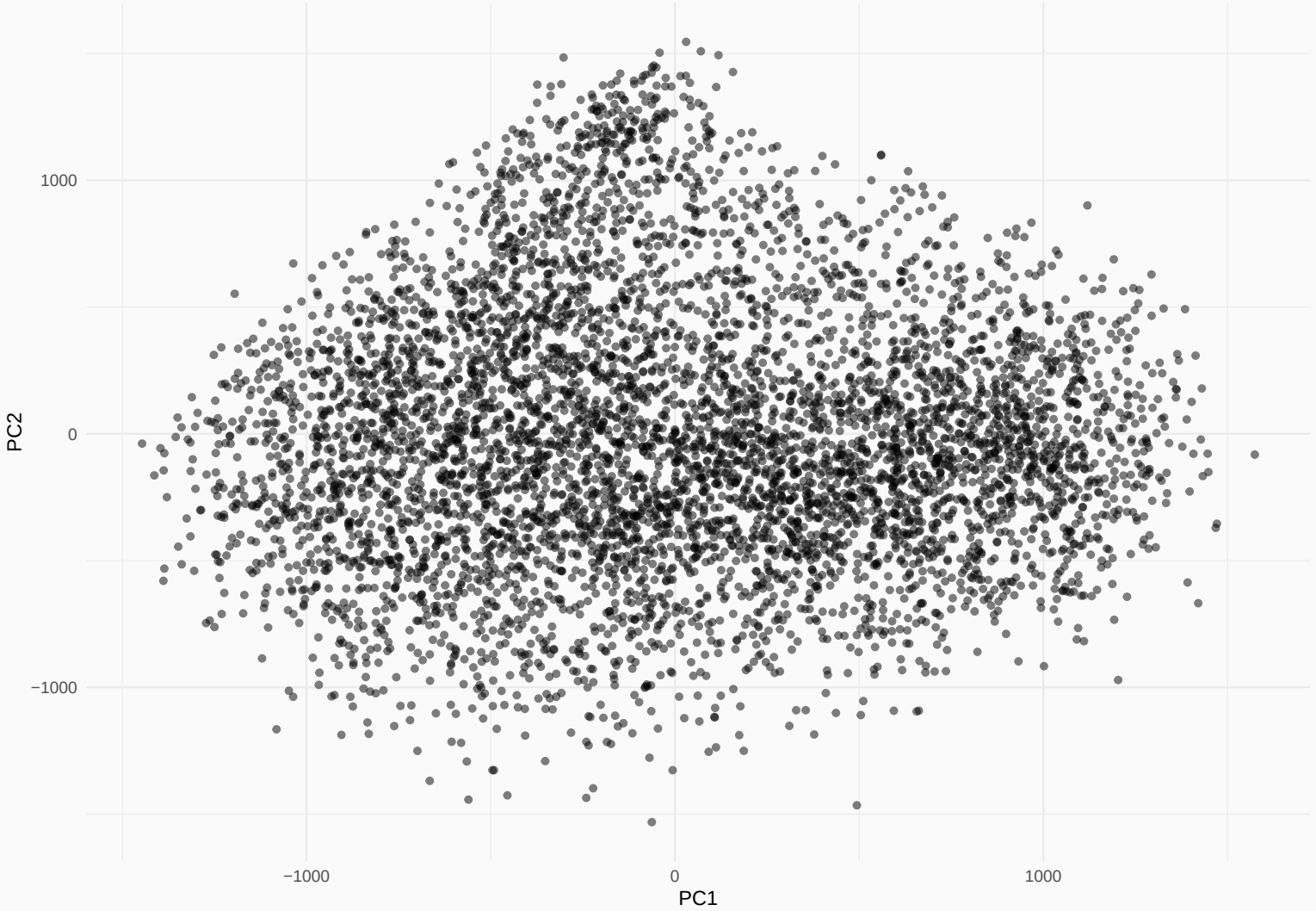
Example iii



Example iv

```
decomp <- prcomp(data)
decomp$x[,1:2] %>%
  as.data.frame() %>%
  ggplot(aes(PC1, PC2)) +
  geom_point(alpha = 0.5) +
  theme_minimal()
```

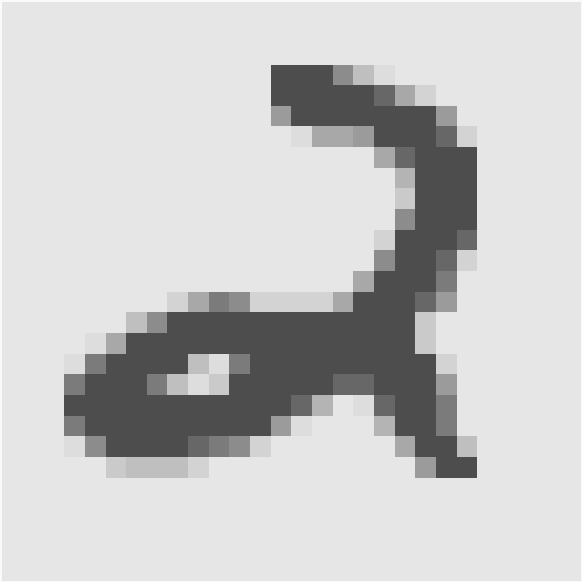
Example v

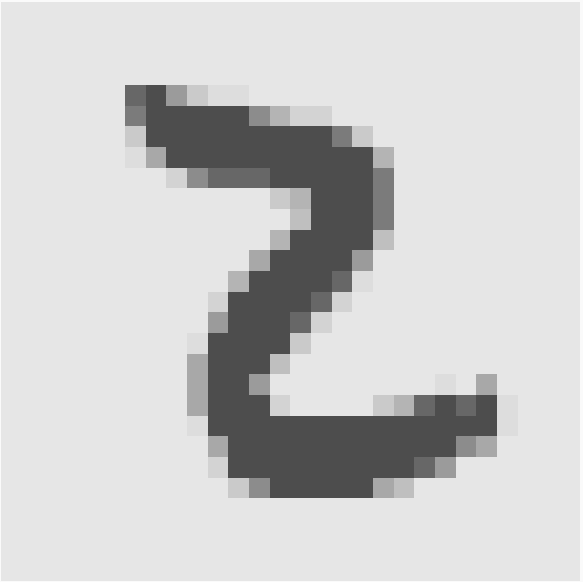


Example vi

```
# First PC
par(mfrow = c(1, 2))
index_right <- which.max(decomp$x[,1])
matrix(data[index_right,], ncol = 28)[ , 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, asp = 1)
index_left <- which.min(decomp$x[,1])
matrix(data[index_left,], ncol = 28)[ , 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, asp = 1)
```

Example vii

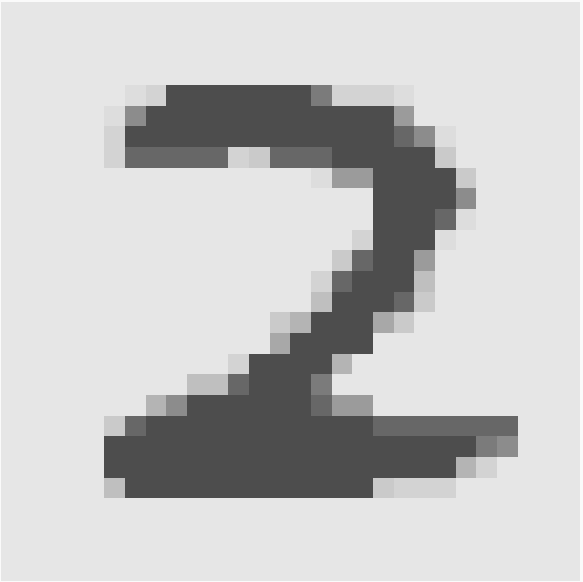




Example viii

```
# Second PC
par(mfrow = c(1, 2))
index_top <- which.max(decomp$x[,2])
matrix(data[index_top,], ncol = 28)[ , 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, asp = 1)
index_bottom <- which.min(decomp$x[,2])
matrix(data[index_bottom,], ncol = 28)[ , 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, asp = 1)
```

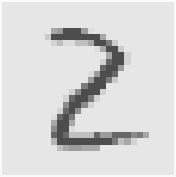
Example ix



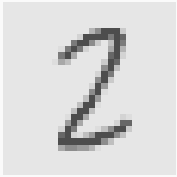
Example x

PC1=−1446

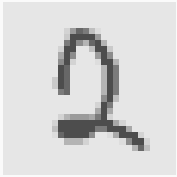
PC1=−906



PC1=−718



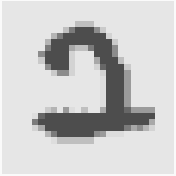
PC1=−546



PC1=−415

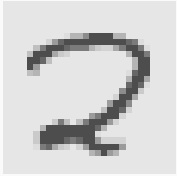
PC1=−290

PC1=−170



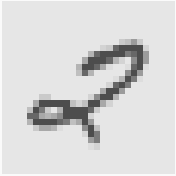
PC1=−57

PC1=89



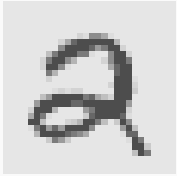
PC1=242

PC1=409



PC1=597

PC1=773



PC1=983

PC1=1574

Example xi

Isomap

Isomap

- Let's look at the algorithm and study each step separately.

Basic algorithm

1. Create a graph $\mathcal{G}$ from the data, where each data point is a node, and two nodes are connected if they are "neighbours".
2. Each edge gets a weight corresponding to the Euclidean distance between the two data points.
3. Create a distance matrix Δ, where the (i, j)-th element is the length of the shortest path in $\mathcal{G}$ between the data points corresponding to nodes i and j.
4. Perform metric Multidimensional Scaling on Δ to obtain the projection onto a lower dimensional subspace.

Definition of neighbourhood

- Two ways of defining the neighbours of a point $\mathbf{Y}$:
  - For an integer $K \geq 1$, we could look at the K-nearest neighbours, i.e. the K points $\mathbf{Y}_1, \ldots, \mathbf{Y}_K$ that are closest (in Euclidean distance) to $\mathbf{Y}$.
  - For a real number $\epsilon > 0$, we could look at all points $\mathbf{Y}_1, \ldots, \mathbf{Y}_{n(\epsilon)}$ whose distance from $\mathbf{Y}$ is less than ϵ.
- **Note**: The first definition guarantees that every point has neighbours, whereas you could get unconnected points using the second definition.
- You could also use a hybrid of both approaches where you take the K-nearest neighbours, but discard neighbours that are "too far away".

Shortest path distance i

- Once we have our weighted graph $\mathcal{G}$ (i.e. nodes represent data points, edges represent neighbours, weights are Euclidean distances), we can compute the length of any path from $\mathbf{Y}_i$ to $\mathbf{Y}_j$ by summing the weights of all the edges along the path.
- We then define a **distance** function on $\mathcal{G}$ by

$$\Delta_{ij} = \min \{\text{Length of path } \gamma \mid \gamma \text{ is a path from } \mathbf{Y}_i \text{ to } \mathbf{Y}_j\}.$$

Shortest path distance ii

- There are efficient algorithms for computing this distance for any weighted graph:
  - Dijkstra's algorithm;
  - Floyd–Warshall algorithm.
- For more details about these algorithms, take a course on graph theory!

Multidimensional Scaling

Recall the algorithm for MDS.

Algorithm (MDS)
Input: Δ; Output: $\tilde{X}$

1. Create the matrix D containing the square of the entries in Δ.
2. Create S by centering both the rows and the columns and multiplying by $-\frac{1}{2}$.
3. Compute the eigenvalue decomposition $S = U\Lambda U^T$.
4. Let $\tilde{X}$ be the matrix containing the first r columns of $\Lambda^{1/2}U^T$.

Swiss roll i

```
library(dimRed)
```

```
isomap_sr <- embed(cbind(X, Y, Z), "Isomap", knn = 10,
                   ndim = 2)
```

```
## 2019-11-21 17:24:55: Isomap START
```

```
## 2019-11-21 17:24:55: constructing knn graph
```

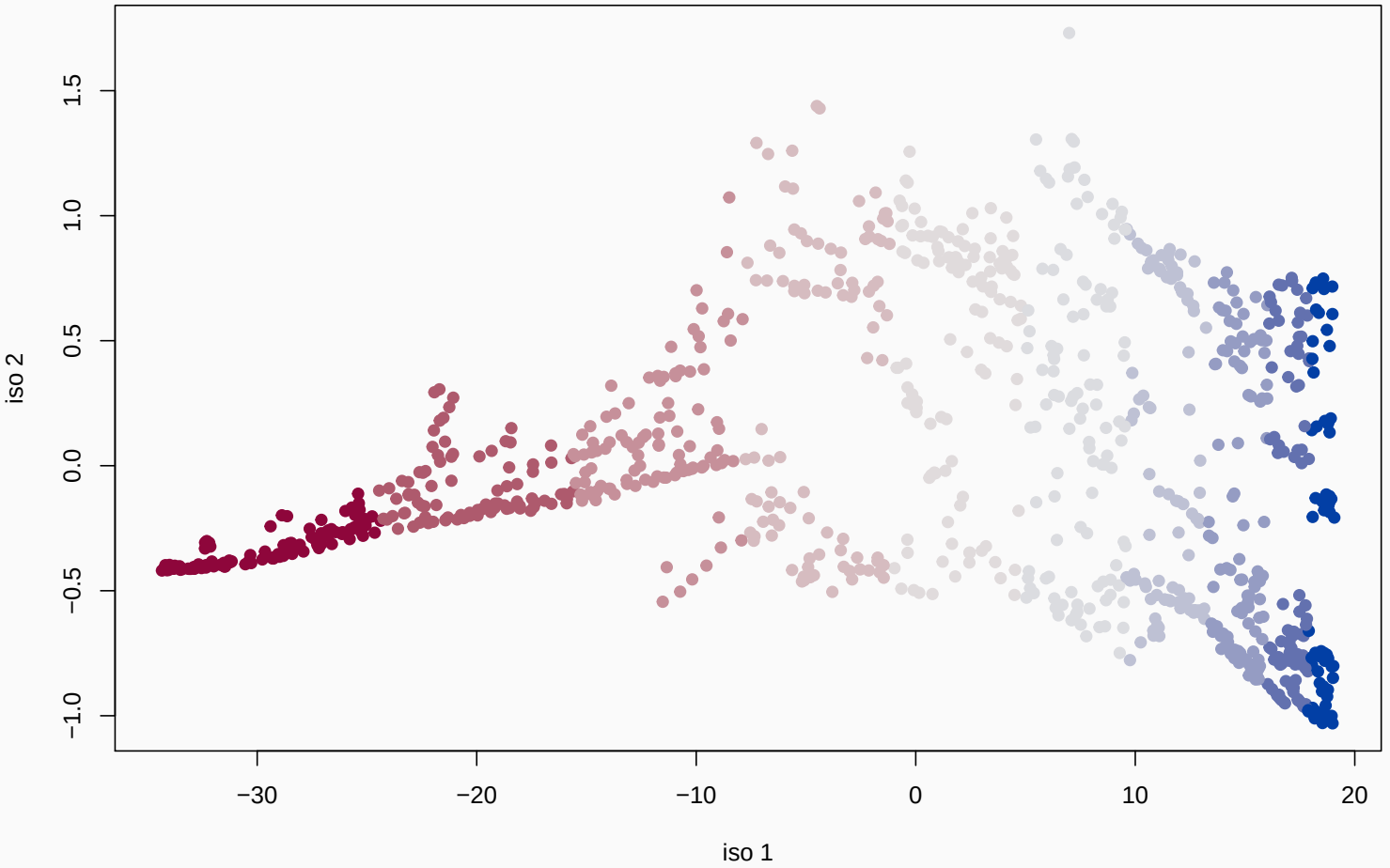
```
## 2019-11-21 17:24:55: calculating geodesic distances
```

Swiss roll ii

```
## 2019-11-21 17:24:55: Classical Scaling
```

```
isomap_sr@data@data %>%
  plot(col = as.character(colours), pch = 19)
```

Swiss roll iii



Example i

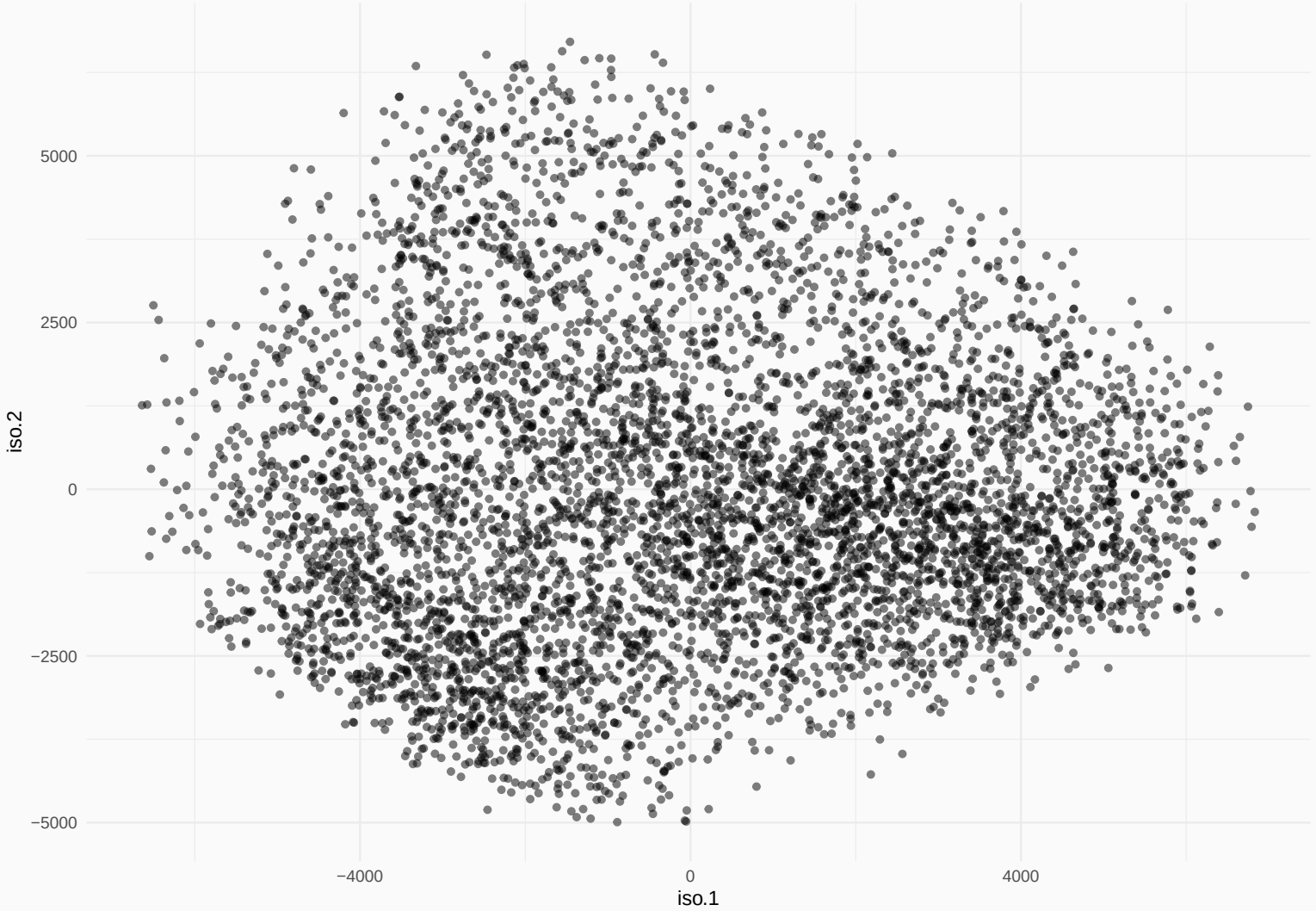
```
isomap_res <- embed(data, "Isomap", knn = 10,
                    ndim = 2)
```

```
## 2019-11-21 17:24:55: Isomap START

## 2019-11-21 17:24:55: constructing knn graph

## 2019-11-21 17:25:15: calculating geodesic distances

## 2019-11-21 17:25:28: Classical Scaling
```

Example ii

```
isomap_res@data %>%
  as.data.frame() %>%
  ggplot(aes(iso.1, iso.2)) +
  geom_point(alpha = 0.5) +
  theme_minimal()
```

Example iii



Example iv

ISO1=−6638

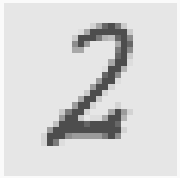
ISO1=−4258

ISO1=−3369

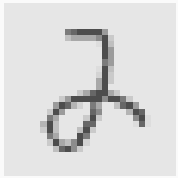
ISO1=−2722

ISO1=−2114

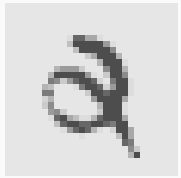
ISO1=−1504



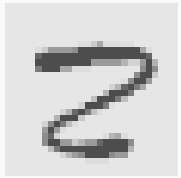
ISO1=−818



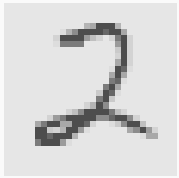
ISO1=−150



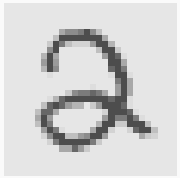
ISO1=548



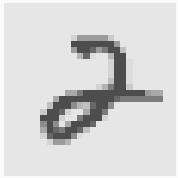
ISO1=1322



ISO1=2017

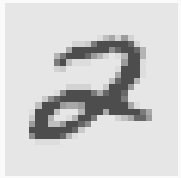


ISO1=2762



ISO1=3589

ISO1=4558

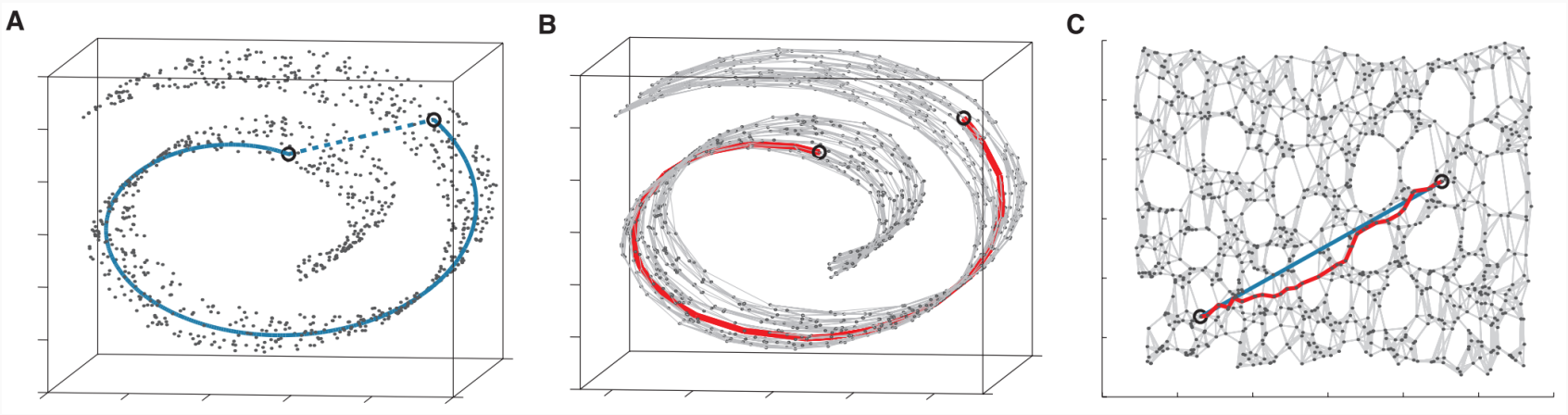


ISO1=6831

Intuition i

- The reason why Isomap works is because the shortest path distance approximates the **geodesic** distance on the manifold
  - "Train tracks distance"
- If we embed the weighted graph in $\mathbb{R}^p$, with each nodes at its corresponding data point, and each edge having length equal to the Euclidean distance, we can see the graph as a scaffold of the manifold.
- As we increase the sample size, the scaffold "converges" to the actual manifold.

Intuition ii



Tenenbaum *et al.* Science (2000)

Further examples i

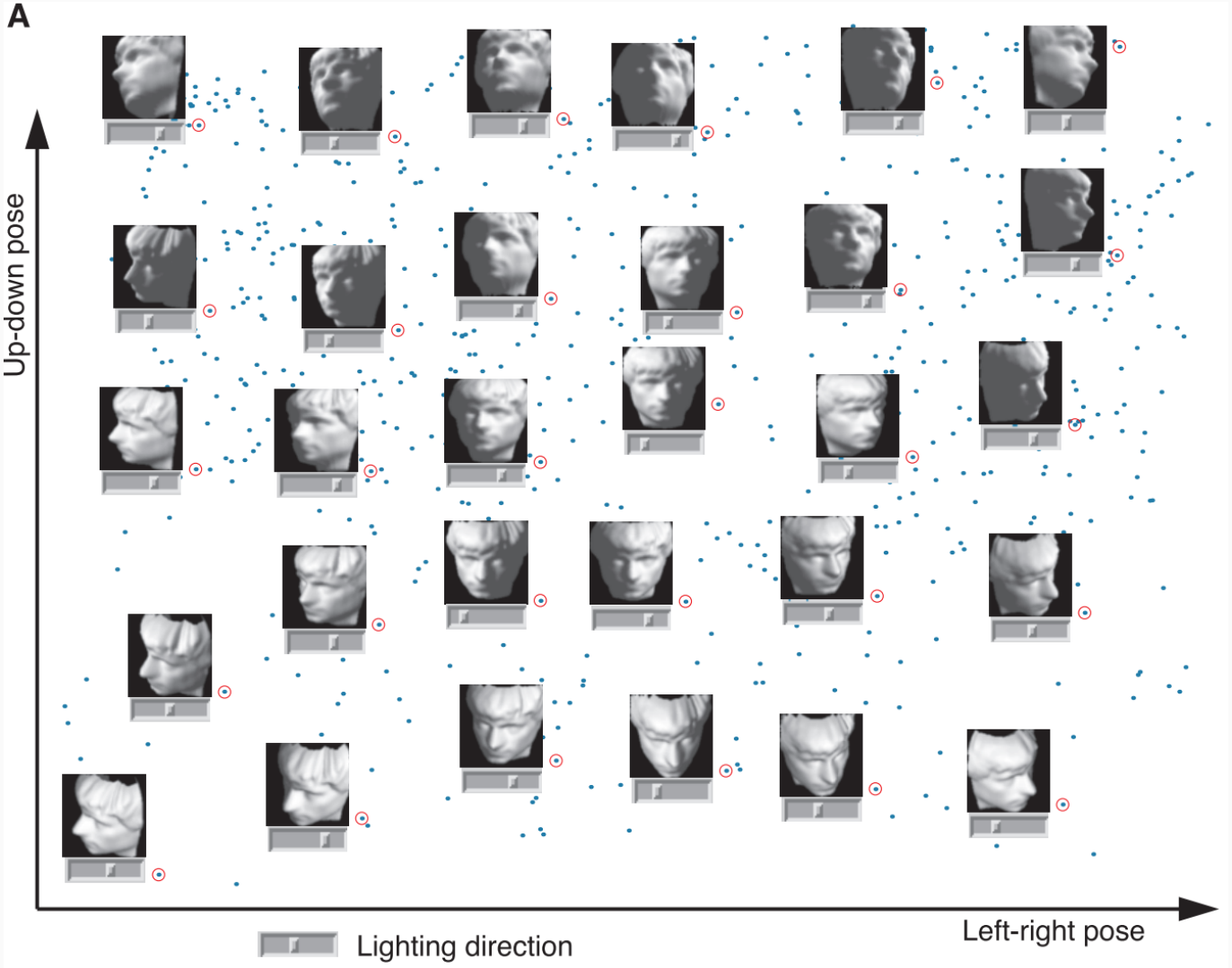


Figure 1

Further examples ii

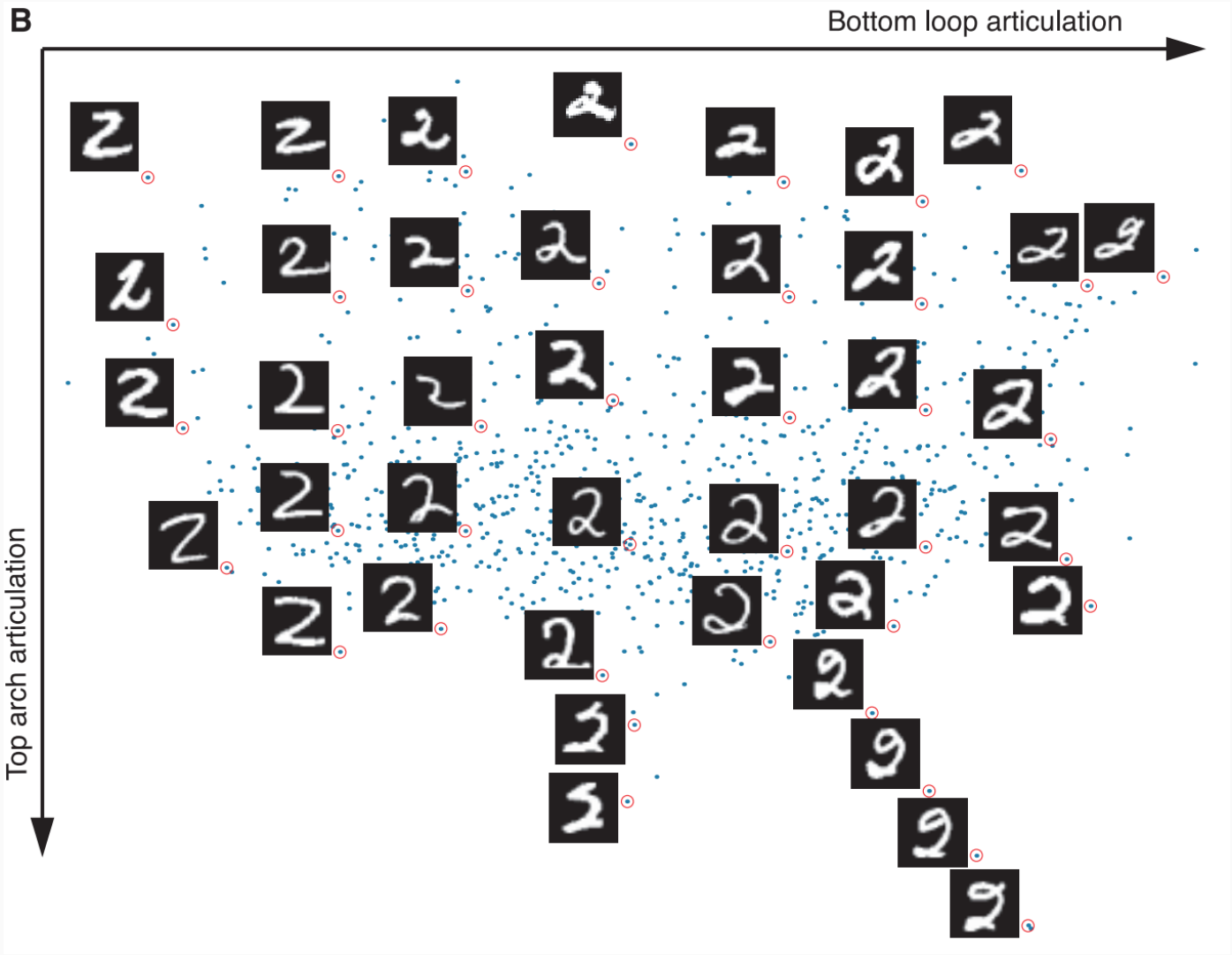


Figure 2

Comments

- Advantages:
  - Simple extension of MDS
  - Preserves distance relationship on the manifold
- Disadvantages:
  - Computing the shortest path distance can be expensive with many data points
  - Doesn't work well with all manifolds (e.g. it fails when the underlying manifold has holes or many folds)

Locally Linear Embedding

Local vs. Global structure

- In Isomap, we estimated pairwise distances by constraining them to be close to the underlying manifold.
  - But we still computed all $O(n^2)$ distances...
- LLE instead focuses on *local* stuctures in the data.
- In particular, it assumes that a linear approximation of these local structures will give a good approximation of the global (nonlinear) structure.

Preserving local structure

- The motivation for LLE is as follows:
  - A given point should be well approximated by a linear combination of its **neighbours**.
  - We want those linear combinations weights to be **invariant** to rotation, scaling, and translation.
  - Therefore, the same weights should be used if we replace the original data with a lower dimensional representation.

Algorithm i

- First, some notation:
  - $\mathbf{Y}_i, i = 1, \ldots, n$ are the p-dimensional data points.
  - $\mathbf{X}_i, i = 1, \ldots, n$ are their k-dimensional representation.
  - If $\mathbf{Y}_j$ is a neighbour of $\mathbf{Y}_i$, we write $j \in \mathcal{N}(i)$.
  - W is an $n \times n$ matrix of weights such that $w_{ij} = 0$ if $\mathbf{Y}_j$ is **not** a neighbour of $\mathbf{Y}_i$.
  - We also impose a constraint that $\sum_{j=1}^{n} w_{ij} = 1$ for all i, i.e. the rows of W sum to 1.

Algorithm ii

LLE Algorithm

Input: $\mathbf{Y}_i \in \mathbb{R}^p, i = 1, \ldots, n.$

Output: $\mathbf{X}_i \in \mathbb{R}^k, i = 1, \ldots, n.$

1. Estimate W by minimising the *reconstruction error*:

$$\hat{W} = \arg\min_W \sum_{i=1}^{n} \|\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij} \mathbf{Y}_j\|^2.$$

2. With $\hat{W}$ fixed, estimate $\mathbf{X}_1, \ldots, \mathbf{X}_n$ by minimising the *embedding cost*:

$$\hat{\mathbb{X}} = \arg\min_{\mathbb{X}} \sum_{i=1}^{n} \|\mathbf{X}_i - \sum_{j=1}^{n} \hat{w}_{ij} \mathbf{X}_j\|^2.$$

Reconstruction error i

- Let $\mathcal{E}_i(W) = \|\mathbf{Y}_i - \sum_{j=1}^n w_{ij}\mathbf{Y}_j\|^2$.
- Recall that we would like invariance under rotation, scaling, and translation.
- Let α be a scalar. We have

$$\|\alpha\mathbf{Y}_i - \sum_{j=1}^n w_{ij}\alpha\mathbf{Y}_j\|^2 = \alpha^2\|\mathbf{Y}_i - \sum_{j=1}^n w_{ij}\mathbf{Y}_j\|^2,$$

and therefore the minimiser of $\mathcal{E}_i(W)$ is the same after rescaling all points by α.

Reconstruction error ii

- Next, let T be a $p \times p$ orthogonal matrix. We have

Reconstruction error iii

$$
\begin{aligned}
\|T\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij} T\mathbf{Y}_j\|^2 &= \left\| T\left(\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\right)\right\|^2 \\
&= \left(T\left(\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\right)\right)^T \left(T\left(\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\right)\right) \\
&= \left(\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\right)^T T^T T \left(\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\right) \\
&= \left(\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\right)^T \left(\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\right) = \mathcal{E}_i(W).
\end{aligned}
$$

Reconstruction error iv

- Finally, let $\mu \in \mathbb{R}^p$. We have

Reconstruction error v

$$
\begin{aligned}
&\|(\mathbf{Y}_i - \mu) - \sum_{j=1}^{n} w_{ij}(\mathbf{Y}_j - \mu)\|^2 \\
&\quad = \|\mathbf{Y}_i - \mu - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j + \sum_{j=1}^{n} w_{ij}\mu\|^2 \\
&\quad = \|\mathbf{Y}_i - \mu - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j + \mu\sum_{j=1}^{n} w_{ij}\|^2 \\
&\quad = \|\mathbf{Y}_i - \mu - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j + \mu\|^2 \\
&\quad = \|\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\|^2 = \mathcal{E}_i(W).
\end{aligned}
$$

Reconstruction error vi

- In other words, invariance comes from the definition of reconstruction error **and** the constraint that weights sum to 1.
- **How to minimise $\mathcal{E}_i(W)$?** Assume that the neighbours of $\mathbf{Y}_i$ are $\mathbf{Y}_{(1)}, \ldots, \mathbf{Y}_{(r)}$. We then have

Reconstruction error vii

$$
\begin{aligned}
\mathcal{E}_i(W) &= \|\mathbf{Y}_i - \sum_{j=1}^{n} w_{ij}\mathbf{Y}_j\|^2 \\
&= \|\mathbf{Y}_i - \sum_{j=1}^{r} w_{i(j)}\mathbf{Y}_{(j)}\|^2 \\
&= \left\| \sum_{j=1}^{r} w_{i(j)} \left(\mathbf{Y}_i - \mathbf{Y}_{(j)}\right) \right\|^2 \\
&= \sum_{j=1}^{r}\sum_{k=1}^{r} w_{i(j)}w_{i(k)} \left(\mathbf{Y}_i - \mathbf{Y}_{(j)}\right)^T \left(\mathbf{Y}_i - \mathbf{Y}_{(k)}\right).
\end{aligned}
$$

Reconstruction error viii

- Let $G(i)$ be the matrix whose (j,k)-th entry is equal to $\left(\mathbf{Y}_i - \mathbf{Y}_{(j)}\right)^T \left(\mathbf{Y}_i - \mathbf{Y}_{(k)}\right)$.
- Using the method of Lagrange multipliers, we can minimise $\mathcal{E}_i(W)$ by solving the linear system

$$G\mathbf{w} = \mathbf{1}$$

  and normalising the weights so they add up to 1.
- If G is singular (or nearly singular), you can add a small constant to the diagonal to regularise it.

Example i

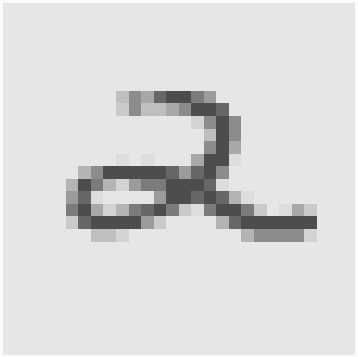
```
n <- 1000
data_knn <- data[seq_len(n),]

# Compute distances
Delta <- dist(data_knn)

# Take 5-NN to first obs.
neighbours <- order(Delta[seq_len(n-1)])[1:5]
```

Example ii

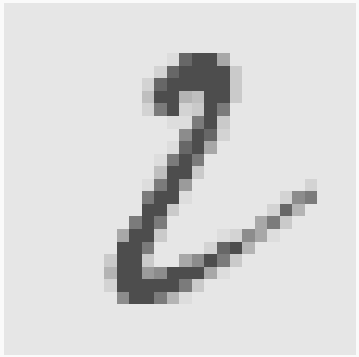
```
main_obs <- data_knn[1,]
nb_data <- data_knn[neighbours,]
```

Example iii

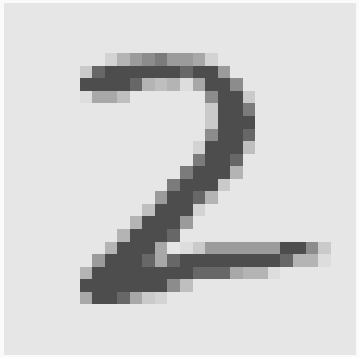
NN 1



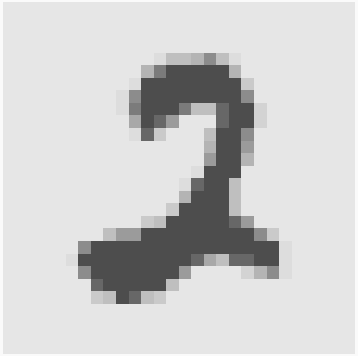
NN 2



NN 3



NN 4



NN 5

Example iv

```
# Center neighbours around main obs
nb_data_c <- sweep(nb_data, 2, main_obs)
# Local cov matrix
Gmat <- tcrossprod(nb_data_c)

# Find weights
w_vect <- solve(Gmat, rep(1, 5))
w_vect <- w_vect/sum(w_vect)
```

Example v

```
# Compare original with approx.
approx <- drop(w_vect %*% nb_data)
par(mfrow = c(1, 2))

matrix(main_obs, ncol = 28)[, 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, main = "Original", asp = 1)
matrix(approx, ncol = 28)[, 28:1] %>%
  image(col = gray.colors(12, rev = TRUE),
        axes = FALSE, main = "Approx. (5 NN)", asp = 1)
```

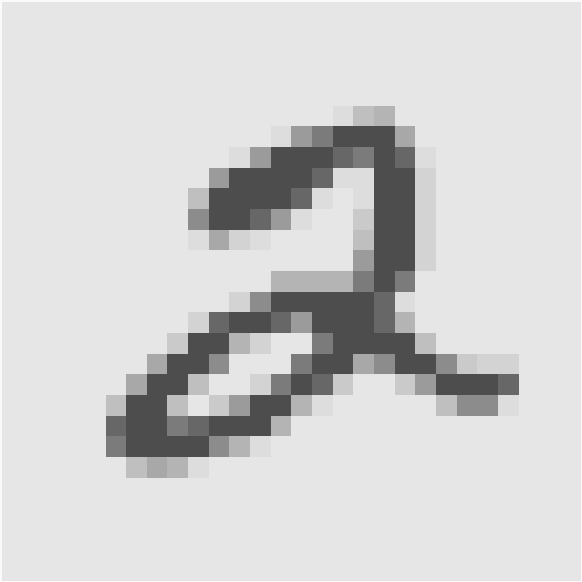
Example vi

Original

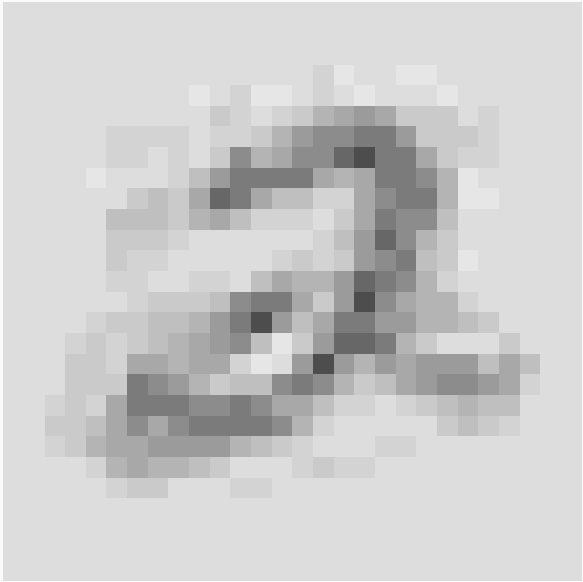
Approx. (5 NN)

Example vii

Original



Approx. (25 NN)



Embedding cost i

- Let $\Phi(\mathbb{X}) = \sum_{i=1}^{n} \left\| \mathbf{X}_i - \sum_{j=1}^{n} w_{ij} \mathbf{X}_j \right\|^2$.
- As we did earlier, we can rewrite this:

$$\begin{aligned}
\sum_{i=1}^{n} \left\| \mathbf{X}_i - \sum_{j=1}^{n} w_{ij} \mathbf{X}_j \right\|^2 &= \sum_{i=1}^{n} \left\| \sum_{j=1}^{n} w_{ij} \left(\mathbf{X}_i - \mathbf{X}_j \right) \right\|^2 \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} w_{ik} \left(\mathbf{X}_i - \mathbf{X}_j \right)^T \left(\mathbf{X}_i - \mathbf{X}_k \right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} m_{ij} \mathbf{X}_i^T \mathbf{X}_j .
\end{aligned}$$

Embedding cost ii

- Above, m_{ij} is the (i, j)-th element of the matrix M, where
$$M = (I - W)^T (I - W).$$
- **Key observation**: M is sparse (i.e. lots of zeroes), symmetric, and positive semidefinite.
- If we impose some restrictions on the projections $\mathbf{X}_i$ (i.e. mean zero, identity covariance matrix), we can minimise $\Phi(\mathbb{X})$ subject to these constraints using Lagrange multipliers.
- The smallest eigenvalue will be zero; we can discard its corresponding eigenvector.

Embedding cost iii

- The eigenvectors corresponding to the next k smallest eigenvalues give us our matrix $\mathbb{X}$ that minimises the embedding cost.

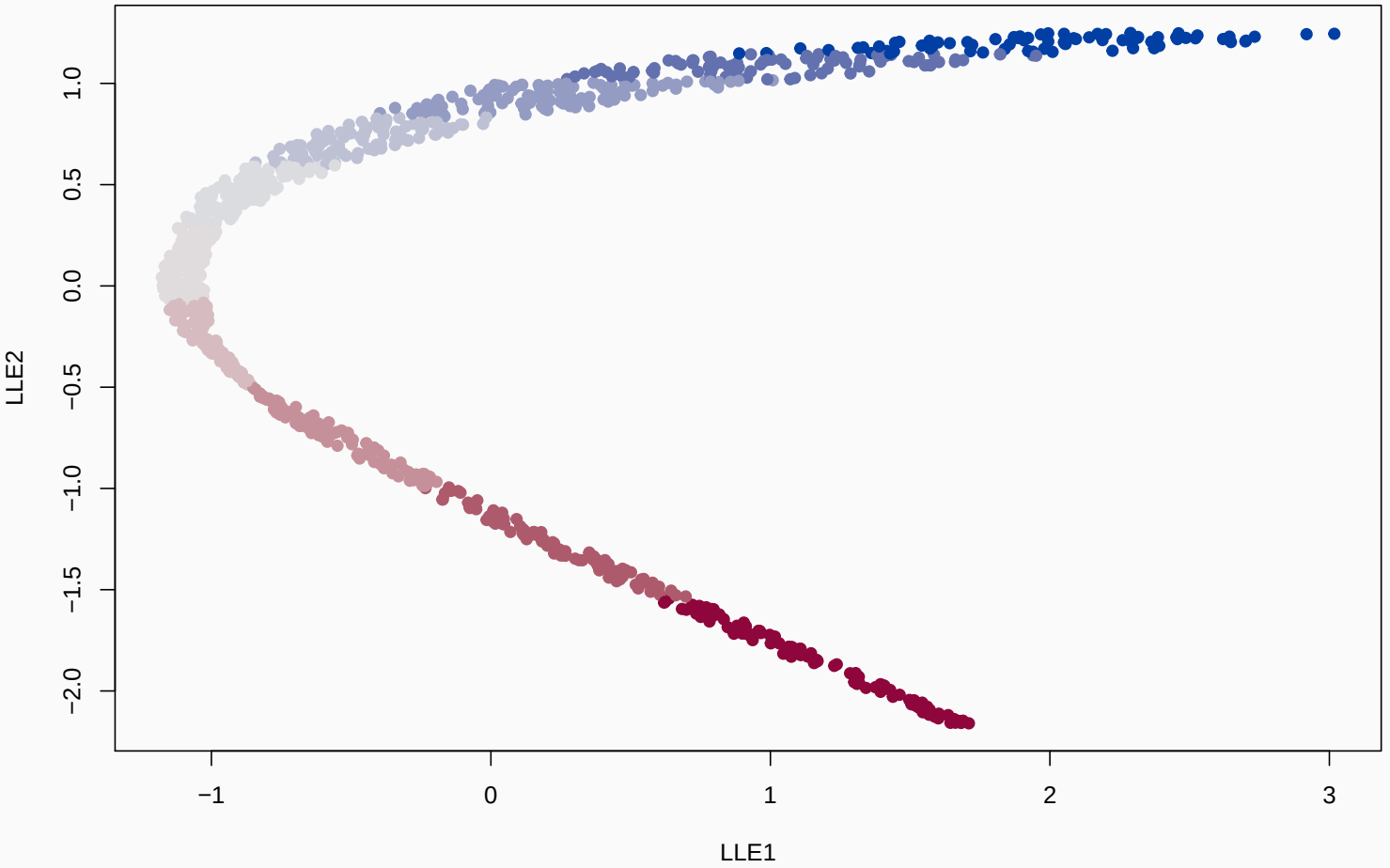
Comments

- Since M is sparse, we can compute these eigenvectors very efficiently using specialised algorithms.
- Since we obtained the data matrix $\mathbb{X}$ as eigenvectors of M, it may seem that we did a linear dimension reduction. However, the sparsity of W (and therefore M) is what gives us our *nonlinear* dimension reduction.

Swiss roll i

```
lle_sr <- embed(cbind(X, Y, Z), "LLE", knn = 20,
                ndim = 2)


## finding neighbours
## calculating weights
## computing coordinates

lle_sr@data@data %>%
  plot(col = as.character(colours), pch = 19)
```

Swiss roll ii



Example i

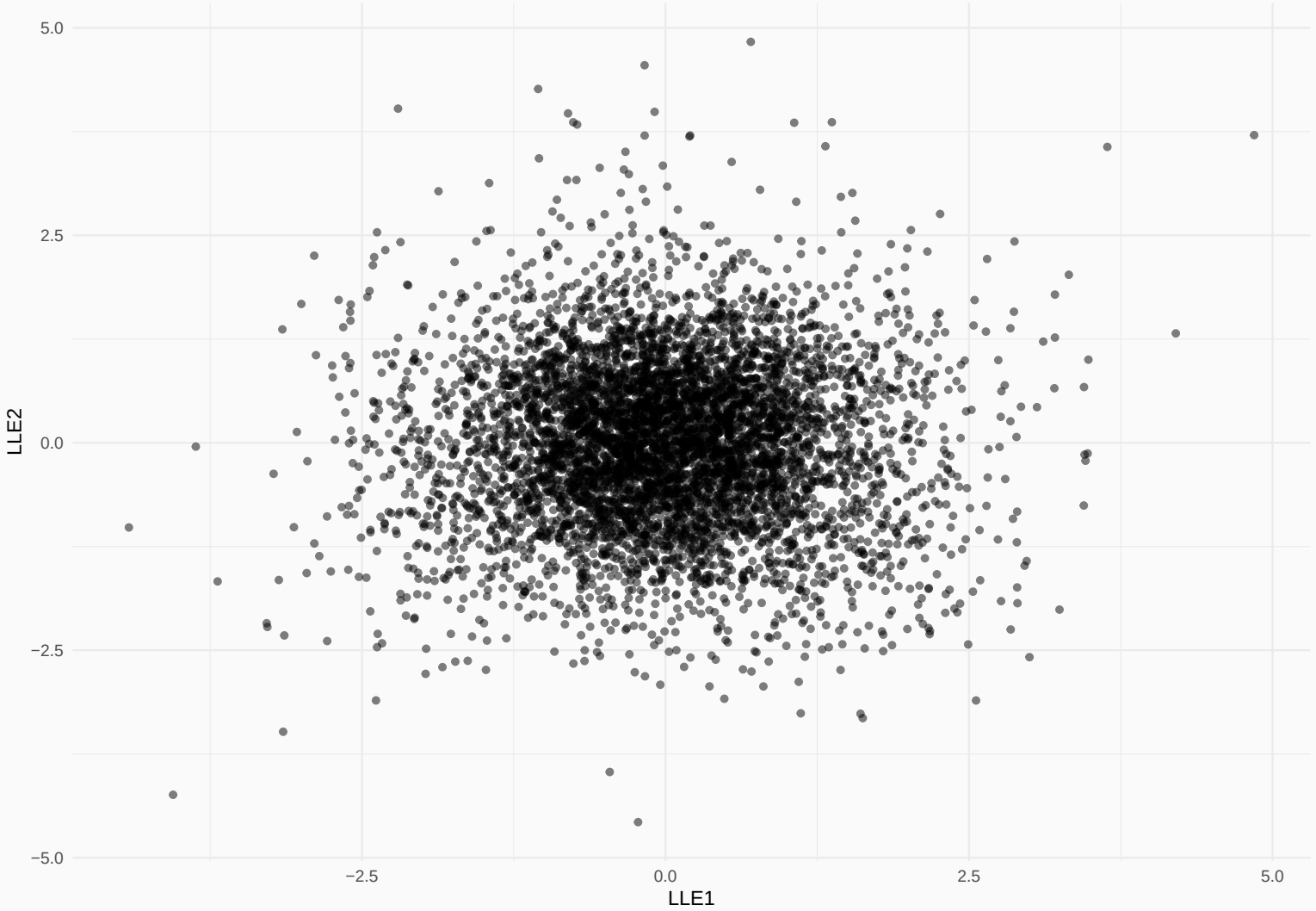
```
lle_res <- embed(data, "LLE", knn = 50,
                 ndim = 2)
```

```
## finding neighbours
## calculating weights
## computing coordinates
```

Example ii

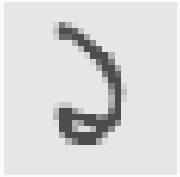
```
lle_res@data %>%
  as.data.frame() %>%
  ggplot(aes(LLE1, LLE2)) +
  geom_point(alpha = 0.5) +
  theme_minimal()
```

Example iii



Example iv

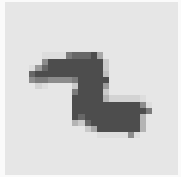
ISO1=−4



ISO1=−1

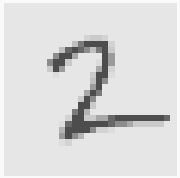
ISO1=−1

ISO1=−1



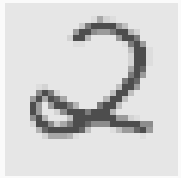
ISO1=−1

ISO1=0

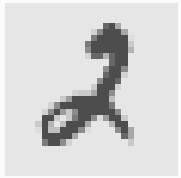


ISO1=0

ISO1=0



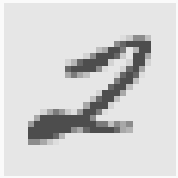
ISO1=0



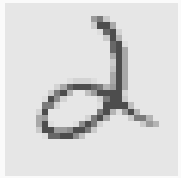
ISO1=0

ISO1=1

ISO1=1



ISO1=1



ISO1=1

ISO1=5

Further comments

- Advantages:
  - Preserves local structure
  - Less computationally expensive than Isomap
- Disadvantages:
  - Less accurate in preserving global structure
  - Doesn't work well with all manifolds (e.g. it fails when the underlying manifold is nonconvex)