ISOMAP Simulations

Bushra Haque, Yichen Ji, Luis Sierra Muntané

2025-04-04

Simulations for the accompanying report on the ISOMAP algorithm by Tenenbaum et al. 2000. Every once in a while there will be code that is commented out involving prefixes such as `rgl` or the function `plot3d`. This code allows for an interactive view of 3D points as scatter plots, which unfortunately cannot be rendered into a pdf file. They are left in for the users that may be interested in such plots and able to run the code themselves.

```
# Libraries used for visualization and to import the MNIST dataset
library(rgl)
```

```
## Warning in rgl.init(initValue, onlyNULL): RGL: unable to open X11 display
```

```
## Warning: 'rgl.init' failed, will use the null device.
## See '?rgl.useNULL' for ways to avoid this warning.
```

```
library(scatterplot3d)
library(keras)
options(rgl.useNULL = TRUE)
```

Algorithm Implementation:

```
# Helper functions for ISOMAP embedding

# Builds the graph by making edges between a vertex and its K nearest neighbours
k_build_graph <- function(n, k, dist_matrix){

  # Initialize graph with Inf and 0 on the diagonal
  graph <- matrix(Inf, n, n)
  diag(graph) <- 0

  # For each point, keep distances only to its k-nearest neighbors
  for (i in 1:n) {
    # order returns indices sorted by distance (first element is i itself)
    neighbors <- order(dist_matrix[i,])[2:(k + 1)]
    graph[i, neighbors] <- dist_matrix[i, neighbors]
  }
  # To ensure symmetry of the adjacency matrix
  for (i in 1:n) {
    for (j in 1:n) {
      graph[i, j] <- min(graph[i, j], graph[j, i])
      graph[j, i] <- graph[i, j]
    }
```

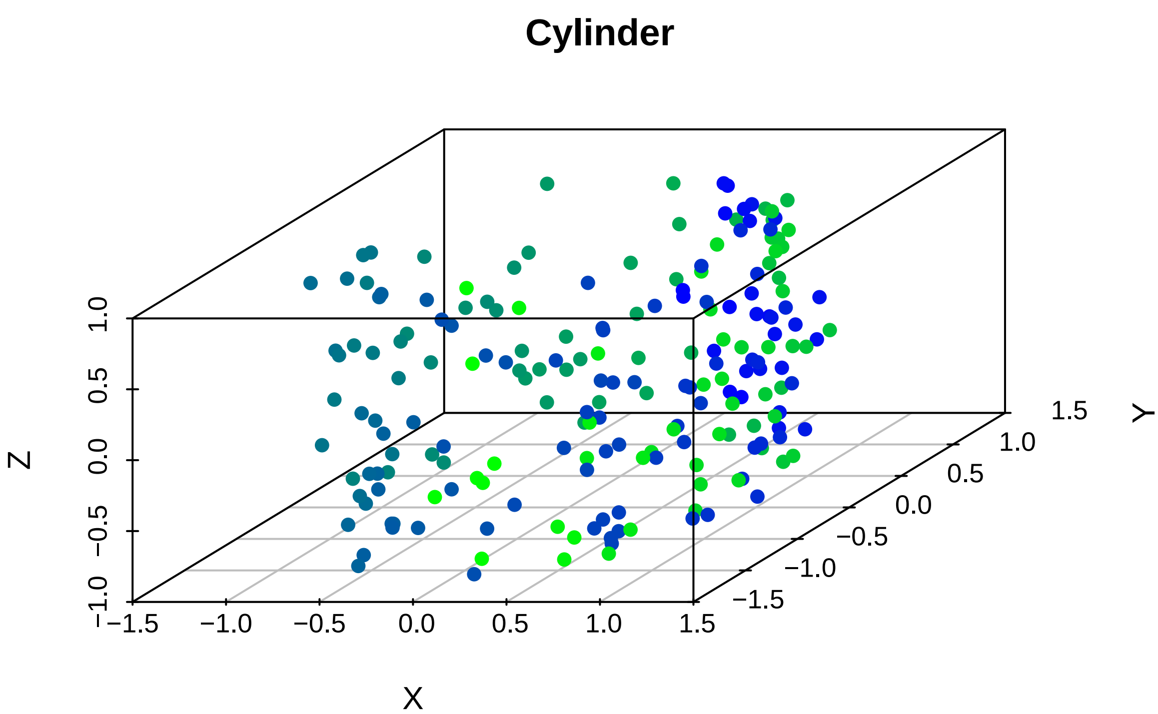
```r
  }
  return(graph)
}

eps_build_graph <- function(n, epsilon, dist_matrix){
  graph <- matrix(Inf, n, n)
  diag(graph) <- 0

  for (i in 1:n) {
    for (j in 1:n) {
      if (i != j && dist_matrix[i, j] < epsilon) {
        graph[i, j] <- dist_matrix[i, j]
      }
    }
  }
  # To ensure symmetry of the adjacency matrix
  for (i in 1:n) {
    for (j in 1:n) {
      graph[i, j] <- min(graph[i, j], graph[j, i])
      graph[j, i] <- graph[i, j]
    }
  }
  return(graph)
}

# Function that constructs the weighted complete graph with the shortest
# distances between each pair of vertices in cubic time Theta(n^3)
floyd_warshall <- function(n, graph){
  for (k_ in 1:n) {
    for (i in 1:n) {
      for (j in 1:n) {
        if (graph[i, k_] + graph[k_, j] < graph[i, j]) {
          graph[i, j] <- graph[i, k_] + graph[k_, j]
        }
      }
    }
  }
  return(graph)
}

# Performs multidimensional scaling
mds_step <- function(n, graph){
  D2 <- graph^2
  # Centering matrix
  J <- diag(n) - matrix(1, n, n)/n

  # Double center the matrix
  B <- -0.5*J %*% D2 %*% J

  return(eigen(B))
}
```

```r
# Function that performs the ISOMAP embedding by constructing the neighborhood
# graph using k nearest neighbors, and d being the number of retained eigenvalues
k_isomap_embedding <- function(data, k = 5, d = 2) {
  n <- nrow(data)

  # 1. Compute full Euclidean distance matrix
  dist_matrix <- as.matrix(dist(data))

  # 2. Build the neighborhood graph (k-nearest neighbors), implemented as an
  # adjacency matrix
  graph <- k_build_graph(n, k, dist_matrix)

  # 3. Compute geodesic distances using the Floyd-Warshall algorithm
  graph <- floyd_warshall(n, graph)

  # 4. MDS: Compute low-dimensional embedding from distance matrix
  eig <- mds_step(n, graph)

  # Extract the top d eigenvectors (and corresponding eigenvalues)
  L <- diag(sqrt(eig$values[1:d]))
  V <- eig$vectors[, 1:d]

  # The low-dimensional coordinates
  Y <- V %*% L

  return(list(embedding = Y, geodesic_distances = graph))
}
```

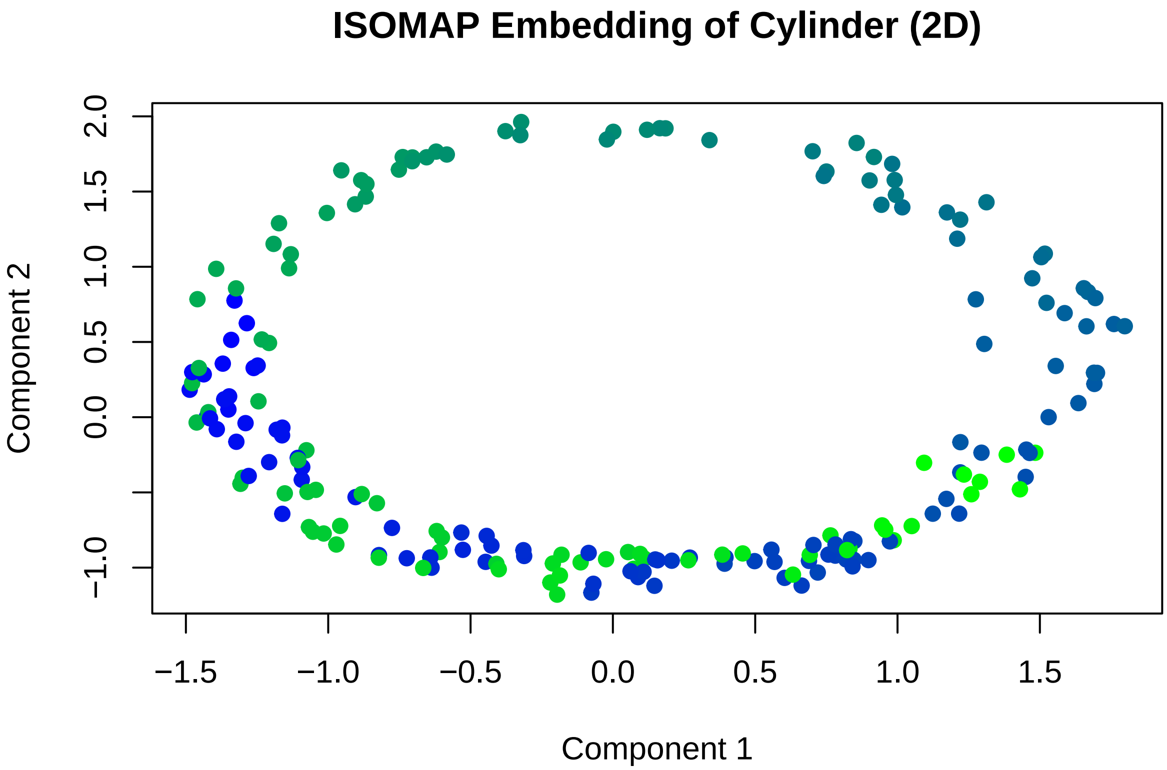
```r
eps_isomap_embedding <- function(data, epsilon, d = 2){
  n <- nrow(data)

  # 1. Compute full Euclidean distance matrix
  dist_matrix <- as.matrix(dist(data))

  # 2. Build the neighborhood graph (k-nearest neighbors), implemented as an
  # adjacency matrix
  graph <- eps_build_graph(n, epsilon, dist_matrix)

  # 3. Compute geodesic distances using the Floyd-Warshall algorithm
  graph <- floyd_warshall(n, graph)

  # 4. MDS: Compute low-dimensional embedding from distance matrix
  eig <- mds_step(n, graph)

  # Extract the top d eigenvectors (and corresponding eigenvalues)
  L <- diag(sqrt(eig$values[1:d]))
  V <- eig$vectors[, 1:d]

  # The low-dimensional coordinates
  Y <- V %*% L

  return(list(embedding = Y, geodesic_distances = graph))
```

```
}
```

```
# Example usage:
# Generate a synthetic dataset (e.g., points on a 2D manifold embedded in 3D)
set.seed(42)
n <- 200
# theta <- runif(n_points, 0, 2 * pi)
t <- (3*pi/2)*(1 + 2*runif(n))
# Points arranged as a cylinder
z <- runif(n, -1, 1)
x <- cos(t) + 0.1 * rnorm(n)
y <- sin(t) + 0.1 * rnorm(n)
data3D <- cbind(x, y, z)

norm_t <- (t - min(t)) / (max(t) - min(t))
gradient_colors <- colorRampPalette(c("green", "blue"))(n)
color_index <- round(norm_t * (n - 1)) + 1
point_colors <- gradient_colors[color_index]

#plot3d(data3D[,1], data3D[,2], data3D[,3],
#       col = point_colors,
#       size = 2,
#       type = "s",
#       xlab = "X", ylab = "Y", zlab = "Z")
#rglwidget()

scatterplot3d(data3D[,1], data3D[,2], data3D[,3],
              main = "Cylinder",
              xlab = "X", ylab = "Y", zlab = "Z",
              color = point_colors, pch = 16)
```



```r
# The low-dimensional embedding should look somewhat like a circle
result <- k_isomap_embedding(data3D, k = 10, d = 2)
#result <- eps_isomap_embedding(data3D, epsilon = 0.5, d = 2)
embedding <- result$embedding

plot(embedding, main = "ISOMAP Embedding of Cylinder (2D)", xlab = "Component 1", ylab = "Component 2",
     pch = 19, col = point_colors)
```

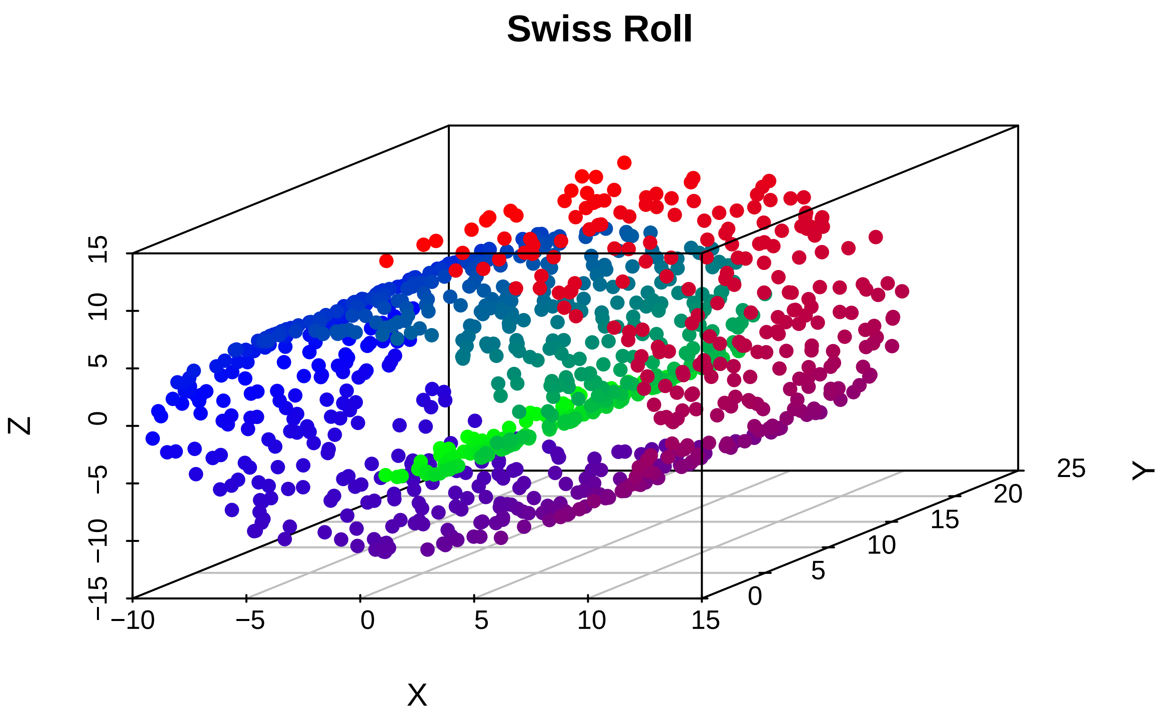


Simulations:

Swiss Roll:

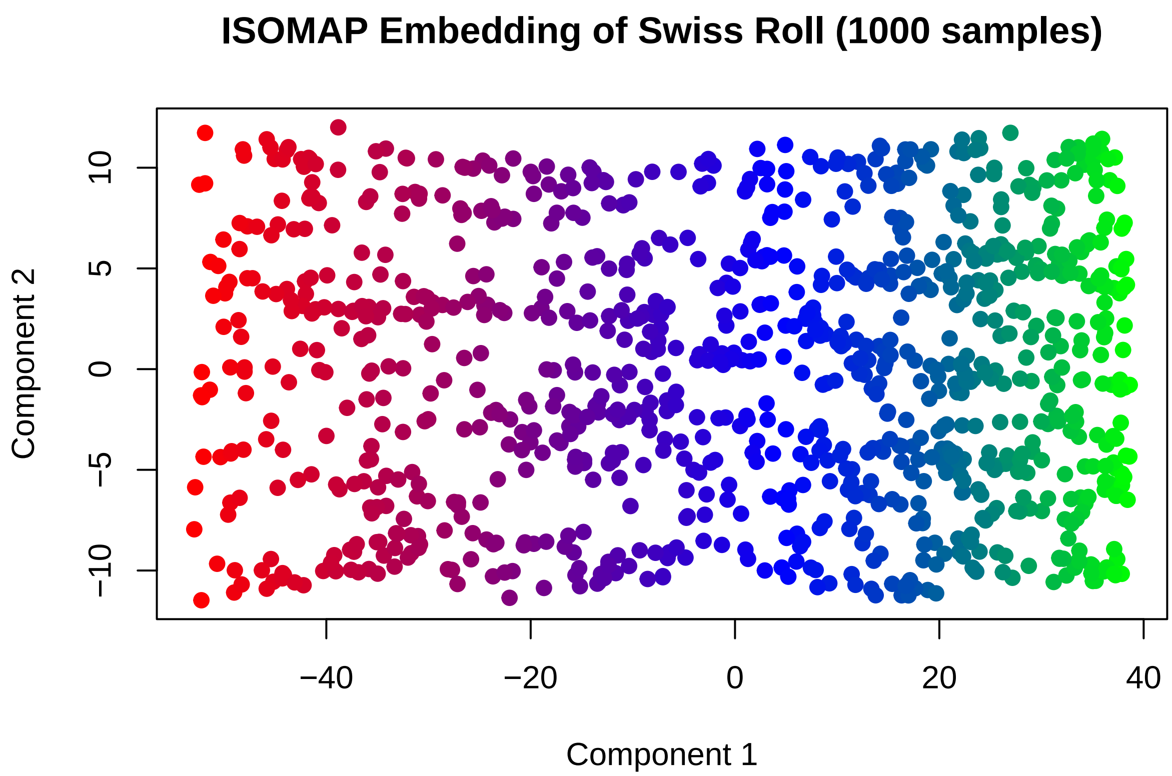
```r
set.seed(2201)

# Number of data points
n <- 1000

# Generate parameters:
# 't' controls the roll (angular parameter) and 'h' controls the height
t <- (3*pi/2)*(1 + 2*runif(n))  # Parameter that defines the roll's curvature
h <- 21*runif(n)                      # Random height values

# Compute the 3D coordinates of the Swiss roll
x <- t*cos(t)
y <- h
z <- t*sin(t)

# Combine the coordinates into a data frame
```

```r
swiss_roll <- data.frame(x = x, y = y, z = z)

# Colors of the roll points
norm_t <- (t - min(t)) / (max(t) - min(t))
gradient_colors <- colorRampPalette(c("green", "blue", "red"))(n)
color_index <- round(norm_t * (n - 1)) + 1
point_colors <- gradient_colors[color_index]

scatterplot3d(swiss_roll$x, swiss_roll$y, swiss_roll$z,
              main = "Swiss Roll",
              xlab = "X", ylab = "Y", zlab = "Z",
              color = point_colors, pch = 16)
```




```r
# Visualize the Swiss roll in 3D using rgl
#plot3d(swiss_roll$x, swiss_roll$y, swiss_roll$z,
#       col = point_colors,
#       size = 2,
#       type = "s",
#       xlab = "X", ylab = "Y", zlab = "Z")
#rglwidget()
```

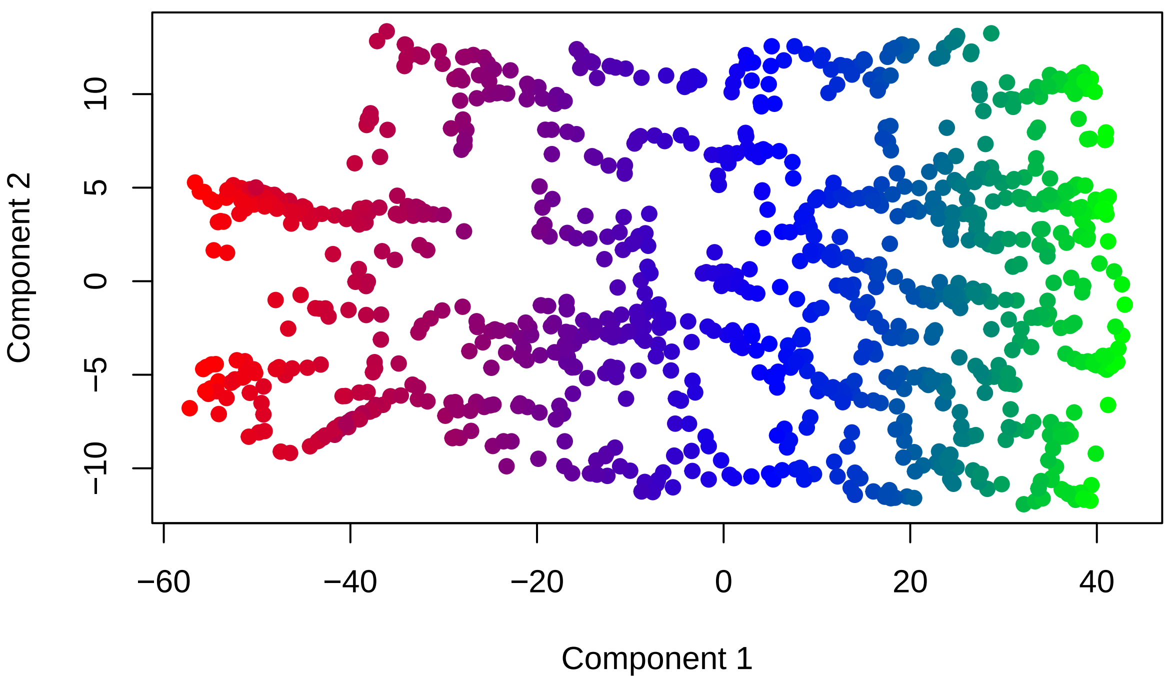
```r
result <- k_isomap_embedding(swiss_roll, k = 10, d = 2)
#result <- eps_isomap_embedding(swiss_roll, eps = 3.5, d = 3)
swiss_roll_embedding <- result$embedding

plot(swiss_roll_embedding,  main = "ISOMAP Embedding of Swiss Roll (1000 samples)",
     xlab = "Component 1", ylab = "Component 2",
     col = point_colors,
     pch = 19)
```



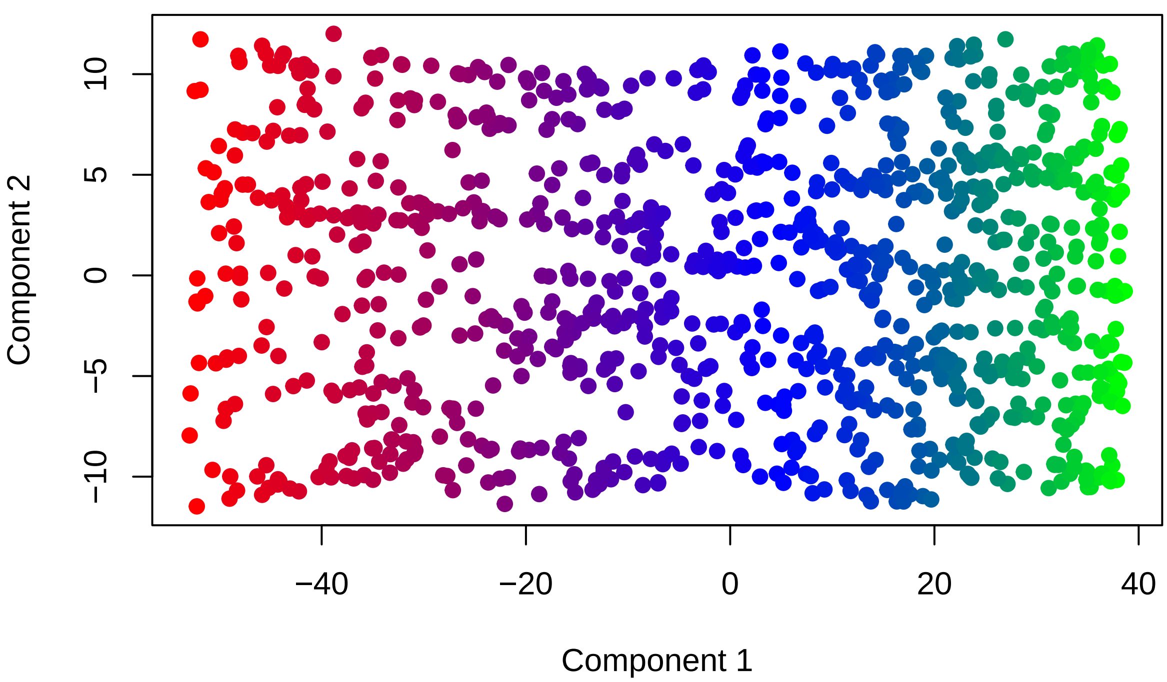

```r
#plot3d(swiss_roll_embedding[,1], swiss_roll_embedding[,2], swiss_roll_embedding[,3],
#       col = point_colors,
#       size = 2,
#       type = "s",
#       xlab = "X", ylab = "Y", zlab = "Z")
#rglwidget()
```

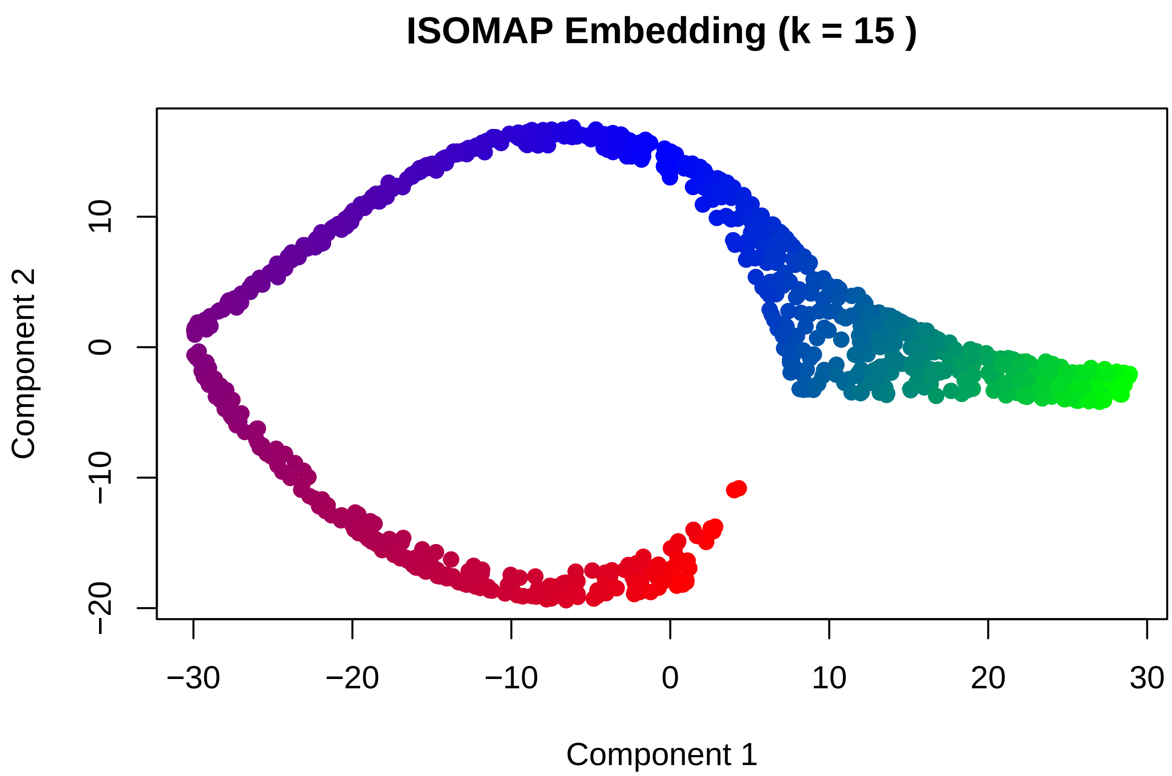
```r
# Testing range of k-values
k_range <- c(5, 10, 15, 20)
embeddings_list <- list()

embeddings_list <- setNames(lapply(k_range, function(k_val) {
  k_isomap_embedding(swiss_roll, k = k_val, d = 2)$embedding
}), k_range)

for (k in names(embeddings_list)) {
  embedding <- embeddings_list[[k]]
  plot(embedding,
       main = paste("ISOMAP Embedding (k =", k, ")"),
       xlab = "Component 1", ylab = "Component 2",
       col = point_colors, pch = 19)
}
```

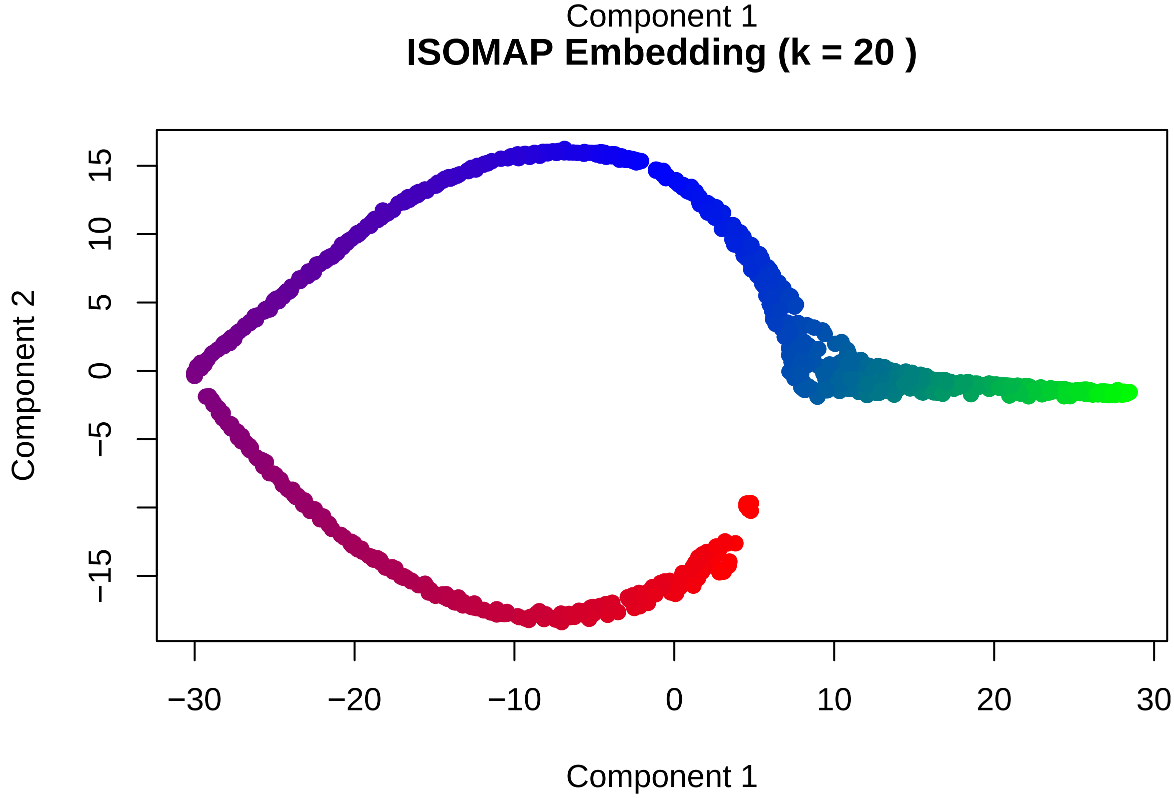
ISOMAP Embedding (k = 5)



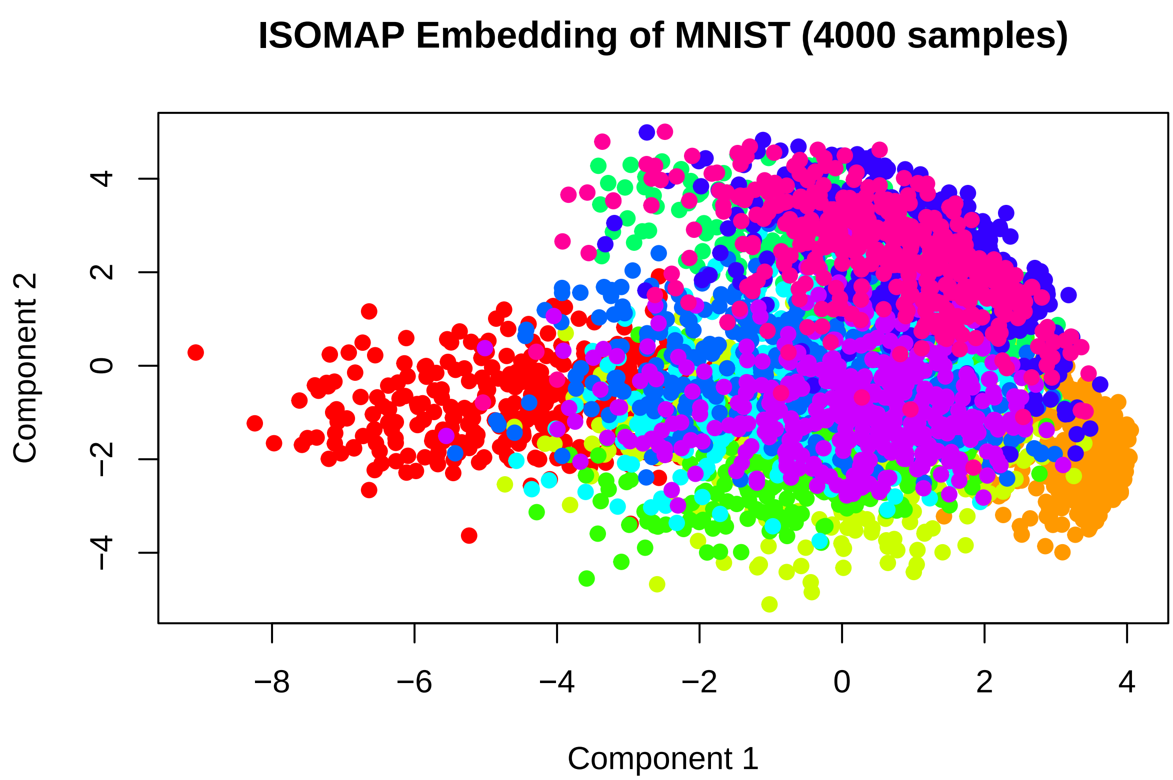
ISOMAP Embedding (k = 10)




ISOMAP Embedding (k = 15)
Component 2
10
0
−10
−20
−30
−20
−10
0
10
20
30
Component 1

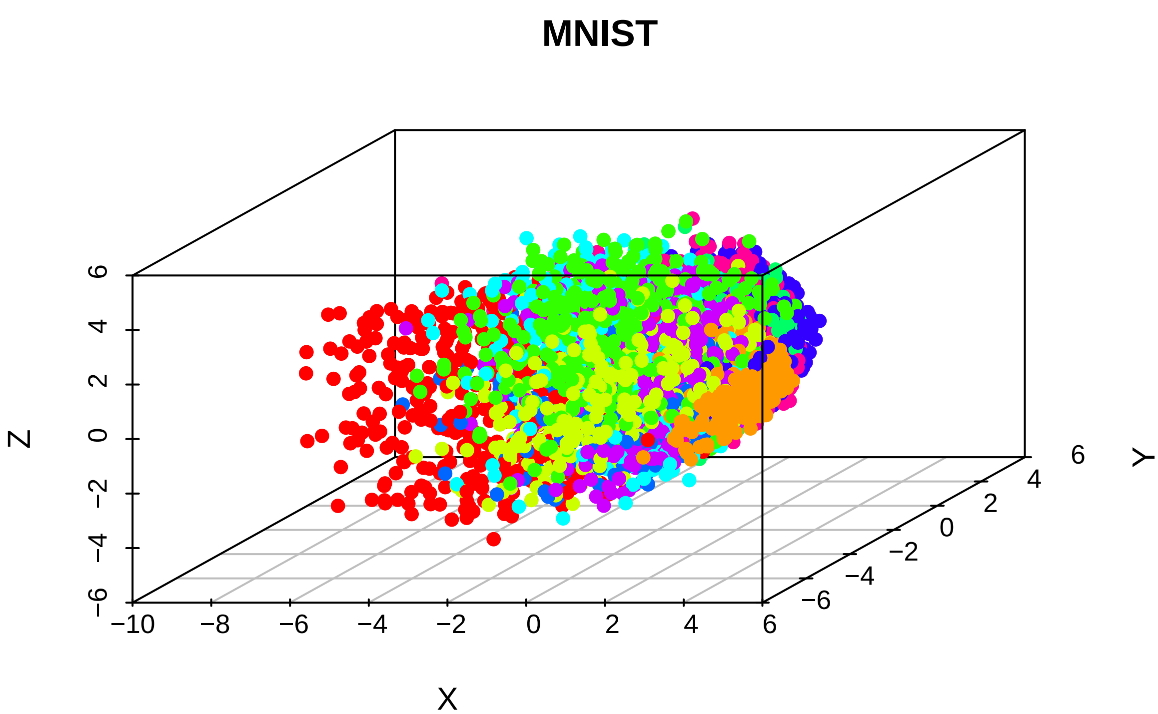

ISOMAP Embedding (k = 20)
Component 2
15
10
5
0
−5
−15
−30
−20
−10
0
10
20
30
Component 1

```r
mnist <- keras::dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y

x_train <- array_reshape(x_train, c(nrow(x_train), 28*28))

# Select a subset of data points to make this faster (max is 60k)
set.seed(123)  # for reproducibility
n_each <- 400  # number of images per digit to keep labels balanced

indices <- c()
for (digit in 0:9) {
  indices_ <- which(mnist$train$y == digit)
  indices <- c(indices, sample(indices_, n_each))
}

X <- mnist$train$x[indices, , ]
X <- array_reshape(X, c(10*n_each, 28*28))/255
labels <- mnist$train$y[indices]

result <- eps_isomap_embedding(X, epsilon = 20, d = 30)
mnist_embedding <- result$embedding

# Plot the 2D embedding, coloring points by their digit labels.
# We use a color palette that maps the 10 digit classes.
plot(mnist_embedding,
     main = "ISOMAP Embedding of MNIST (4000 samples)",
     xlab = "Component 1", ylab = "Component 2",
     pch = 19, col = rainbow(10)[labels + 1])
```



MNIST:

```r
scatterplot3d(mnist_embedding[,1], mnist_embedding[,2], mnist_embedding[,3],
              main = "MNIST",
              xlab = "X", ylab = "Y", zlab = "Z",
              color = rainbow(10)[labels + 1], pch = 16)
```




```r
#plot3d(mnist_embedding[,1], mnist_embedding[,2], mnist_embedding[,3],
#       col = rainbow(10)[labels + 1],
```

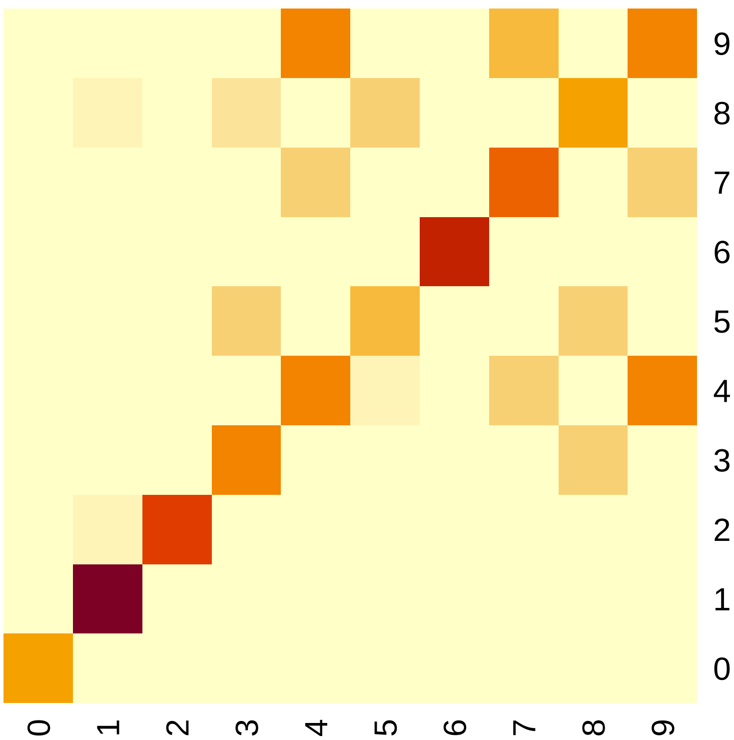
```
#       size = 2,
#       type = "s",
#       xlab = "X", ylab = "Y", zlab = "Z")
#rglwidget()
```

```r
set.seed(2201)
kmeans_result <- kmeans(mnist_embedding, centers = 10)
clusters <- kmeans_result$cluster - 1

aux_confusion_matrix <- table(True = labels, Cluster = clusters)
confusion_matrix <- aux_confusion_matrix[, max.col(aux_confusion_matrix, 'first')]
colnames(confusion_matrix) = seq(0,9)

print("Confusion Matrix:")
```

K-Means:

```
## [1] "Confusion Matrix:"
```

```r
#print(confusion_matrix)
heatmap(confusion_matrix, Colv=NA, Rowv=NA, scale='none')
```




```r
sensitivity <- diag(confusion_matrix) / rowSums(confusion_matrix)
print("Sensitivity for each class:")
```

```
## [1] "Sensitivity for each class:"
```

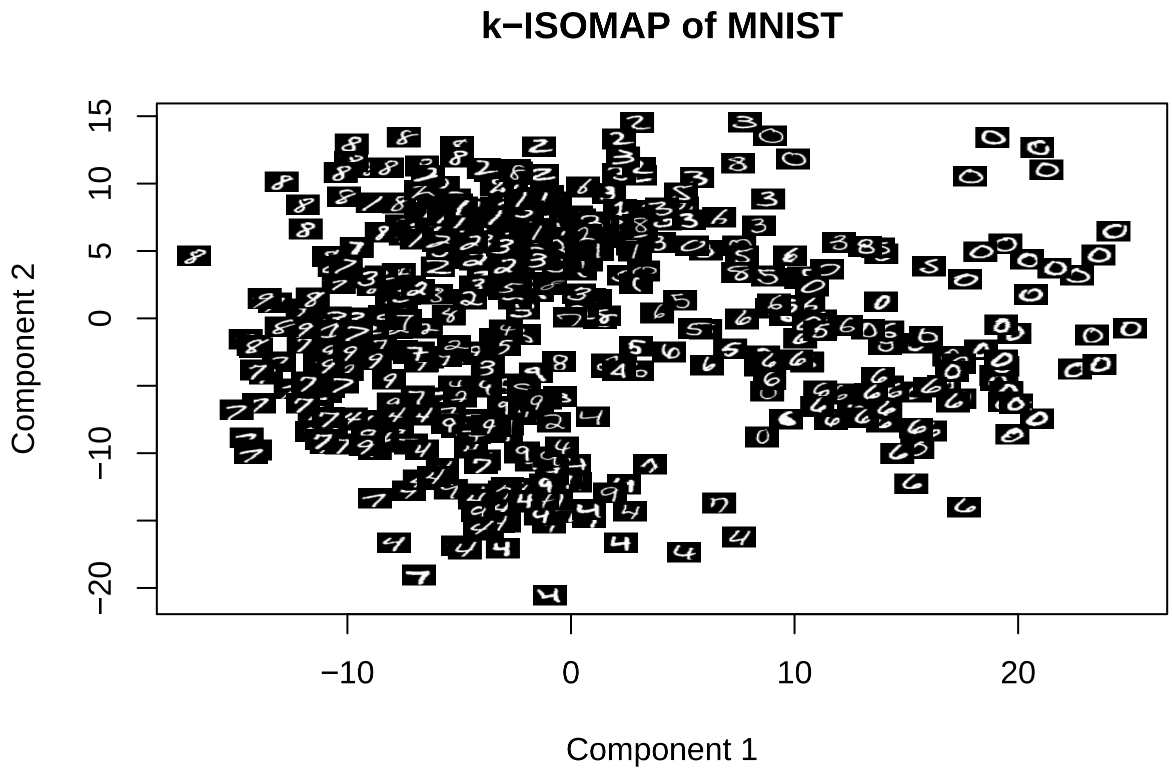
```r
print(sensitivity)
```

```
##         0         1         2         3         4         5         6         7
## 0.8280543 0.9328358 0.6715686 0.5170732 0.3517018 0.2978723 0.7819549 0.4506770
##         8         9
## 0.4411765 0.3383333
```

```r
average_sensitivity <- mean(sensitivity)
print(paste("Average Sensitivity:", round(average_sensitivity*100, 2), "%"))
```

```
## [1] "Average Sensitivity: 56.11 %"
```

```r
# Select a subset for speed (e.g., 500 images)
n <- 500
X <- mnist$train$x[1:n, , ]
# Reshape to 784-dimensional vectors and normalize
X <- array_reshape(X, c(n, 28*28)) / 255
labels <- mnist$train$y[1:n]

# Compute the k-Isomap embedding
result <- k_isomap_embedding(X, k = 10, d = 2)
embedding <- result$embedding

# Define a helper function to convert a digit vector into a raster image.
convert_to_raster <- function(vec, img_dim = 28) {
  # Reshape the vector into a matrix with img_dim rows.
  m <- matrix(vec, nrow = img_dim, byrow = TRUE)
  # Transpose the matrix and reverse the row order to rotate 90° counterclockwise.
  m <- m[1:img_dim, ]
  # Convert the grayscale matrix to colors using the gray scale and return as a raster.
  as.raster(matrix(gray(m), nrow = nrow(m), ncol = ncol(m)))
}

# Plot the 2D embedding without points (reserve space for images)
plot(embedding, type = "n", main = "k-ISOMAP of MNIST",
     xlab = "Component 1", ylab = "Component 2")

# Choose a scaling factor for the thumbnail images (adjust as needed)
img_size <- 0.8*diff(range(embedding[,1]))/sqrt(n)

# Overlay each image on the plot
for (i in 1:n) {
  img_raster <- convert_to_raster(X[i, ])
  x <- embedding[i, 1]
  y <- embedding[i, 2]
  rasterImage(img_raster, x - img_size/2, y - img_size/2, x + img_size/2, y + img_size/2)
}
```




```r
n <- 300  # number of images to use
X <- mnist$train$x[1:n, , ]
X <- array_reshape(X, c(n, 28*28))/255  # flatten and normalize
labels <- mnist$train$y[1:n]

# Compute the k-ISOMAP embedding on the entire subset (all labels included)
result <- k_isomap_embedding(X, k = 10, d = 2)
embedding <- result$embedding

# ----------------------------------------------------------
# Visualization: only overlay images for a chosen label, here "2"

# Plot the entire embedding as a background (optional)
plot(embedding, type = "n",
     main = "k-Isomap Embedding (Only Label '5' Images Overlaid)",
     xlab = "Component 1", ylab = "Component 2")
points(embedding, pch = 19, col = "lightgray")

# Helper function to convert a digit vector into a correctly oriented raster image
convert_to_raster <- function(vec, img_dim = 28) {
  m <- matrix(vec, nrow = img_dim, byrow = TRUE)
  # Transpose and reverse row order to rotate 90° counterclockwise
  m <- m[1:img_dim, ]
  as.raster(matrix(gray(m), nrow = nrow(m), ncol = ncol(m)))
}

# Choose the target label
target_label <- 5
```

```
indices <- which(labels == target_label)

# Choose a thumbnail size (adjust as needed)
img_size <- 0.8*diff(range(embedding[, 1]))/sqrt(n)

# Overlay the raster images only for points with the target label
for (i in indices) {
  img_raster <- convert_to_raster(X[i, ])
  x <- embedding[i, 1]
  y <- embedding[i, 2]
  rasterImage(img_raster, x - img_size/2, y - img_size/2,
              x + img_size/2, y + img_size/2)
}
```