DSA 8070 R Session 14: Nonlinear Dimensionality Reduction and Manifold Learning

Whitney Huang, Clemson University

Contents

Recap: Linear PCA

Principal Component Analysis (PCA) finds an orthogonal linear transformation of the data that:

- maximizes variance along successive components, and
- equivalently, minimizes reconstruction error in a low-dimensional linear subspace.

If X is an $n \times p$ data matrix (columns centered), PCA finds eigenvalues $\lambda_1 \geq \cdots \geq \lambda_p$ and eigenvectors $v_1, \ldots, v_p$ of the covariance matrix. The first k eigenvectors span the best k-dimensional linear subspace for the data.

However, when the data lie on a **nonlinear manifold** (e.g., a Swiss roll), any global linear subspace is a poor approximation. This motivates **nonlinear PCA**.

Polynomial PCA

Idea

Polynomial PCA generalizes PCA by **explicitly expanding** the feature space with polynomial terms (quadratic, cubic, etc.), and then applying ordinary linear PCA to the expanded features.

For example, with two variables X_1, X_2, a quadratic expansion is

$$X' = (X_1, X_2, X_1^2, X_2^2, X_1 X_2).$$

We then run PCA on X'. If the true relationship between X_1 and X_2 is approximately quadratic, the smallest eigenvalues of $\mathrm{Cov}(X')$ correspond to directions representing the nonlinear constraint.

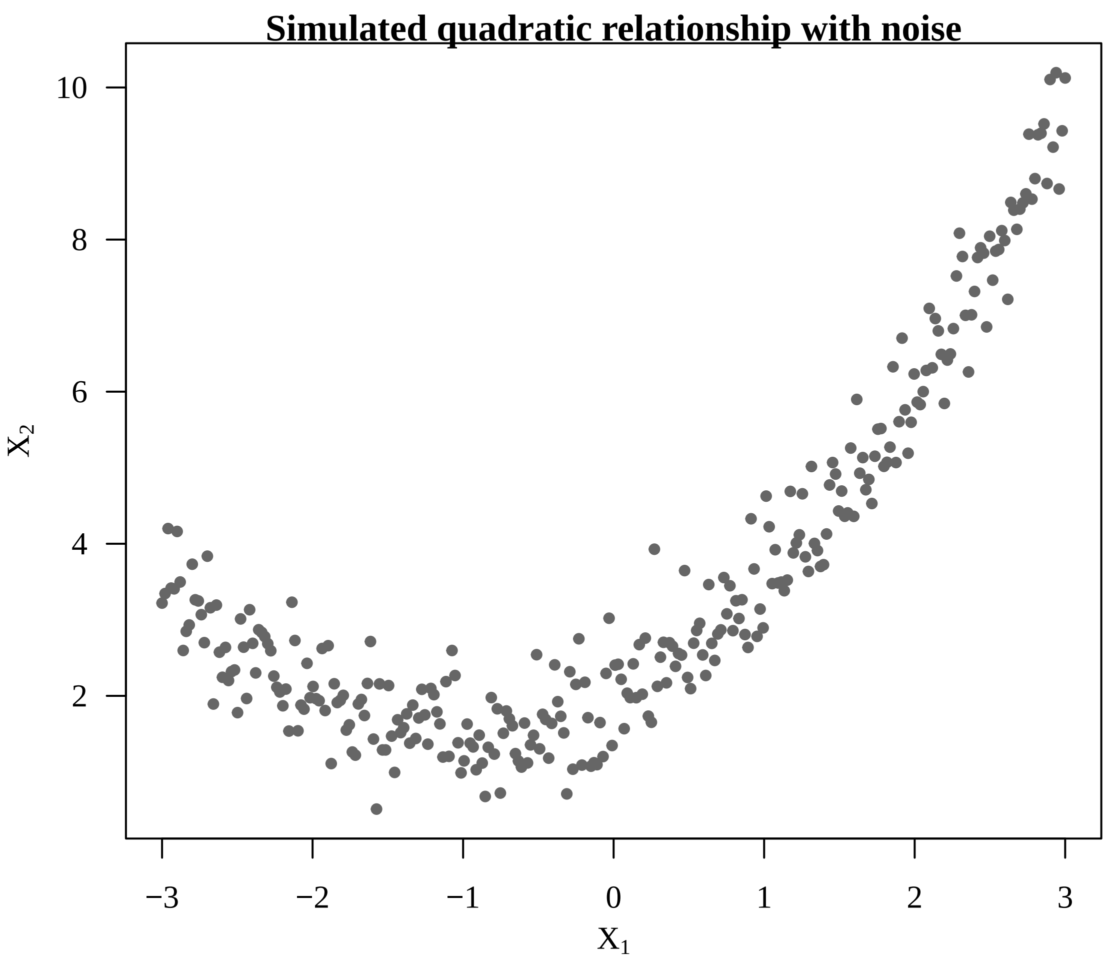
Example: Noisy quadratic curve

We simulate bivariate data lying near a quadratic curve, then compare linear PCA and quadratic PCA.

```r
set.seed(123)

n <- 300
x1 <- seq(-3, 3, length.out = n)
x2_true <- 0.5 * x1^2 + x1 + 2
x2 <- x2_true + rnorm(n, sd = 0.5)

dat <- data.frame(x1 = x1, x2 = x2)
head(dat)
```

```
##          x1       x2
## 1 -3.000000 3.219762
## 2 -2.979933 3.344979
## 3 -2.959866 4.199892
## 4 -2.939799 3.416665
## 5 -2.919732 3.407330
## 6 -2.899666 4.161897
```

```r
par(mgp = c(2.2, 1, 0), mar = c(3.5, 4, 0.8, 0.6), las = 1,
    family = "serif")
plot(dat$x1, dat$x2, pch = 20, col = "gray40",
     xlab = expression(X[1]), ylab = expression(X[2]),
     main = "Simulated quadratic relationship with noise")
```

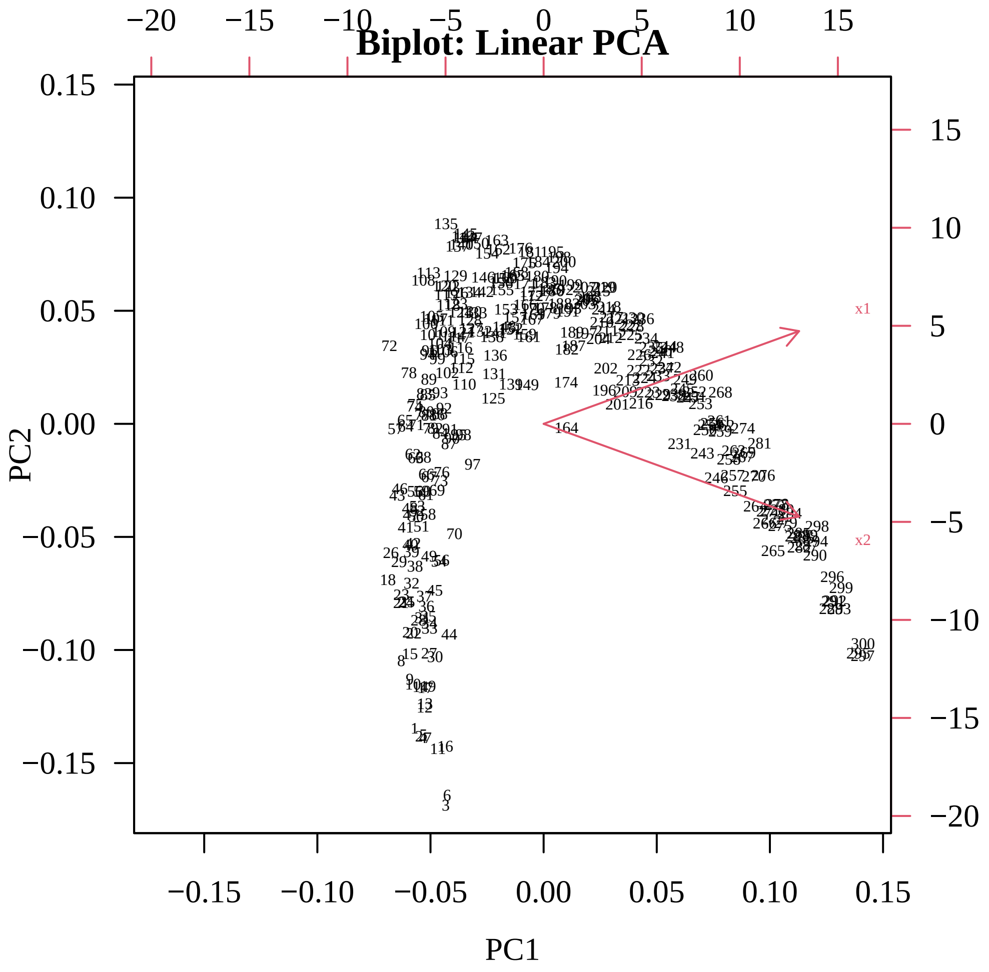


```r
X <- scale(dat)  # center and scale

pca_lin <- prcomp(X, center = FALSE, scale. = FALSE)
summary(pca_lin)
```

Linear PCA

```
## Importance of components:
##                           PC1    PC2
## Standard deviation     1.3293 0.4827
## Proportion of Variance 0.8835 0.1165
## Cumulative Proportion  0.8835 1.0000
```

```r
par(mgp = c(2.5, 1, 0), mar = c(3.5, 4, 0.8, 0.6), las = 1, family = "serif")
biplot(pca_lin, main = "Biplot: Linear PCA", cex = 0.5)
```



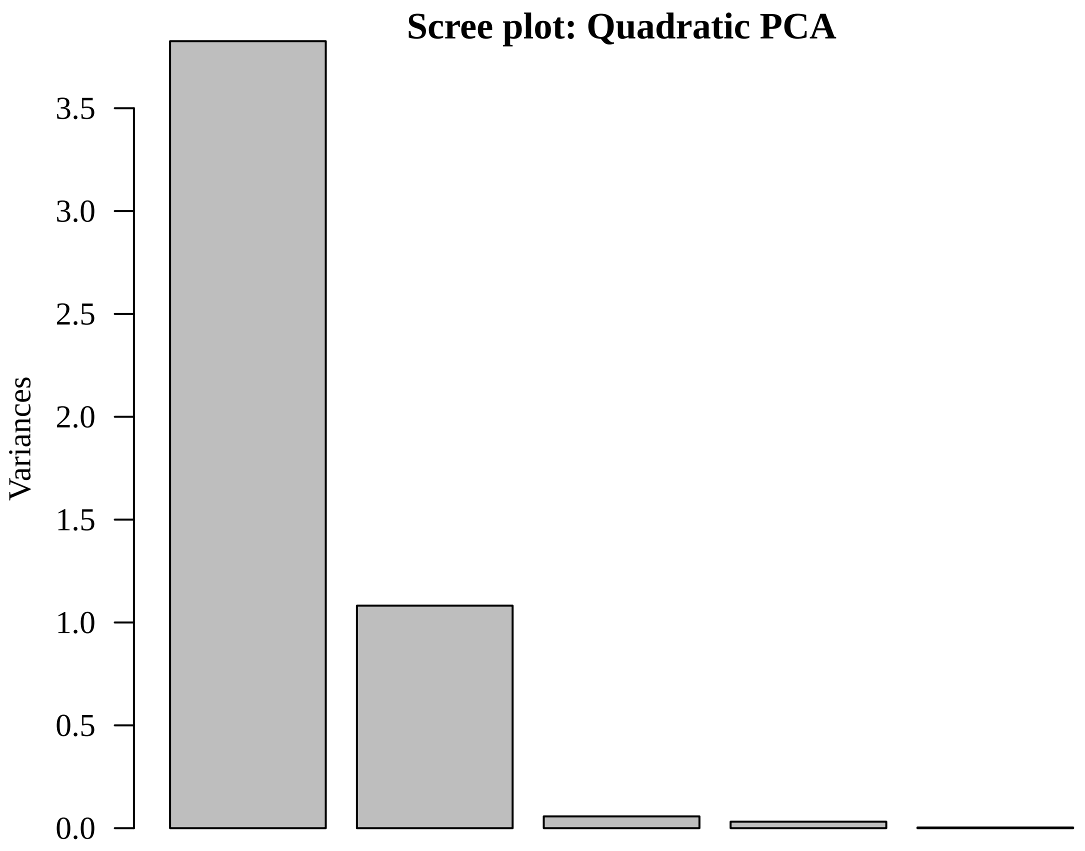
Linear PCA sees only an elongated cloud; it cannot capture the underlying quadratic functional form.

```r
# Build quadratic features: x1, x2, x1^2, x2^2, x1*x2
X_quad <- with(dat, cbind(x1, x2, x1_sq = x1^2,
                          x2_sq = x2^2, x1x2 = x1 * x2))

X_quad_scaled <- scale(X_quad)   # center/scale

pca_quad <- prcomp(X_quad_scaled, center = FALSE, scale. = FALSE)
summary(pca_quad)
```

Quadratic PCA

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5
## Standard deviation     1.9560 1.0401 0.23961 0.17802 0.05741
## Proportion of Variance 0.7652 0.2163 0.01148 0.00634 0.00066
## Cumulative Proportion  0.7652 0.9815 0.99300 0.99934 1.00000
```

```r
par(mgp = c(2.5, 1, 0), mar = c(3.5, 4, 0.8, 0.6), las = 1,
    family = "serif")
plot(pca_quad, main = "Scree plot: Quadratic PCA")
```



Kernel PCA

Polynomial PCA requires explicit feature construction and can quickly become high-dimensional. Kernel PCA avoids this by mapping data to a high-dimensional feature space **implicitly** via a kernel function and then performing PCA there.

A popular choice is the **Gaussian (RBF) kernel**:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right).$$

We will apply kernel PCA to the 3D Swiss roll data and visualize the 2D embedding, and compare it to linear PCA.

```r
library(kernlab)
library(scatterplot3d)
```

Simulate Swiss roll data

The **Swiss roll** is a classic example in manifold learning, illustrating how high-dimensional data can lie on a lower-dimensional, curved surface. In this example, we generate 3D points following a spiral structure defined by:

$$x = t\cos(t), \quad y = h, \quad z = t\sin(t),$$

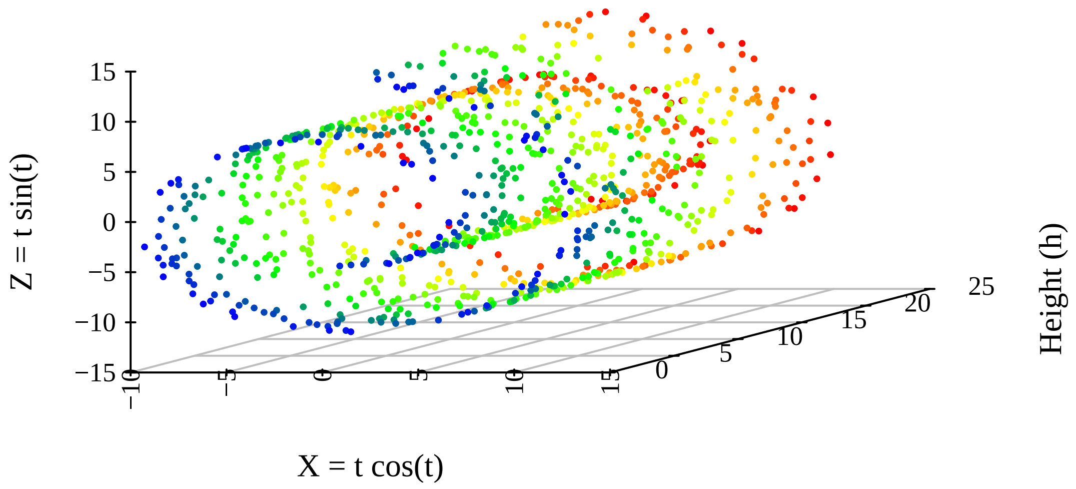
where t controls the spiral angle and h represents the vertical height.

Each point is colored by its height (h), producing a smooth gradient along the roll.

```r
set.seed(123)

n <- 1000
t <- 3 * pi / 2 * (1 + 2 * runif(n))
h <- 21 * runif(n)
x <- t * cos(t)
y <- h
z <- t * sin(t)

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

```
##           x         y         z         h
## 1  3.102678  5.746077  6.743182  5.746077
## 2 11.064921 12.471206 -4.999544 12.471206
## 3 -5.603171  3.363881  6.480459  3.363881
## 4 11.631511 17.922035  5.882992 17.922035
## 5  7.223877 17.802522 11.494593 17.802522
## 6  2.140453 10.035623 -4.675045 10.035623
```

```r
swiss_palette <- colorRampPalette(c("blue", "green", "yellow", "red"))
cols <- swiss_palette(100)[cut(swiss_roll$h, 100)]

par(mar = c(4, 5, 1, 1), las = 2, family = "serif")
scatterplot3d(x = swiss_roll$x, y = swiss_roll$y,
  z = swiss_roll$z, color = cols,
  pch = 20, cex.symbols = 0.5,
  angle = 30, xlab = "X = t cos(t)", ylab = "Height (h)",
  zlab = "Z = t sin(t)", box = FALSE, grid = TRUE)
```



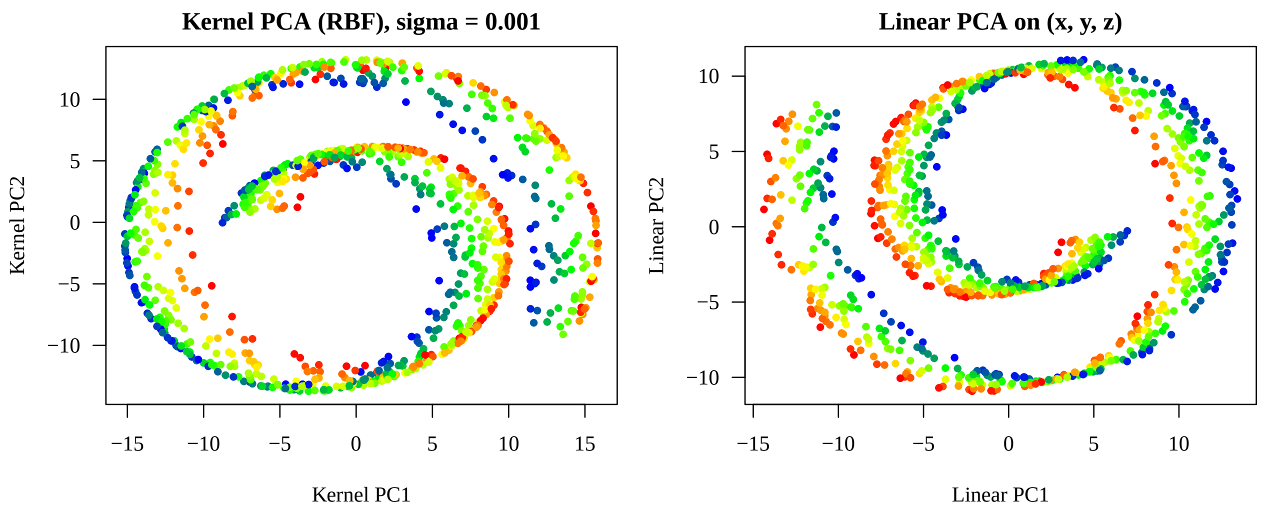
Kernel PCA with Gaussian kernel

We apply kernel PCA with an RBF kernel. The bandwidth σ controls how local the notion of similarity is.

```r
X3 <- as.matrix(swiss_roll[, c("x", "y", "z")])
# Choose a sigma value (you can experiment with others)
sigma_val <- 0.001

kpc <- kpca(x = X3, kernel = "rbfdot",
            kpar = list(sigma = sigma_val), features = 3)
```

Extract the first two kernel principal components and visualize:

```r
pc_scores <- rotated(kpc)[, 1:2]

# For comparison: linear PCA on the original (x,y,z)
pca_lin_3d <- prcomp(X3, center = TRUE, scale. = FALSE)
lin_scores <- pca_lin_3d$x[, 1:2]

par(mfrow = c(1, 2), mar = c(4, 4, 2, 1), family = "serif", las = 1)

plot(pc_scores[, 1], pc_scores[, 2], col = cols, pch = 20,
  xlab = "Kernel PC1", ylab = "Kernel PC2",
  main = paste("Kernel PCA (RBF), sigma =", sigma_val))

plot(lin_scores[, 1], lin_scores[, 2], col = cols, pch = 20,
  xlab = "Linear PC1", ylab = "Linear PC2",
  main = "Linear PCA on (x, y, z)")
```



Discussion prompts:

- How do the kernel PCA coordinates differ qualitatively from the linear PCA coordinates?
- Does kernel PCA appear to "unfold" the Swiss roll more than linear PCA?
- Try different values of `sigma_val` (e.g., `1e-4`, `5e-4`, `5e-3`, `1e-2`). How does the embedding change as σ varies?

You can experiment by running:

```
for (sigma_val in c(1e-4, 5e-4, 1e-3, 5e-3)) {
  kpc <- kpca(x = X3, kernel = "rbfdot", kpar = list(sigma = sigma_val),
              features = 2)
  pc_scores <- rotated(kpc)

  plot(pc_scores[, 1], pc_scores[, 2], col = cols, pch = 20,
    xlab = "Kernel PC1", ylab = "Kernel PC2",
    main = paste("Kernel PCA: sigma =", sigma_val))
}
```

Summary and Assessment Ideas

- **Polynomial PCA** performs PCA on explicitly constructed polynomial features. It is conceptually simple but can create very high-dimensional feature spaces.
- **Kernel PCA** performs PCA in an implicitly defined feature space using kernels. It often captures nonlinear structure more flexibly and is closely related to other kernel methods.

Nonlinear Manifold Learning

Next, we explore manifold learning and nonlinear embedding methods using both **simulated** and **real** datasets. Methods covered include:

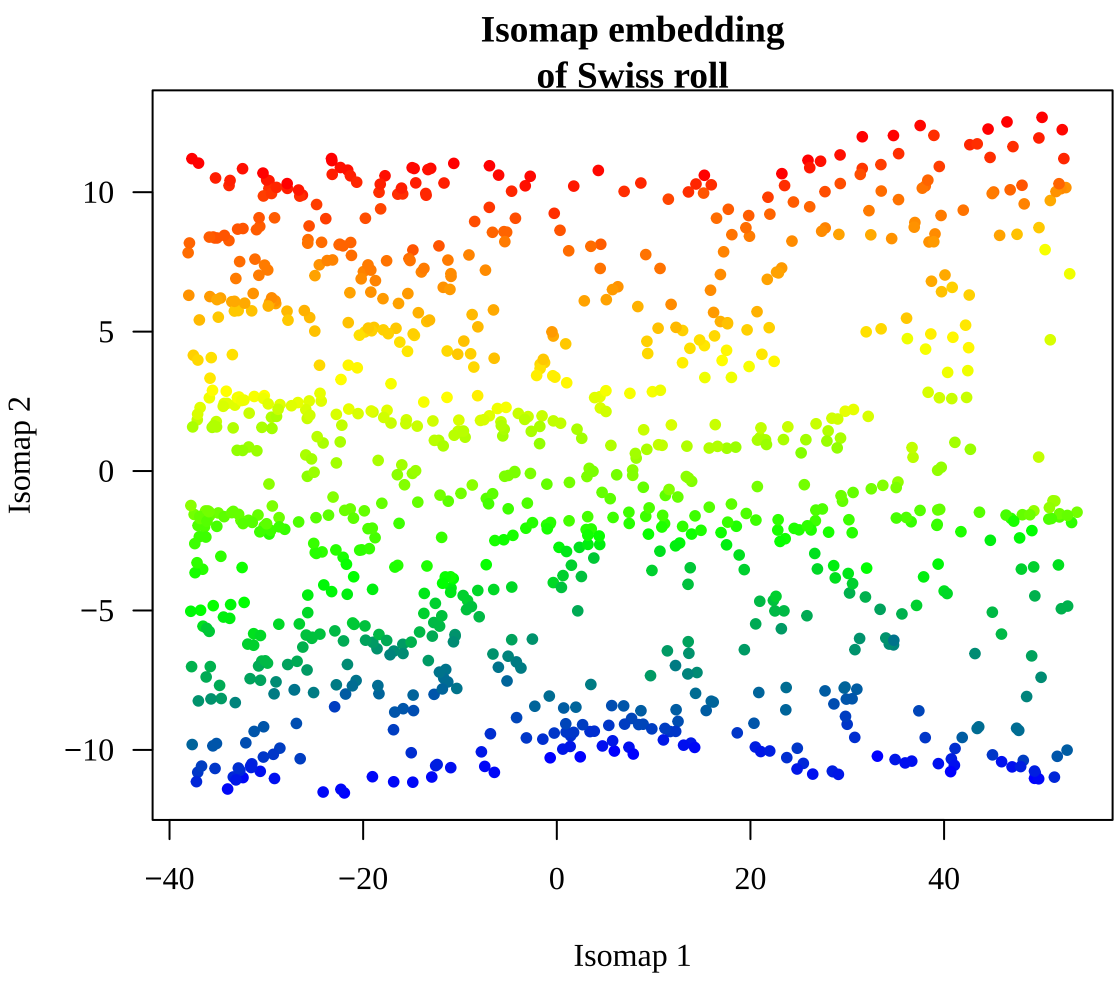
- Isometric Feature Mapping (Isomap)
- Locally Linear Embedding (LLE)
- t-SNE
- UMAP

Here, the goal is to visualize the nonlinear structure that algorithms such as `Isomap` or `Locally Linear Embedding (LLE)` attempt to 'unfold' into a low-dimensional embedding for the `Swiss roll` data.

Isomap

```
library(dimRed)

# Use only the 3D coordinates as input
X <- swiss_roll[, c("x", "y", "z")]

# Convert to dimRedData object
dat <- as(X, "dimRedData")

# Run Isomap: knn = 10, 2D embedding
iso_emb <- embed(dat, "Isomap", knn = 10, ndim = 2)

# Extract 2D embedded coordinates
iso_coords <- getData(getDimRedData(iso_emb))

# Color by height h
```

```r
par(mar = c(4, 4, 2, 1), las = 1, family = "serif")

plot(iso_coords[, 1], iso_coords[, 2], col = cols, pch = 20,
  xlab = "Isomap 1", ylab = "Isomap 2",
  main = "Isomap embedding\nof Swiss roll")
```



Locally Linear Embedding (LLE)

```r
library(Rdimtools)

# Rdimtools wants a numeric matrix: rows = samples, cols = features
X <- as.matrix(swiss_roll[, c("x", "y", "z")])
dim(X)
```

```
## [1] 1000    3
```

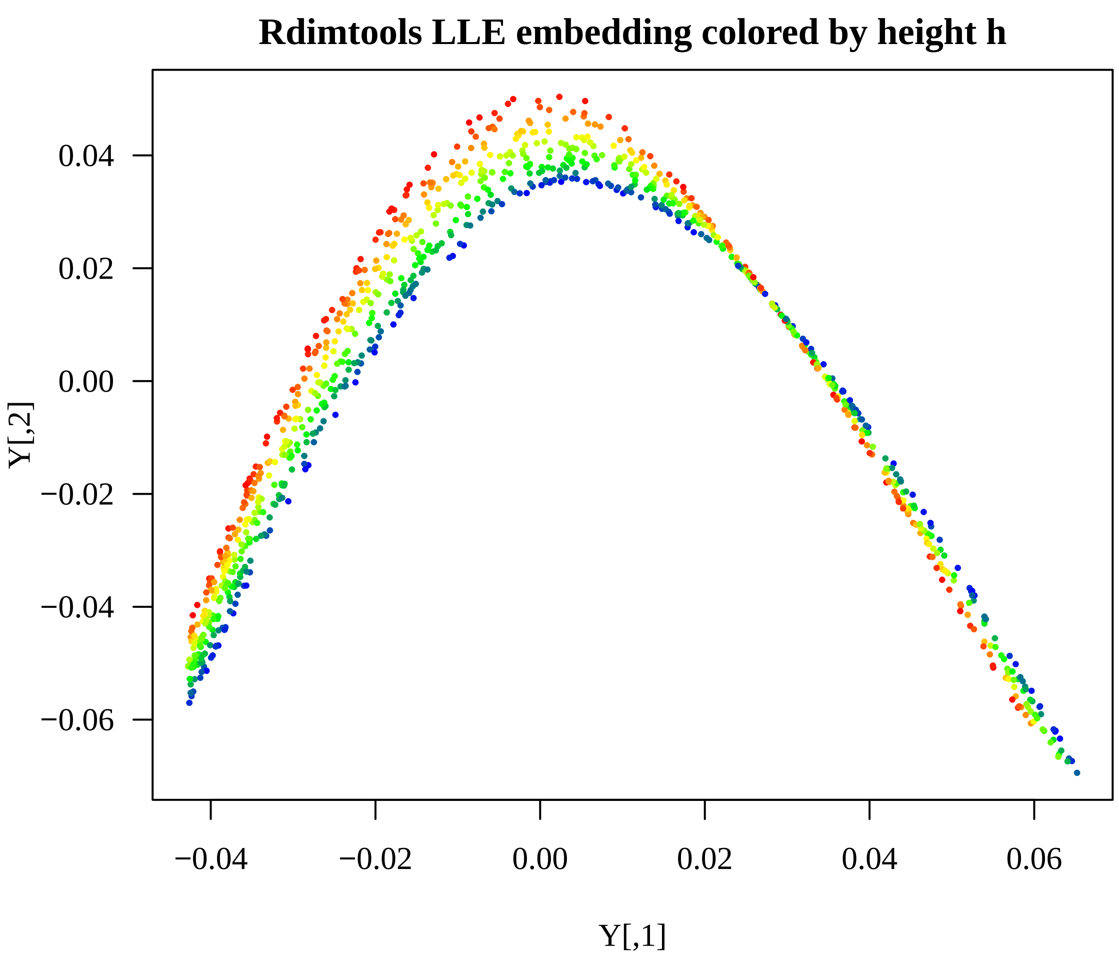
```r
## 2. LLE with Rdimtools
set.seed(123)  # for reproducibility

out_lle <- do.lle(X = X,
  ndim       = 2,                # target dimension (2D)
  type       = c("knn", 12),     # k = 12 neighbors
  symmetric  = "union", weight = TRUE, preprocess = "null",
  regtype    = TRUE, regparam   = 1)

str(out_lle)
```

```
## List of 3
##  $ Y      : num [1:1000, 1:2] -0.0262 0.032 -0.0155 0.0433 0.0511 ...
##  $ trfinfo:List of 4
##   ..$ type      : chr "null"
##   ..$ mean      : num [1:3] 0 0 0
##   ..$ multiplier: num 1
##   ..$ algtype   : chr "nonlinear"
##  $ eigvals: num [1:2] 1.07e-08 4.04e-07
```

```r
Y <- out_lle$Y   # low-dimensional embedding (n x 2)
dim(Y)
```

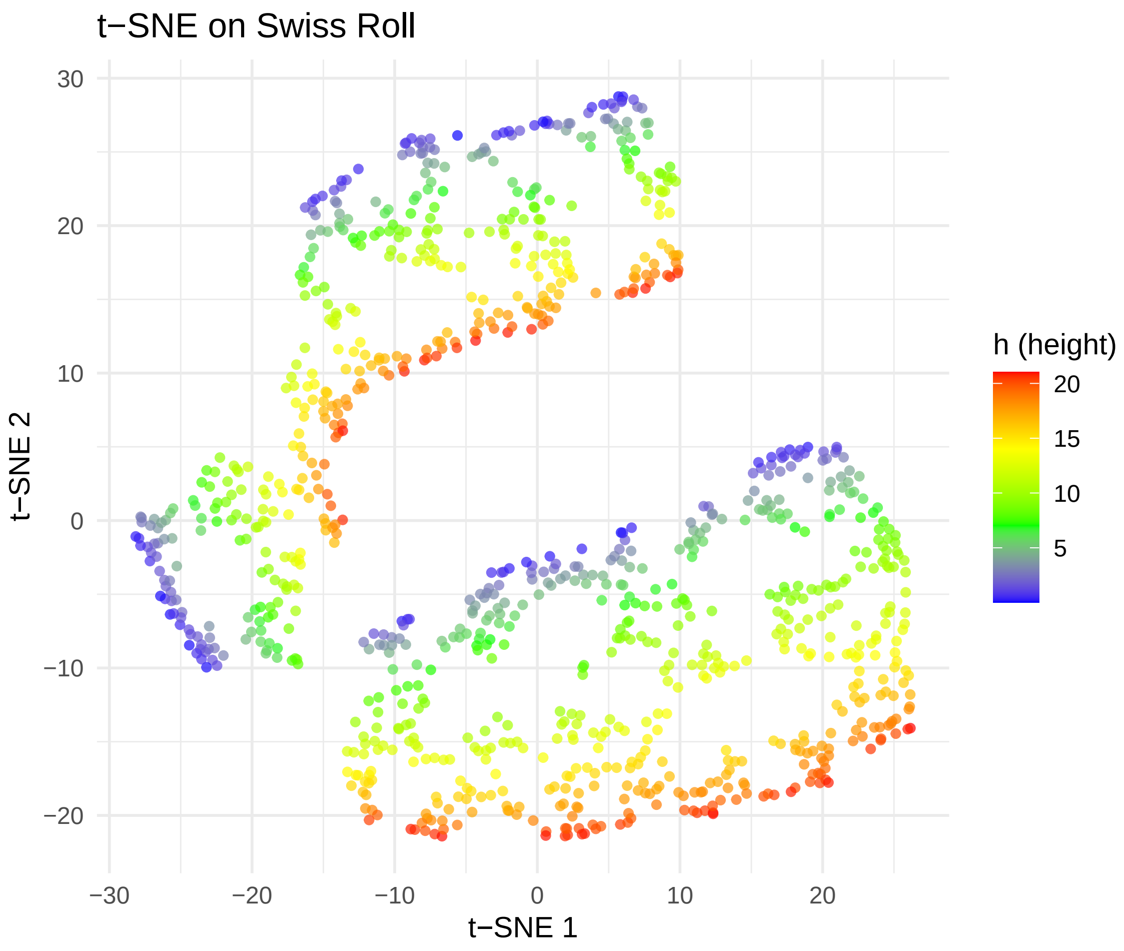
```
## [1] 1000    2
```

```r
## 3. Plots
par(mar = c(4, 4, 2, 1), las = 1, family = "serif")

cols <- colorRampPalette(c("blue", "green", "yellow", "red"))(nrow(X))
ord  <- rank(swiss_roll$h)   # ordering based on h

plot(Y, col  = cols[ord], pch  = 19, cex  = 0.3,
     main = "Rdimtools LLE embedding colored by height h")
```



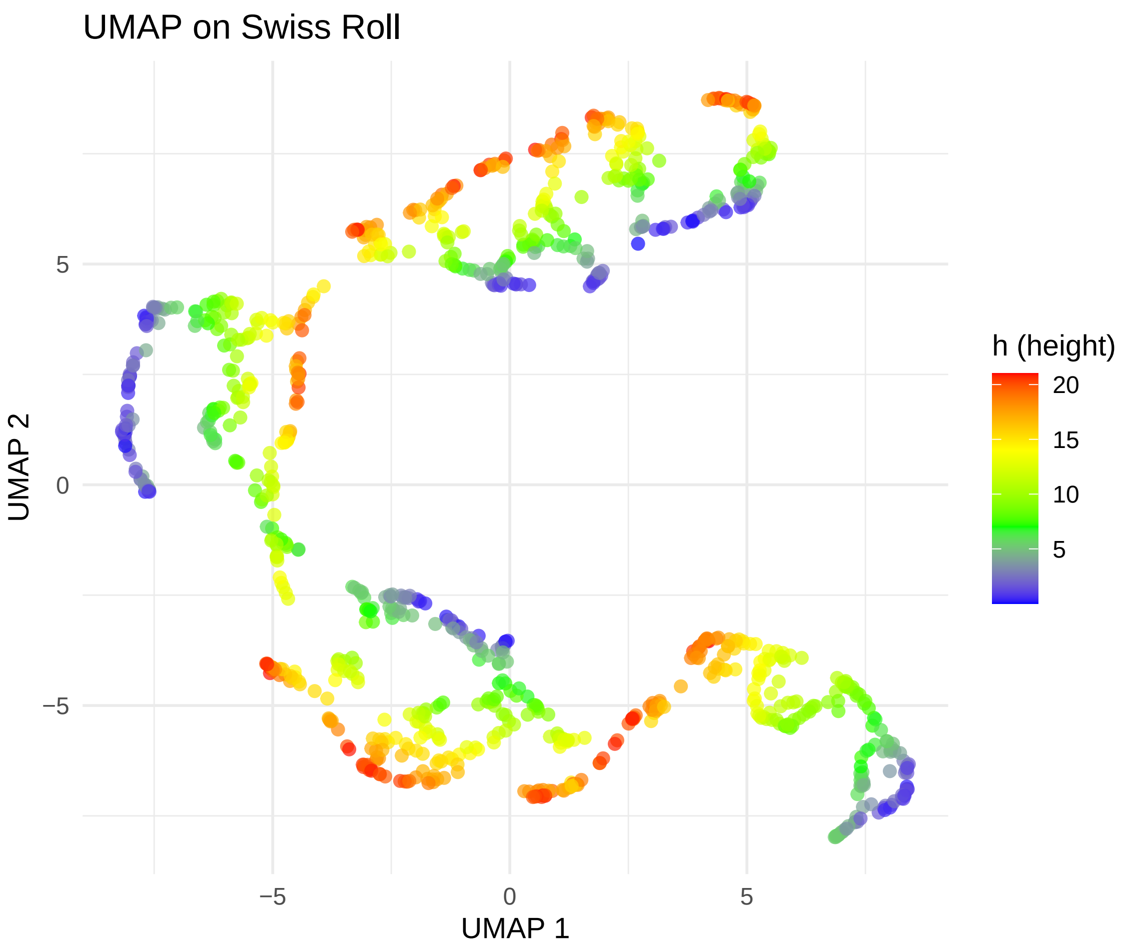
t-SNE

```
library(Rtsne)
library(ggplot2)

# Features for t-SNE (x, y, z)
X <- as.matrix(swiss_roll[, c("x", "y", "z")])

# Use height h as the "label" for color scale
labels <- swiss_roll$h

## 2. t-SNE
# For Swiss roll, small perplexity (20-40) usually works well
tsne_result <- Rtsne(X, dims = 2, perplexity = 30,
                     verbose = FALSE, check_duplicates = F)

tsne_embed <- data.frame(tsne_result$Y)
tsne_embed$label <- labels

## 3. Plot
ggplot(tsne_embed, aes(X1, X2, color = label)) +
  geom_point(size = 1.2, alpha = 0.7) +
  scale_color_gradientn(colors = c("blue", "green",
                                   "yellow", "red")) +
  labs(title = "t-SNE on Swiss Roll",
    x = "t-SNE 1", y = "t-SNE 2", color = "h (height)") +
  theme_minimal()
```



UMAP

```
library(umap)
umap_result <- umap(X)      # default settings work well for Swiss roll

# Output is a matrix, convert to data frame
umap_df <- as.data.frame(umap_result$layout)
colnames(umap_df) <- c("UMAP1", "UMAP2")
umap_df$label <- labels

## 3. Plot
ggplot(umap_df, aes(UMAP1, UMAP2, color = label)) +
  geom_point(size = 1.8, alpha = 0.7) +
  scale_color_gradientn(colors = c("blue", "green", "yellow", "red")) +
  labs(title = "UMAP on Swiss Roll",
    x = "UMAP 1", y = "UMAP 2", color = "h (height)") +
  theme_minimal()
```



Summary

We explored several nonlinear dimensionality reduction techniques:

- Isomap preserves geodesic distances
- LLE maintains local linear structure
- t-SNE is excellent for visualizing clusters
- UMAP balances local/global geometry and scales well