MetabolSSMF

Introduction

This vignette presents the **MetabolSSMF** package, which implements a framework for the simplex-structured matrix factorisation (SSMF) algorithm in R. This package provides a standard algorithm, which allows users to apply the SSMF to perform soft clustering of the data set that they are interested in. The usage of this package will be illustrated through an application to a simulated metabolomic data set. This vignette also includes the usage of:

- **gap statistic** to select the number of clusters;
- **bootstrap resampling** for estimating confidence intervals for the cluster prototypes;
- **soft adjusted Rand index** for comparing known and estimated partitions;
- **Shannon diversity index** for evaluating the average uncertainty in a soft clustering partition.

This document gives a quick tour of **MetabolSSMF** functionalities. It was written in R Markdown, using the knitr package for production. See help(package="MetabolSSMF") for further details.

Installation: The latest version of the package can be installed from **Github**:

```
# Install the package
devtools::install_github("WenxuanLiu1996/MetabolSSMF")

# Load the package
library(MetabolSSMF)
```

Data

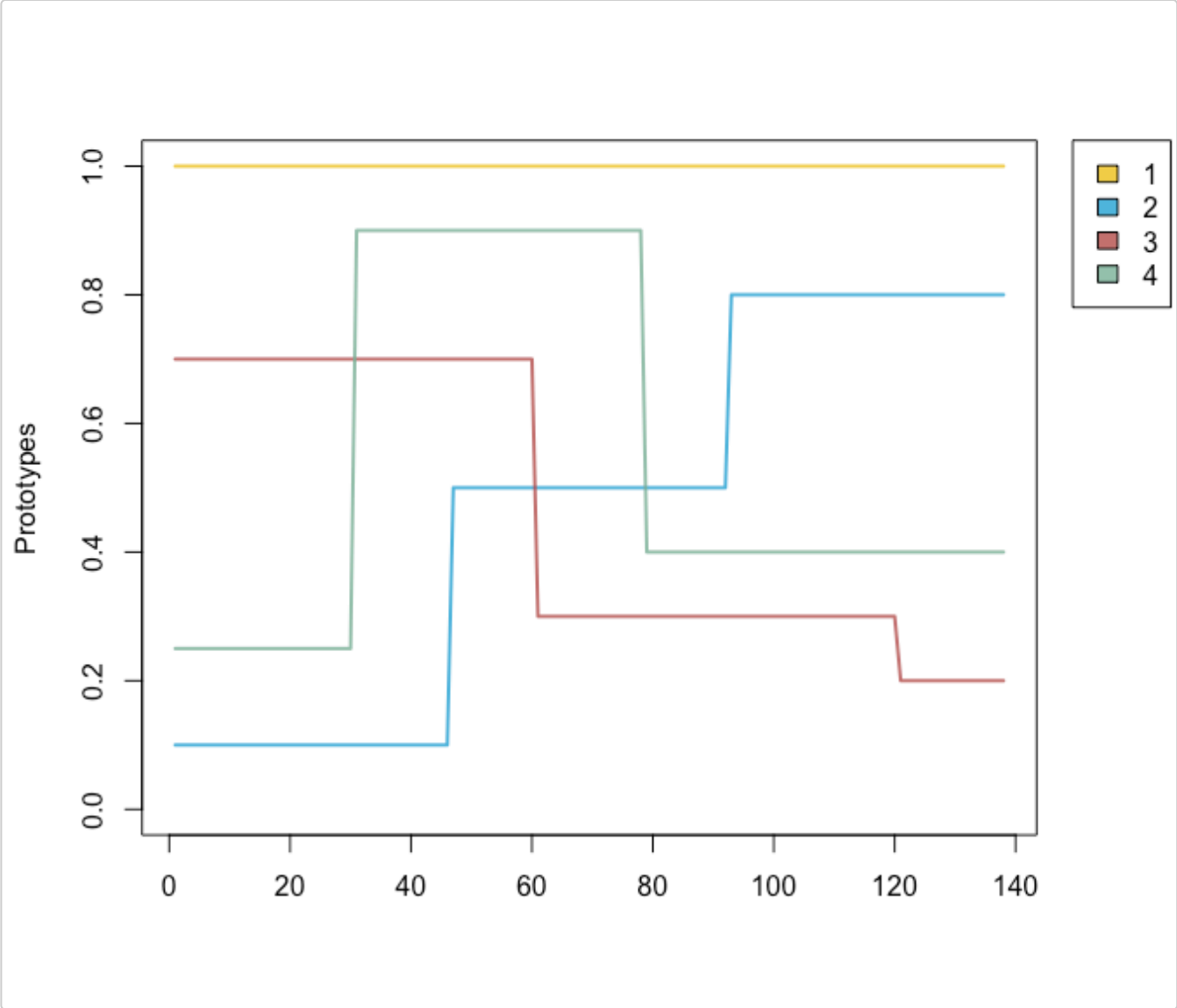
The simulated data set is created by the product of a soft membership matrix H and a prototype matrix W. A user can use a Dirichlet distribution to generate the values for the soft membership matrix H with simplex structure. W can be defined using a designed pattern. The package provides a simulated data set X, simulated prototype matrix W and simulated membership matrix H. These can be loaded through the code `data(SimulatedDataset)`, `data(SimulatedPrototypes)` and `data(SimulatedMemberships)`, respectively.

```
# Simulated data set X
X <- as.matrix(SimulatedDataset)

# Simulated W matrix (prototype matrix)
W <-  as.matrix(SimulatedPrototypes)

# Simulated H matrix (membership matrix)
H <- as.matrix(SimulatedMemberships)
```

Visualise the simulated data

Visualise the values of simulated prototype matrix W

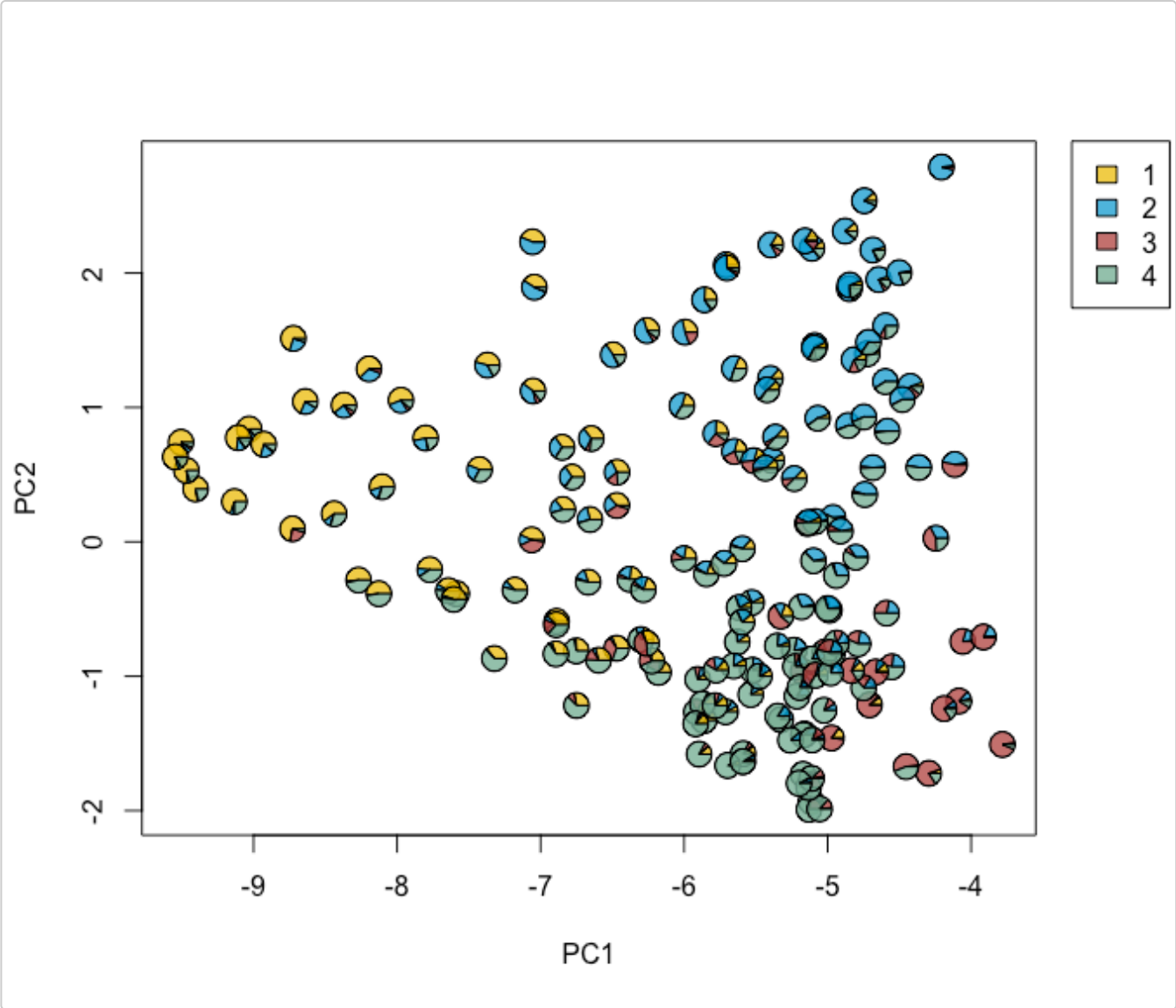
```r
# Define 4 different colors
color <- c(ggsci::pal_jco("default", alpha=0.8)(7)[2], ggsci::pal_jama("default",
        alpha=0.8)(7)[c(3:5)])

par(mar=c(5.1, 4.1, 4.1, 4.1), xpd=TRUE)
matplot(t(W), type='l', lty = 1, ylab='Prototypes', col = color, lwd=2, ylim = c(0,1))
legend('topright', inset=c(-0.15,0), legend = 1:4, fill=color[1:4])
```



Visualise the values of simulated soft membership matrix H in each observation

The scatter pies plot of the 2-dimensional projected clusters. It represents the simulated data with 4 soft clusters, the area of colour in the pies depicts the values of the soft membership of the observations.

```r
pca_X <- prcomp(X, center = F, scale. = F)
par(mar=c(5.1, 4.1, 4.1, 4.1), xpd=TRUE)
caroline::pies(lapply(apply(H, 1, list), unlist), color.table = caroline::nv(color[1:4],
        paste0('Pty', 1:4)), x0=pca_X$x[,1], y0=pca_X$x[,2], radii = 2, ylab='PC2',
        xlab='PC1')
legend('topright', inset=c(-0.15,0), legend = 1:4, fill=color[1:4])
```



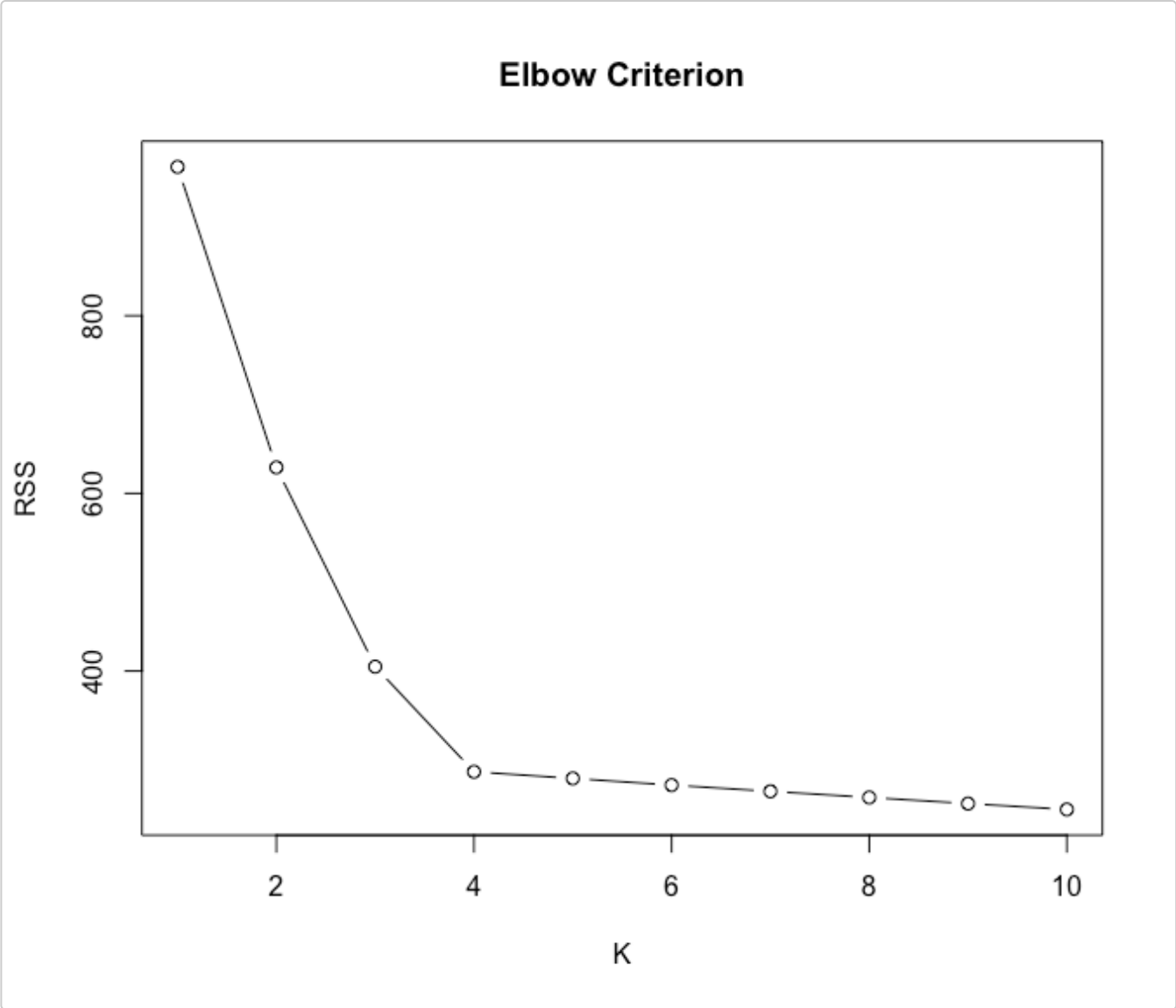
Choose the number of clusters

The SSMF algorithm can be run with $k = 1, 2, \ldots, K$ using a `for` loop and the results can be saved to a list. The 'elbow criterion' or the gap statistic can be done as a subsequent step.

```
# Set the maximum k that is used in loop.
K <- 10

# Run the SSMF algorithm with various k and save the results as a list
fit_SSMF <- list()
for(k in 1:K){
  fit <- ssmf(X, k = k, lr = 0.001, meth='kmeans')
  fit_SSMF[[k]] <- fit
}
```

Plot the residual sum of squares ("elbow criterion")

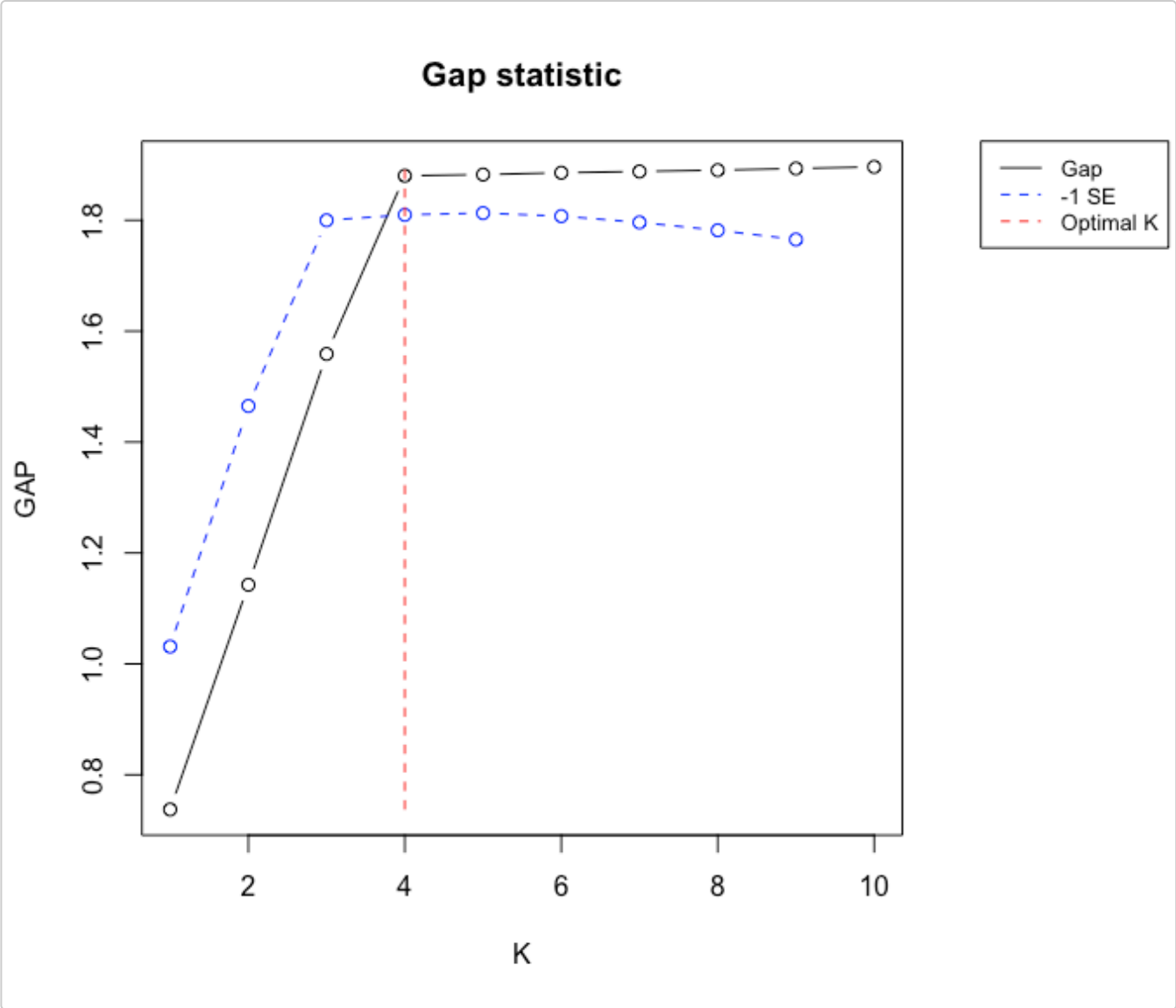
```
# Extract the RSS and save as a vector
rss_SSMF <- unlist(lapply(fit_SSMF, function(x) x$SSE))

# Plot RSS
plot(1:K, rss_SSMF, type="b", xlab = "K", ylab = "RSS", main='Elbow Criterion')
```



If the ‘elbow’ is not clear, the gap statistic is suggested.

Gap statistic

```
# Apply the gap statistic to the simulated data
fit_gap <- gap(X, rss = rss_SSMF)

# Visualise the results
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)
plot(x = c(1:K), y = fit_gap$gap, type='b', ylim=range((fit_gap$gap), (fit_gap$gap -
        fit_gap$standard.error)[-1]), xlab='K', ylab='GAP', main='Gap statistic')
lines(x = c(1:(K-1)), y = (fit_gap$gap - fit_gap$standard.error)[-1], col='blue', lty=2,
        type='b')
lines(x = rep(fit_gap$optimal.k, 2), y = range(fit_gap$gap), lty = 2, col='red')
legend('topright', inset=c(-0.35,0), col=c('black', 'blue', 'red'), lty=c(1, 2, 2), legend
        = c('Gap', '-1 SE', 'Optimal K'), cex=0.8)
```



The estimated number of prototypes is 4 from both the 'elbow criterion' and gap statistic.

Bootstrap

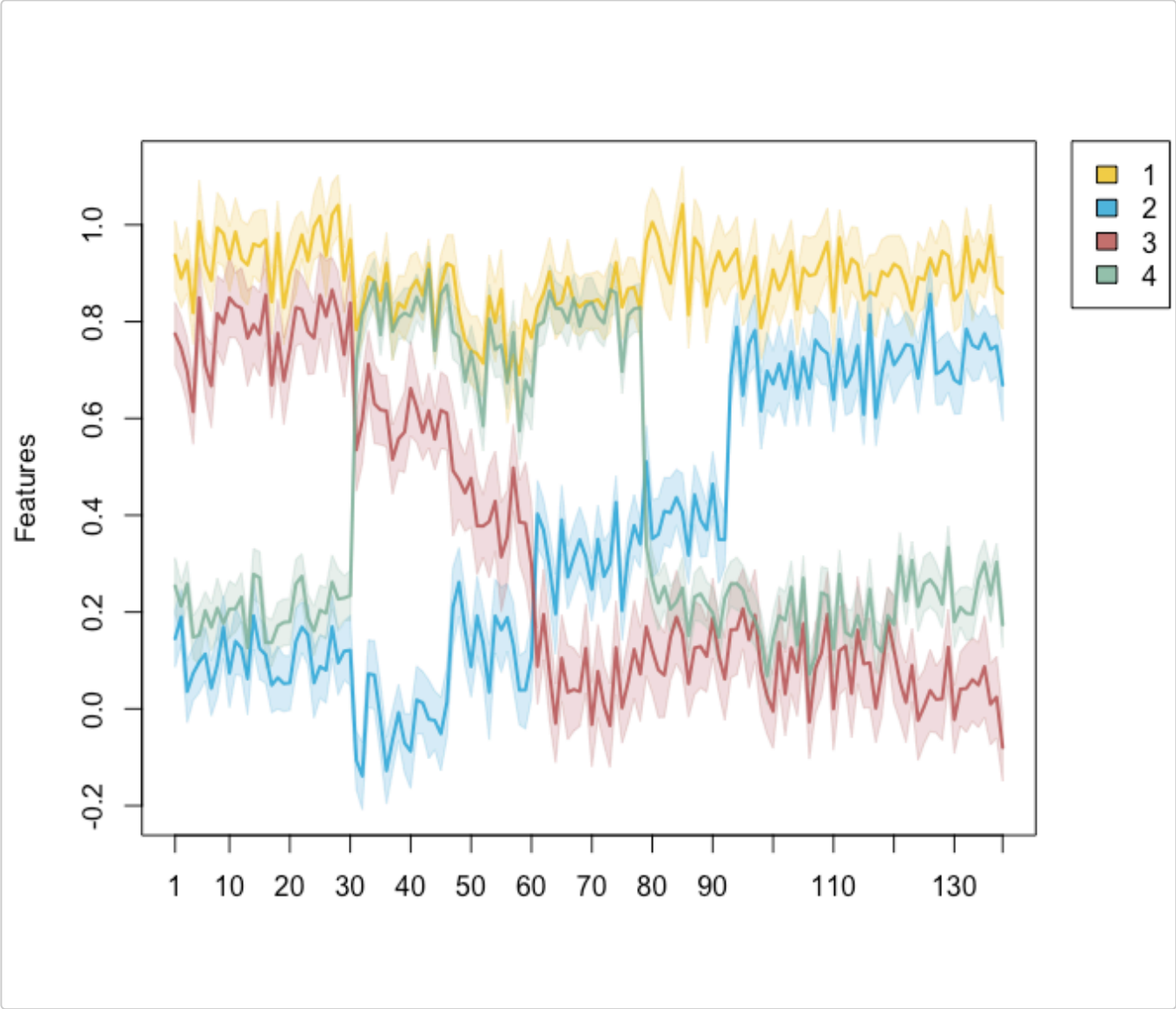
```
# Set the number of times to bootstrap
M <- 1000

# Bootstrap
# Initial H (membership) matrix to start the algorithm
# Based on the results above, k=4 here
initialisation <- init(data = X, k = 4, method = 'kmeans')
fit_boot <- bootstrap(data = X, k = 4, H = initialisation$H, mtimes=M)
```

Note: Usually the H matrix is the output of the function `ssmf( )`. If H is not supplied, the bootstrapped W matrix might have different prototype orders from the outputs of the function `ssmf( )`.

Visualise the values of the bootstrap estimation and confidence intervals for the prototypes

```
par(mar=c(5.1, 4.1, 4.1, 4.1), xpd=TRUE)
matplot(t(fit_boot$W.est[c(2,4,1,3),]), type='n', xaxt='n', ylab='Features', ylim =
        range(fit_boot$lower,fit_boot$upper))
for(i in 1:4){
  alpha <- 0.2
```

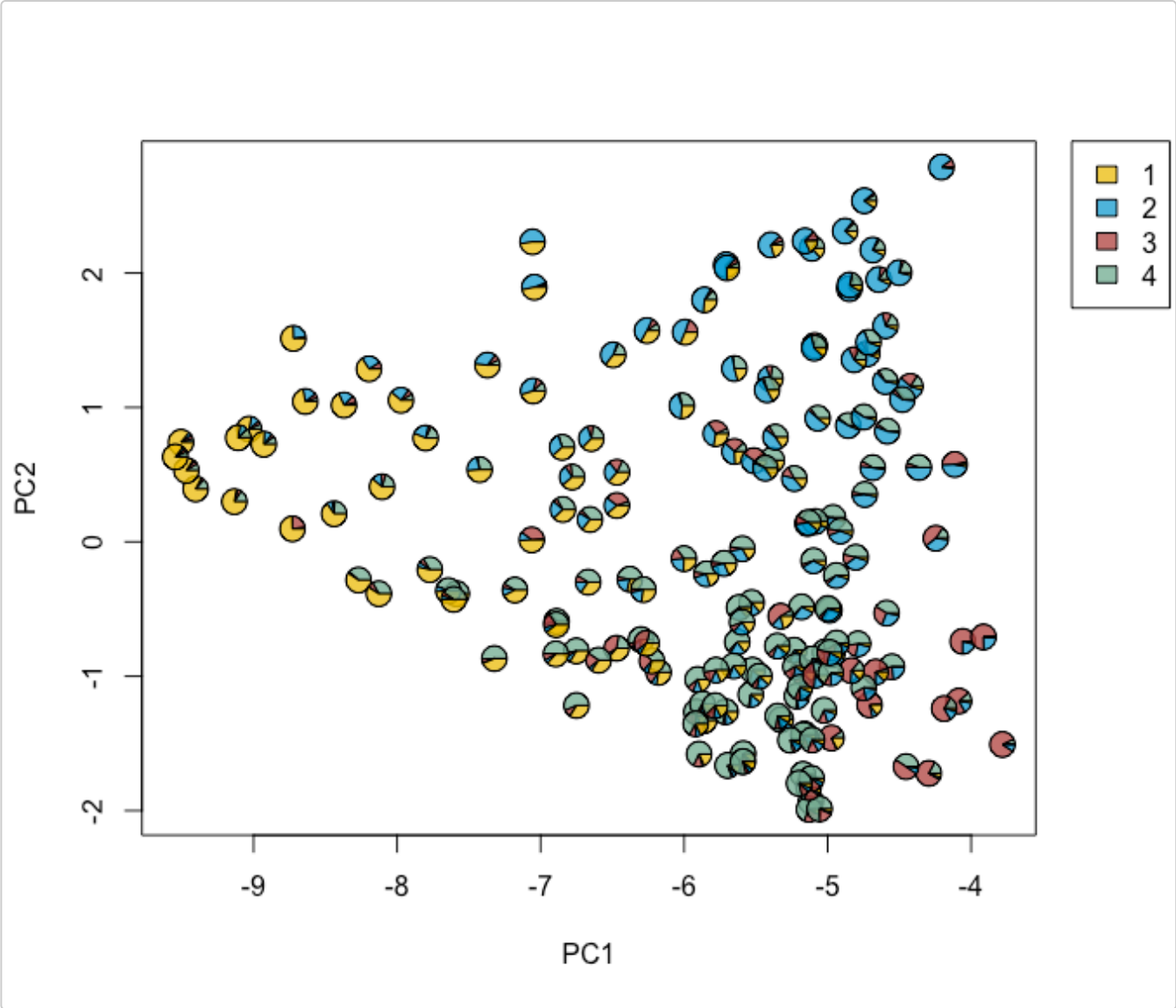
```
  plat <- c(ggsci::pal_jco("default",alpha=alpha)(7)[2], ggsci::pal_jama("default",
        alpha=alpha)(7)[c(3:5)])
  col <- switch(i, plat[1], plat[2], plat[3], plat[4])
  polygon(c(1:138, 138:1), c(fit_boot$upper[c(2,4,1,3),][i,],
        rev(fit_boot$lower[c(2,4,1,3),][i,])), col = col, border= col, lty = 1)
}
matplot(t(fit_boot$W.est[c(2,4,1,3),]), type='l', lty = 1, ylab='Features', add=T,
        col=color, lwd=2)
legend('topright', inset=c(-0.15,0), legend = 1:4, fill=color[1:4])
axis(1, at=c(1, seq(10, 130, 10), 138), labels = c(1, seq(10, 130, 10), 138), cex=0.8)
```



Estimated prototypes are represented by the solid lines and the 4 colours represent 4 clusters. Shadows around the lines are confidence intervals for the prototypes.

Visualise the values of the estimated soft membership matrix

```
par(mar=c(5.1, 4.1, 4.1, 4.1), xpd=TRUE)
caroline::pies(lapply(apply(fit_SSMF[[4]]$H, 1, list), unlist), color.table =
        caroline::nv(color[1:4], c(4,3,2,1)), x0=pca_X$x[,1], y0=pca_X$x[,2], radii = 2,
        ylab='PC2', xlab='PC1')
legend('topright', inset=c(-0.15,0), legend = 1:4, fill=color[1:4])
```



Soft Adjusted Rand Index (sARI)

```
# Compare the true and estimated soft membership matrix
sARI(fit_SSMF[[4]]$H, H)
#> [1] 0.3439103

# Self comparison of the true soft membership matrix
sARI(H, H)
#> [1] 0.3816684
```

The sARI between estimated soft membership matrix and the true soft membership matrix is 0.344. The self comparison of the true soft membership matrix shows the upper bound of this sARI is 0.382. We can use the percentage of these two values 0.344/0.382 = 0.901 to states that the estimated soft membership matrix explained 90.1% information of the true soft membership matrix.

Shannon diversity index

```
E <- rep(NA, nrow(fit_SSMF[[4]]$H))
for(i in 1:nrow(fit_SSMF[[4]]$H)){
  E[i] <- diversity(fit_SSMF[[4]]$H[i,], two.power=T)
}
```

```r
round(mean(E), 1)
#> [1] 2.7
```

The estimated soft memberships of the simulated observations have an average values of $2^{E(h_{i\cdot})} = 2.7$, indicating that most observations softly clustered by SSMF are fuzzy between 2 and 3clusters.

Additional NOTE

The SSMF algorithm has randomness, this document presents one of the situation using `set.seed(12345)` at the beginning. The results will vary with different seeds that are used in R. It is suggested to not set the seed and run the algorithm multiple times using the same values of k (the number of prototypes). Then, select a result with the lowest RSS, save it to the list and continue processing next steps.