Demonstation of cheng package

edited on 2015.12.14 Cheng

Table of contents

Package installation

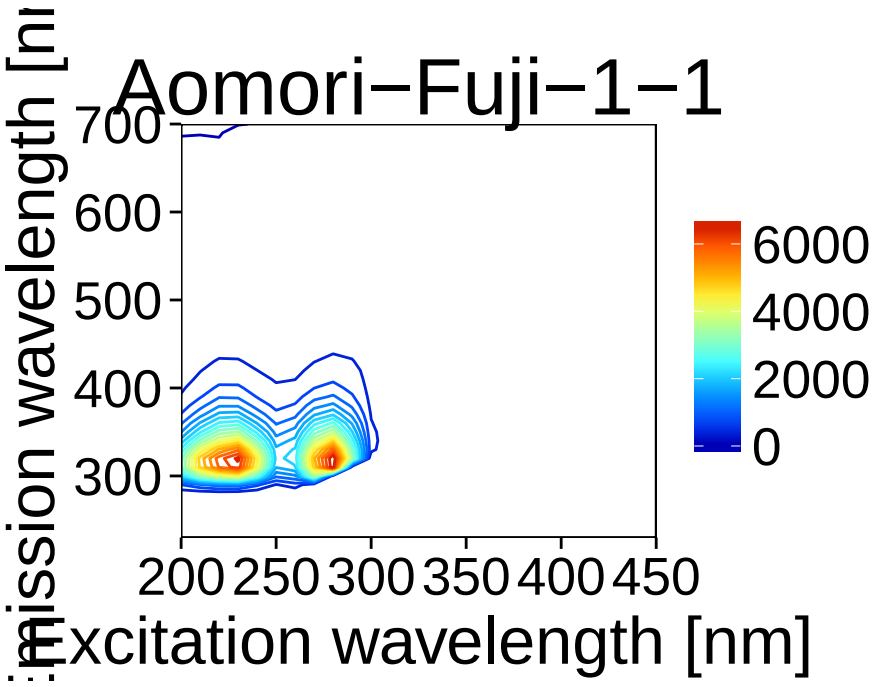
The package is put on private repo on bitbucket. It can be installed using `install_bitbucket` funciton in devtools package.

```
library(devtools)
install_bitbucket("chengvt/cheng", auth_user="keisoku",
            password="kikuchi", dependencies = TRUE)
library(cheng)
```

drawEEMgg: create normal contours for EEM and EEMweight objects

`drawEEMgg` uses ggplot2 to create normal contours. Plan to move this to EEM package in the future.

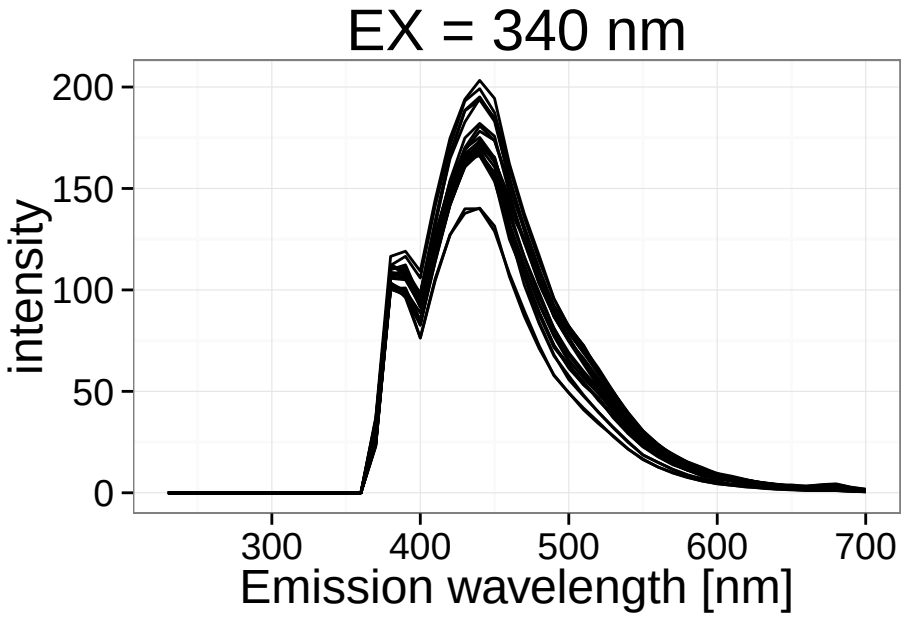
```
data(applejuice)
drawEEMgg(applejuice, 1)
```



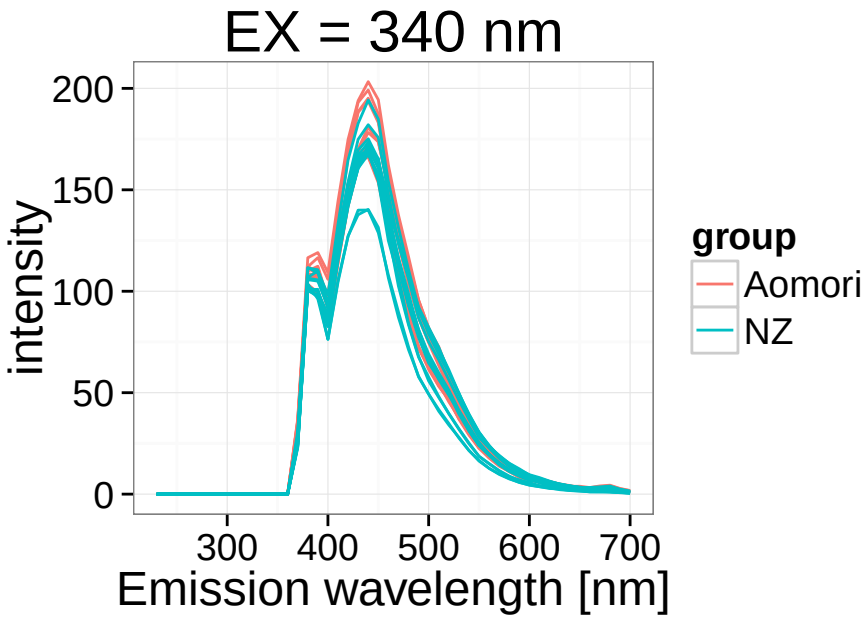
drawSpec: drawSpectra

Select and draw excitation or emission spectra. This function should also go to EEM package in the future.

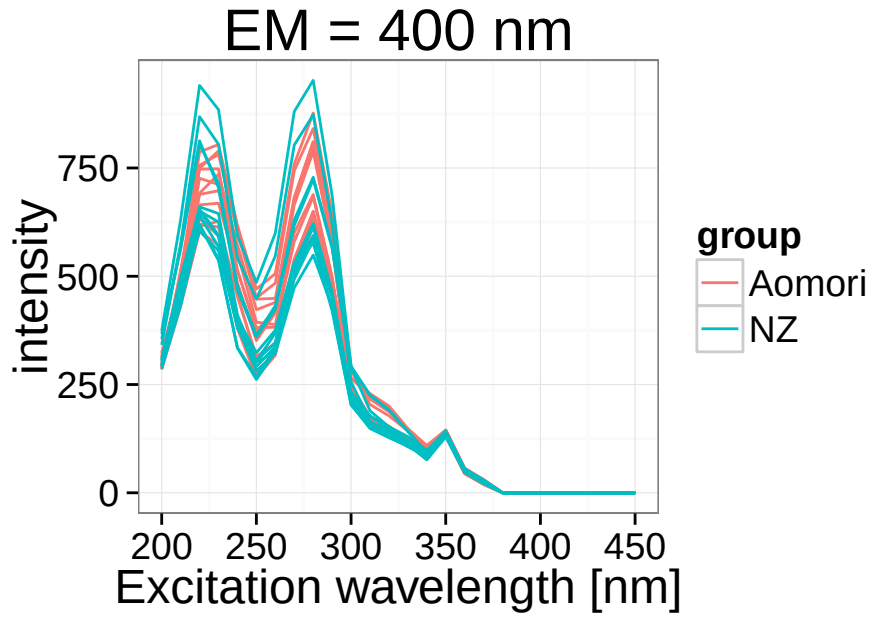
```
require(EEM)
data(applejuice)
country <- sapply(strsplit(names(applejuice), split = "-"), "[", 1)

# ggplot
drawSpec(unfold(applejuice), EX = 340)
```



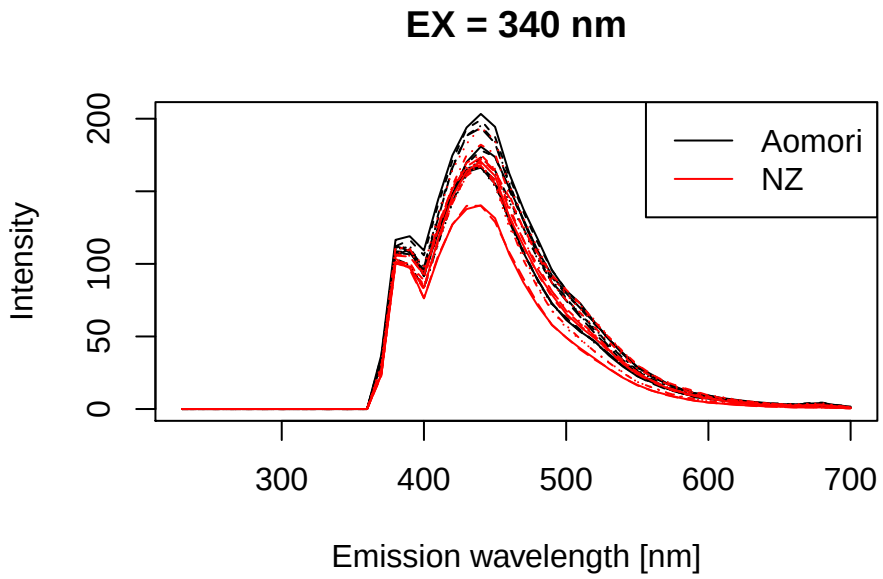

```
drawSpec(unfold(applejuice), EX = 340, group = country)
```



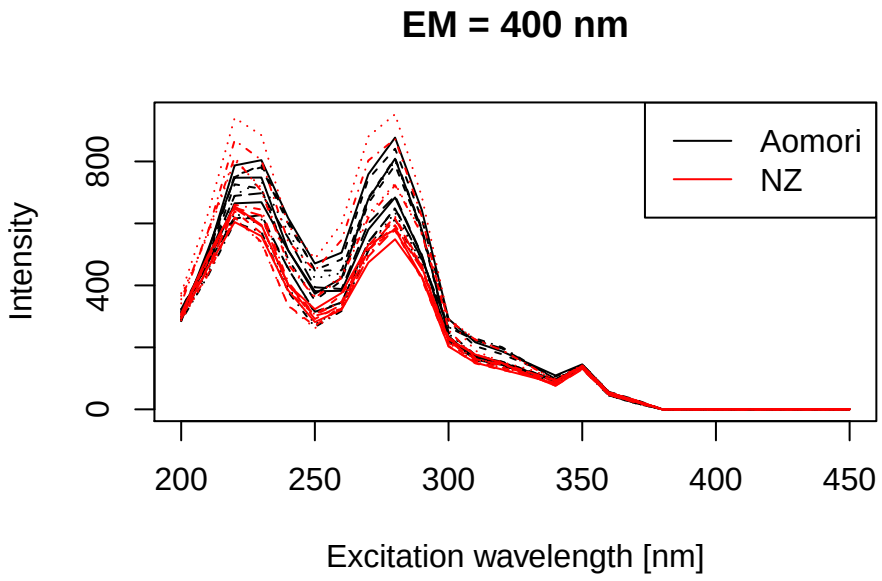

```
drawSpec(unfold(applejuice), EM = 400, group = country)
```




```
# base plot
drawSpec(unfold(applejuice), EX = 340, group = country, ggplot = FALSE)
```




```
drawSpec(unfold(applejuice), EM = 400, group = country, ggplot = FALSE)
```



delZeroCol

Useful for deleting columns which are filled with zero for autoscaling operation.

```
data(applejuice)
applejuice_uf <- unfold(applejuice)
dim(applejuice_uf)
```

```
## [1]   24 1248
```

```
applejuice_uf_nozero <- delZeroCol(applejuice_uf)
dim(applejuice_uf_nozero)
```

```
## [1]  24 923
```

no_zero_col

Function for detecting non-zero columns. Used by `delZeroCol` function.

```
data(applejuice)
applejuice_uf <- unfold(applejuice)
no_zero_col(applejuice_uf)[1:5] # return index
```

```
## EX200EM230 EX200EM240 EX200EM250 EX200EM260 EX200EM270
##          1          2          3          4          5
```

getR2: getR2 value from mvr class object

Alternative to pls package's R2. While R2 requires a newdata dataframe which combines both predictors and target, getR2 lets you put in newx and newy separately. Aside from that, declaring estimate in getR2 remind you which value you got.

```
require(pls)
data(yarn)
model <- plsr(density ~ NIR, 6, data = yarn, validation = "CV")
R2(model)
```

```
## (Intercept)      1 comps      2 comps      3 comps      4 comps
##    -0.07545      0.95679      0.98002      0.99379      0.99927
##     5 comps      6 comps
##     0.99961      0.99975
```

```
getR2(model, estimate = "train") # return R2 at particular ncomp without intercept value
```

```
## R2C = 0.9999 (ncomp = 6)
```

```
## [1] 0.9999099
```

```
getR2(model, estimate = "CV")
```

```
## R2CV = 0.9997 (ncomp = 6)
```

```
## [1] 0.9997464
```

getRMSE: getRMSE value from mvr class object

Alternative to pls package's RMSEP. While RMSEP requires a newdata dataframe which combines both predictors and target, getRMSE lets you put in newx and newy separately. Aside from that, declaring estimate in getRMSE remind you which value you got.

```
require(pls)
data(yarn)
model <- plsr(density ~ NIR, 6, data = yarn, validation = "CV")
RMSEP(model)
```

```
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           27.46    5.294    4.509    2.175   0.8969   0.6516   0.5900
## adjCV        27.46    4.830    4.553    2.155   0.8550   0.6286   0.5717
```

```
getRMSE(model, estimate = "train") # return RMSE at particular ncomp without intercept value
```

```
## RMSEC = 0.2514 (ncomp = 6)
```

```
## [1] 0.2513607
```

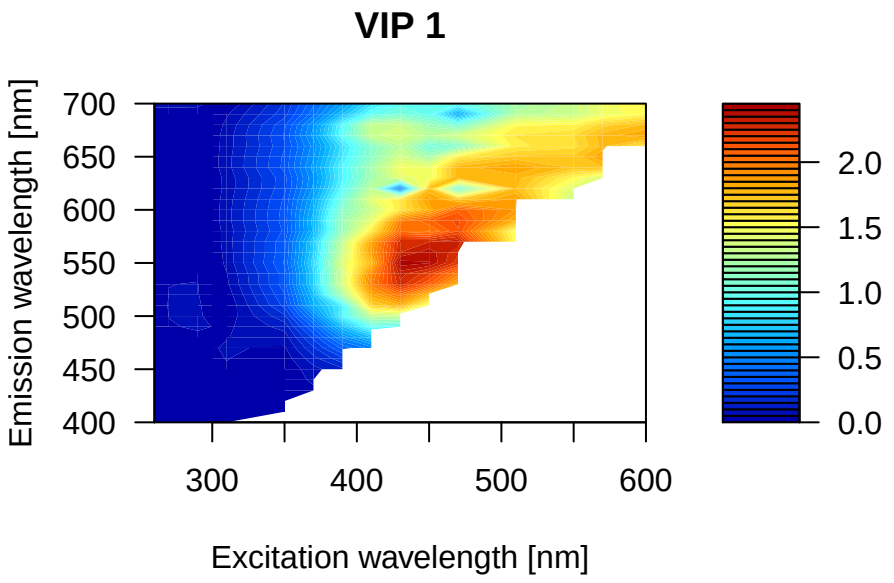
```r
getRMSE(model, estimate = "CV")
```

```
## RMSECV = 0.59 (ncomp = 6)
```

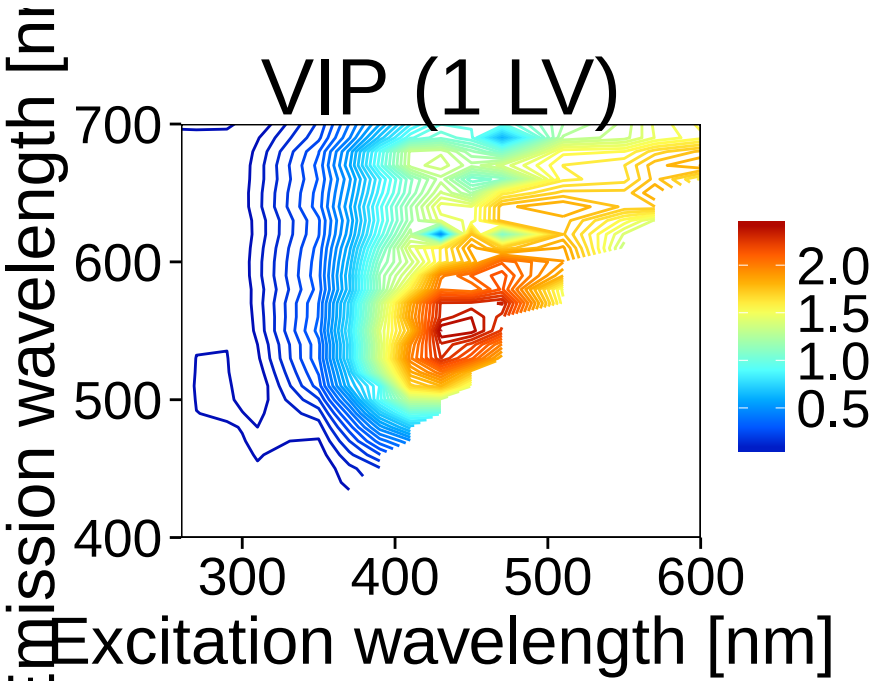
```
## [1] 0.5900449
```

getVIP: getVIP from mvr class object

```r
# load data
require(EEM)
data(gluten)
gluten_uf <- unfold(gluten) # unfold list into matrix
# delete columns with NA values
index <- colSums(is.na(gluten_uf)) == 0
gluten_uf <- gluten_uf[, index]
gluten_ratio <- as.numeric(names(gluten))

# build pls model using pls model
require(pls)
model <- plsr(gluten_ratio ~ gluten_uf, ncomp = 10, method = "oscorespls")
drawEEM(getVIP(model), 1) # color contour
```



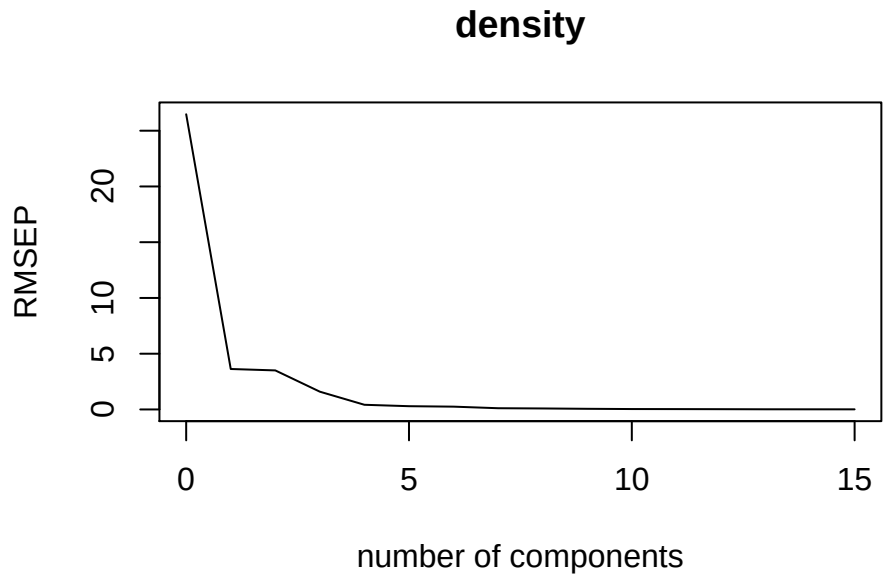

```r
drawEEMgg(getVIP(model), 1, nlevel = 50) # normal contour
```



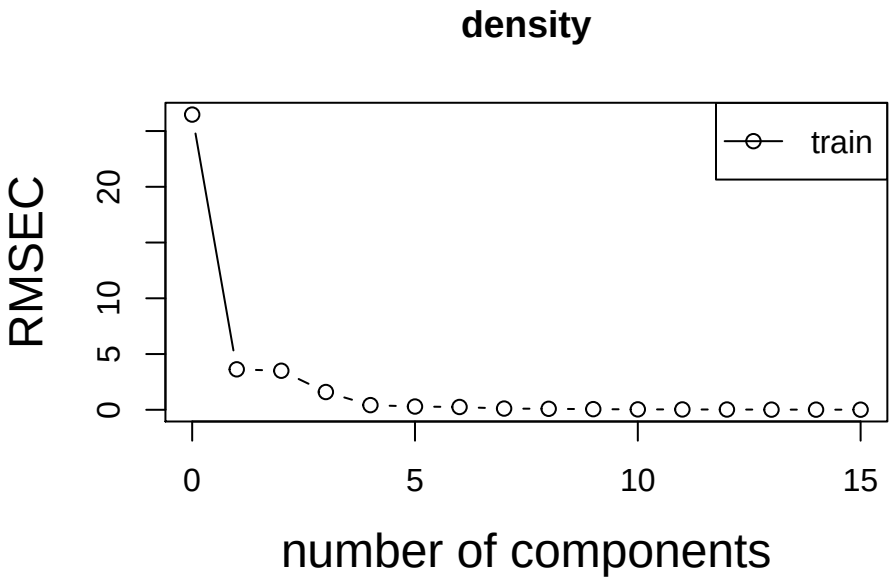
plot_ncomp: Plotting selected ncomp

Plotting ncomp of pls model.

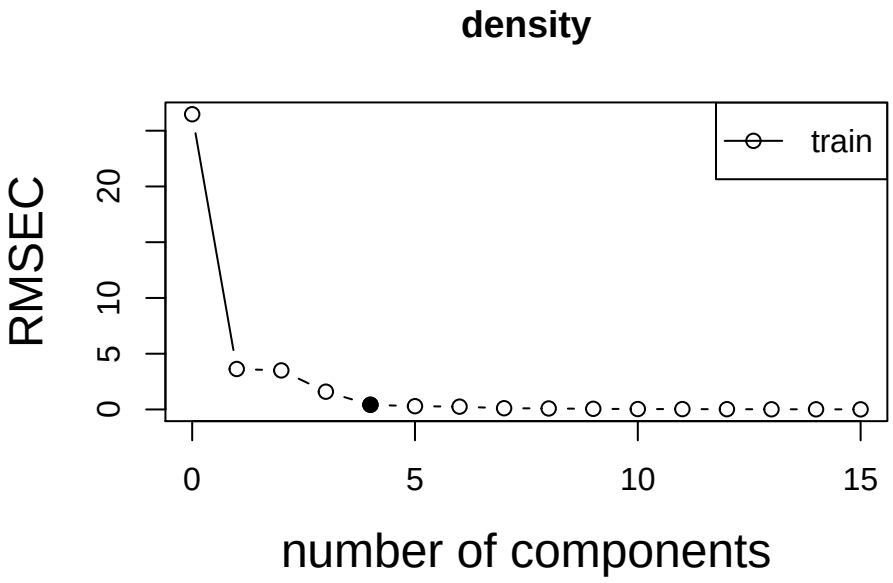
```
require(pls)
data(yarn)
model <- plsr(density ~ NIR, 15, data = yarn)

# plot ncomp
validationplot(model) # available function in pls package
```



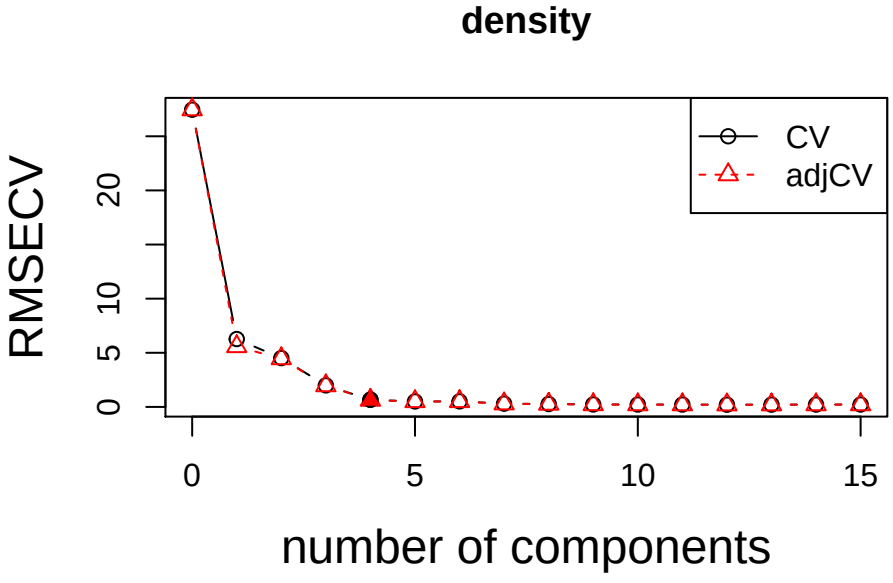

```
plot_ncomp(model) # offer a new format for easier view
```




```
plot.new()
plot_ncomp(model, ncomp = 4) # fill in the point of the selected LV
```

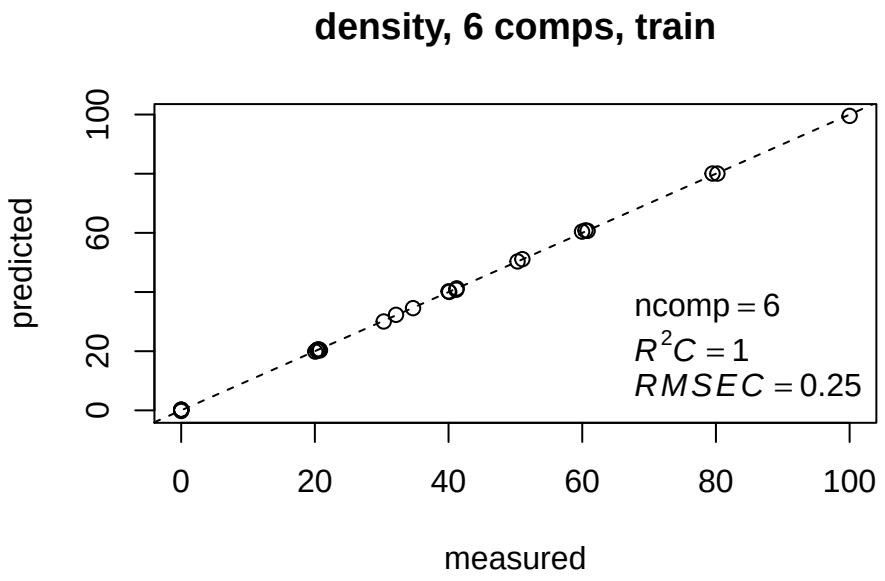



```
# now build pls model again using cross-validation
model <- plsr(density ~ NIR, 15, data = yarn, validation = "CV")
plot_ncomp(model) # notice that y-axis label change to RMSECV
plot_ncomp(model, ncomp = 4) # fill in the point of the selected LV
```

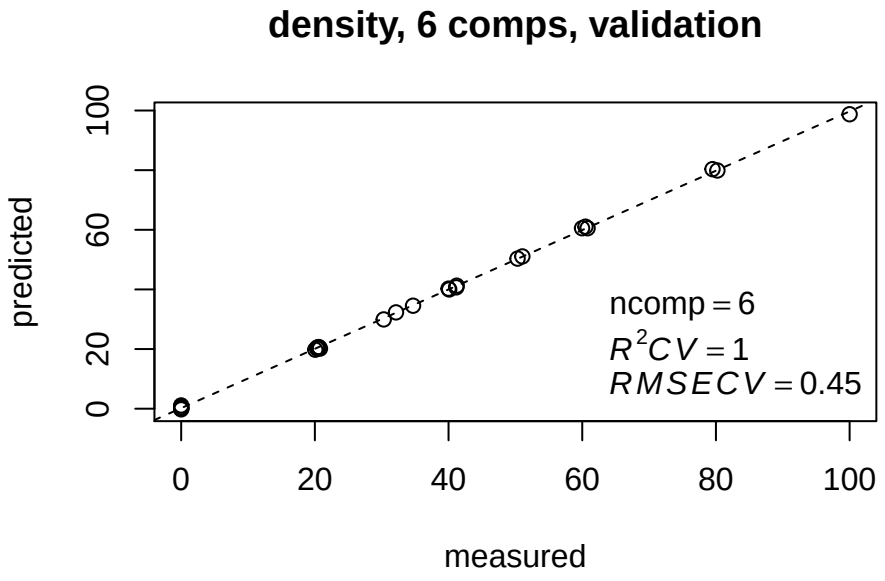


plsplot: Plotting pls prediction result

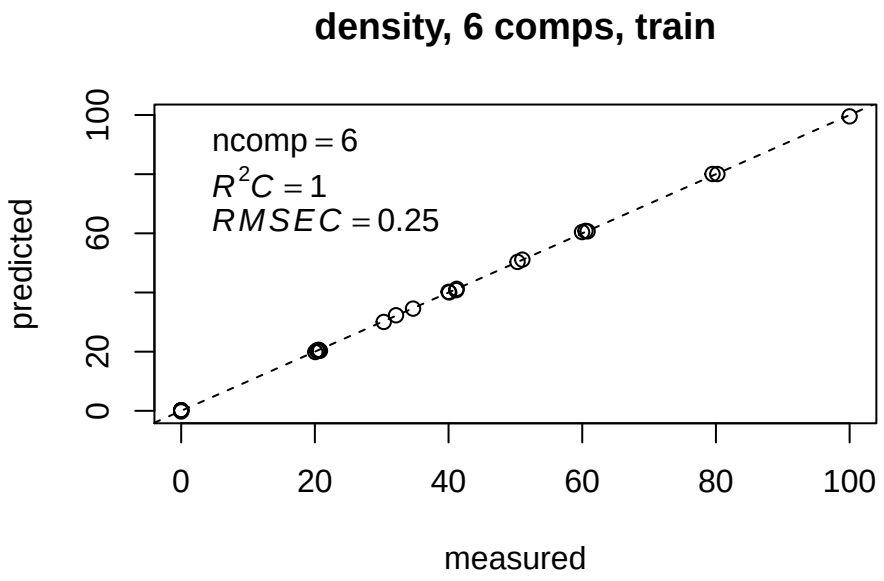
```
require(pls)
data(yarn)
model <- plsr(density ~ NIR, 6, data = yarn, validation = "CV")
plsplot(model) # calibration set result
```



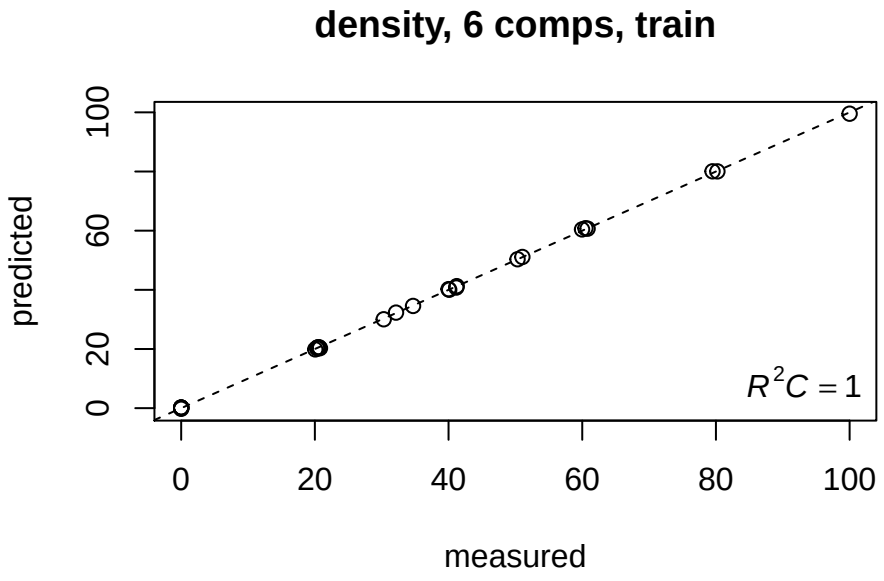

```
plsplot(model, estimate = "CV") # cross validation set result
```



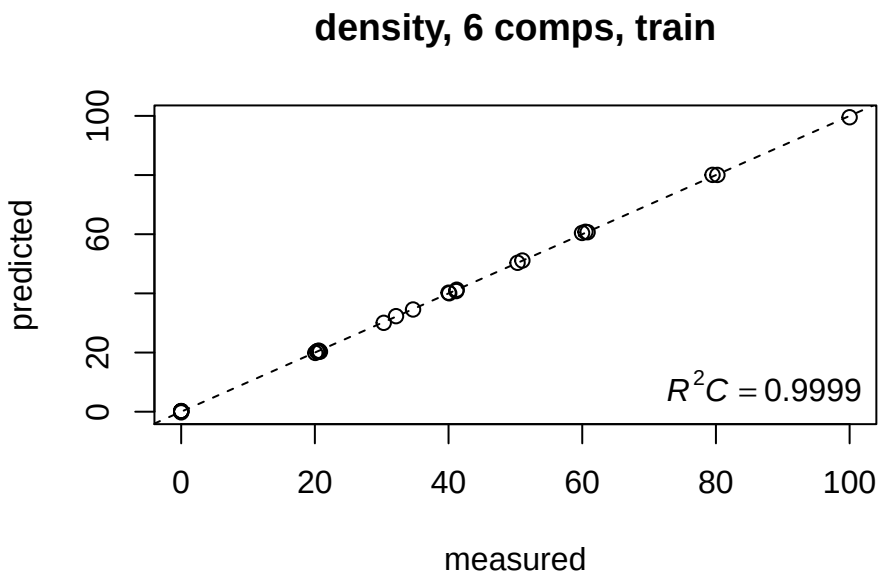
```
## customizing the graphs
plsplot(model, location = "topleft") # change legend position
```



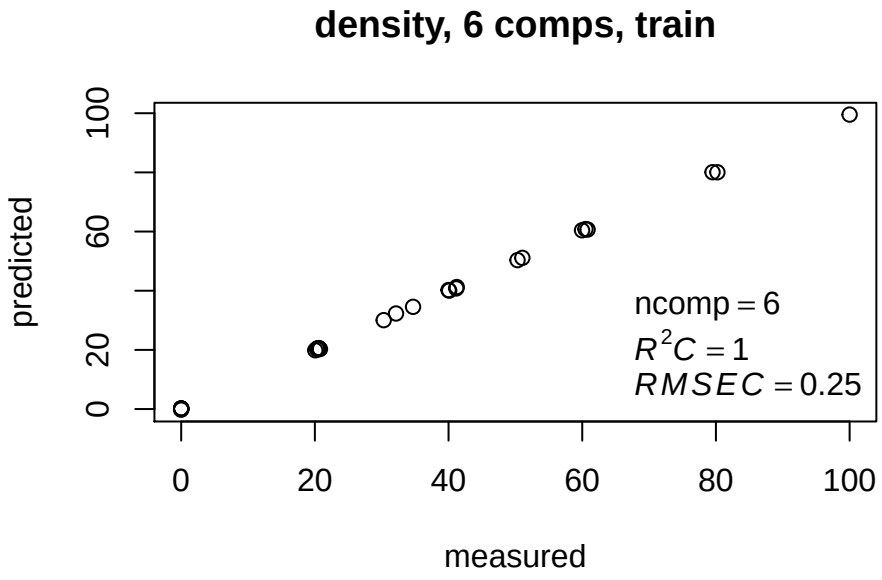

```
plsplot(model, show = "R2") # show only R2
```



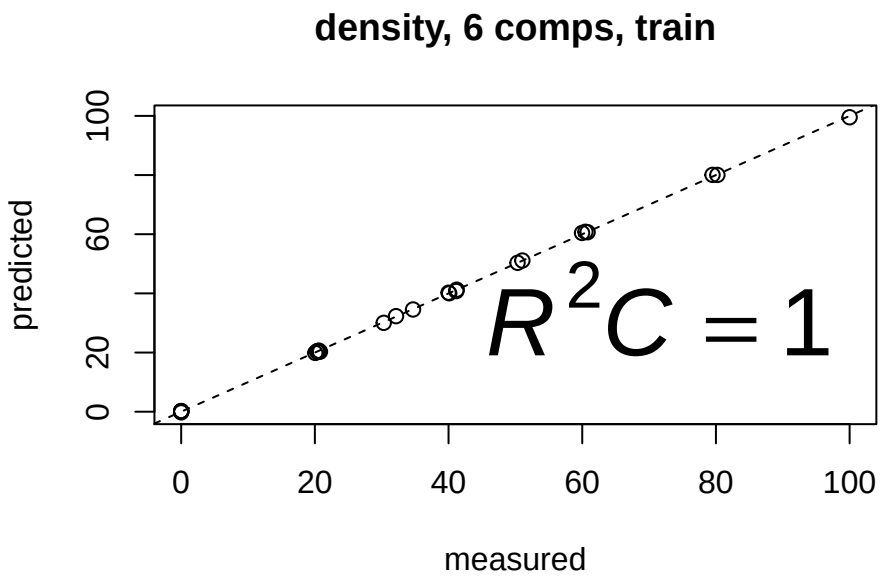

```
plsplot(model, show = "R2", round = 4) # round to four digits
```



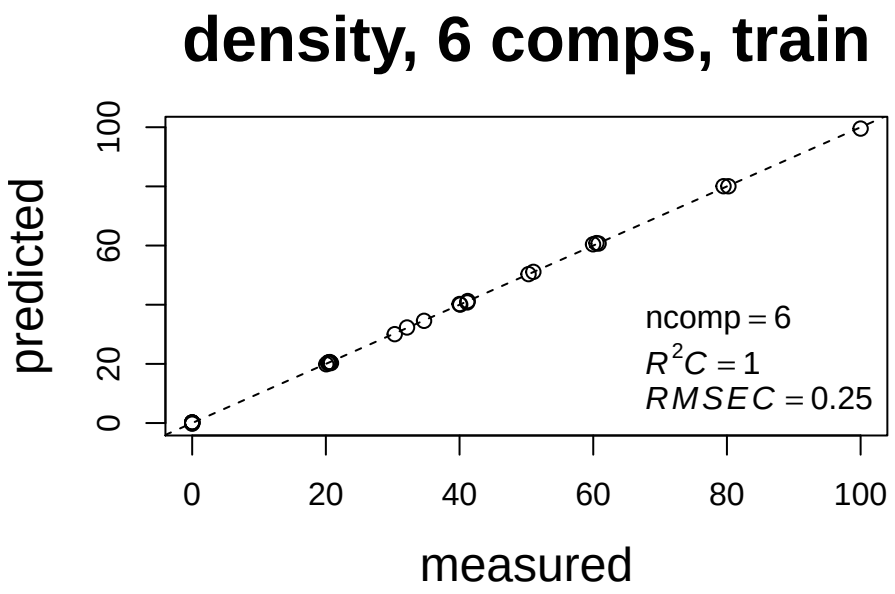

```
plsplot(model, fitline = FALSE) # get rid of fitline
```




```
plsplot(model, show = "R2", cex.stats = 3) # bigger stats font
```




```
plsplot(model, cex.lab = 1.5, cex.main = 2) # bigger labels font
```

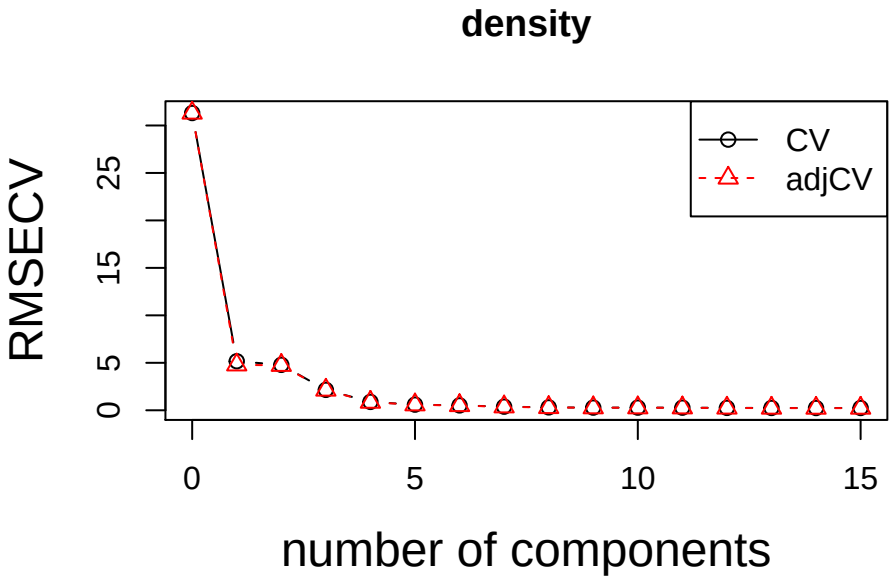


plsplot2: Plotting pls prediction result

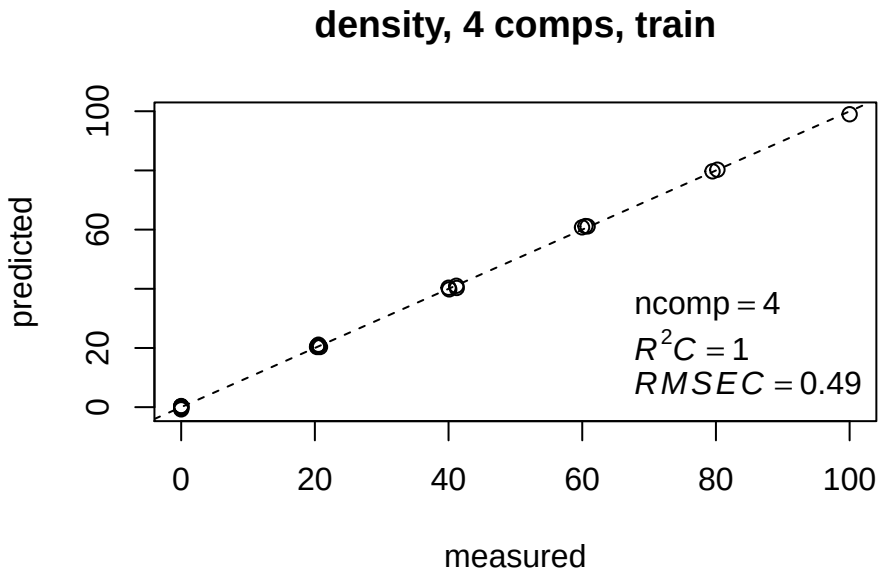
`plsplot2` plots both calibration and validation on the same graph

```
require(pls)
data(yarn)
```

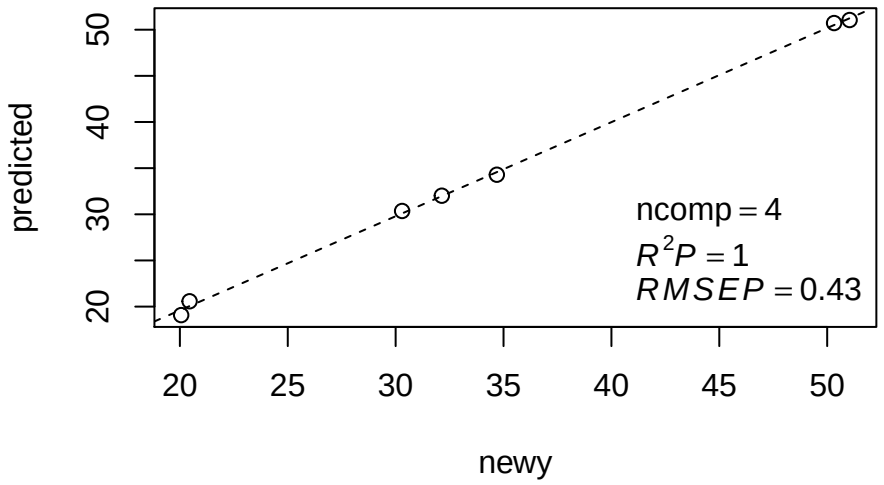
```
yarn.cal <- yarn[yarn$train,]
yarn.val <- yarn[!yarn$train,]
model <- plsr(density ~ NIR, 15, data = yarn.cal, validation = "CV")
plot_ncomp(model)
```



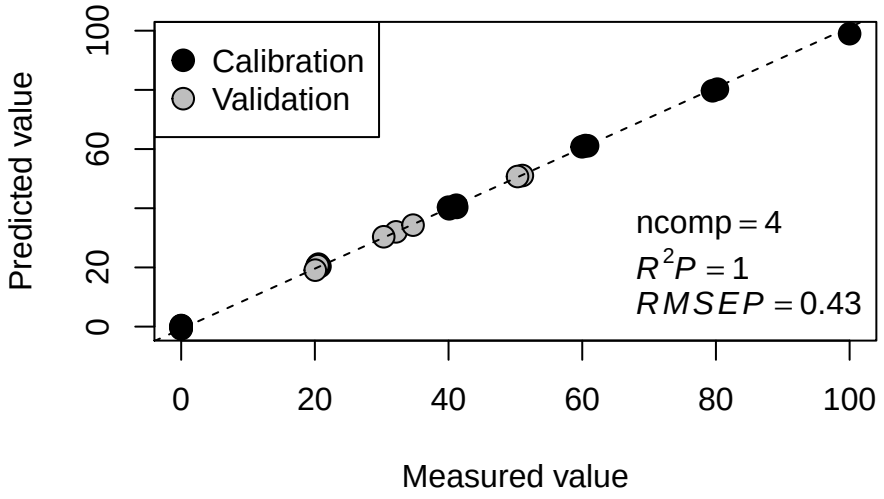

```
ncomp <- 4 # 4 components seem to be appropriate
model <- plsr(density ~ NIR, ncomp, data = yarn.cal) # recalculate

plsplot(model) # calibration
```



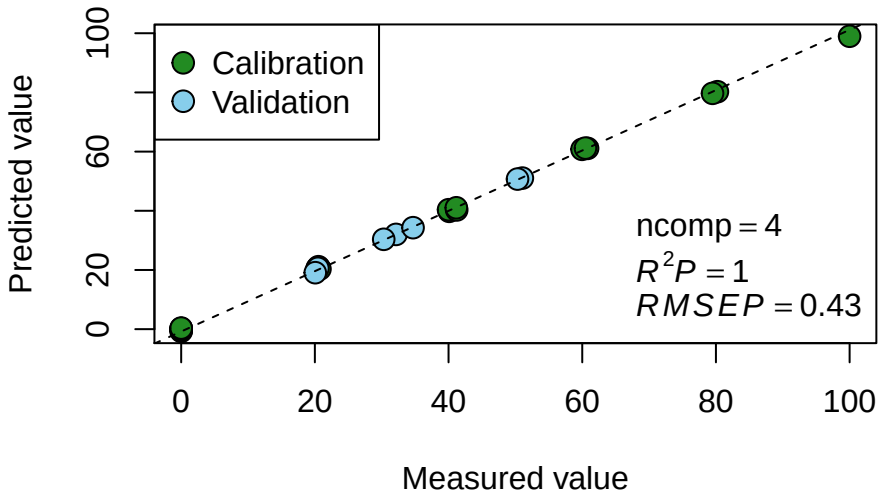
```r
plsplot(model, newx = yarn.val$NIR, newy = yarn.val$density) # validation
```




```r
# now put those two plots together
plsplot2(model, newx = yarn.val$NIR, newy = yarn.val$density) # calibration and validation
```




```r
plsplot2(model, newx = yarn.val$NIR, newy = yarn.val$density, col.cal = "forestgreen", col.val = "skybl
```



trainPLS

`trainPLS` can be used to train PLS for train dataset by cross-validation. The preprocessing method will be optimized automatically. However, the number of latent variables has to be determined manually.

```
# load data
require(EEM)
data(gluten)
gluten_uf <- unfold(gluten) # unfold list into matrix
# delete columns with NA values
index <- colSums(is.na(gluten_uf)) == 0
gluten_uf <- gluten_uf[, index]
gluten_ratio <- as.numeric(names(gluten))

# build pls model using pls model
require(pls)
result <- trainPLS(x = gluten_uf, y = gluten_ratio)
```

The three preprocessing methods were applied automatically, namely, mean-center, normalize + mean-center and autoscale.

trainPLS2

Train PLS for train dataset by cross-validation. This is different from trainPLS as you have to specify the preprocessing method manually. In addition, the variable reduction by VIP can be done automatically with number of cycles specifiable.

```
result <- trainPLS2(x = gluten_uf, y = gluten_ratio, cycles = 2)
```

The number of variables will decrease with each cycle.

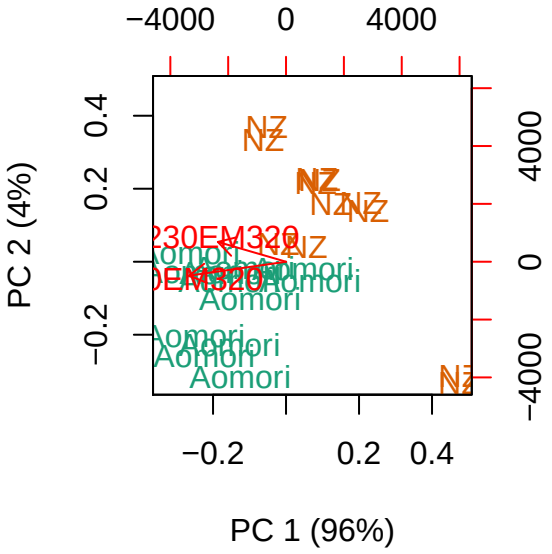
Biplot functions

Some customizations were added to `biplot` function of stats package. The usages were illustrated below.

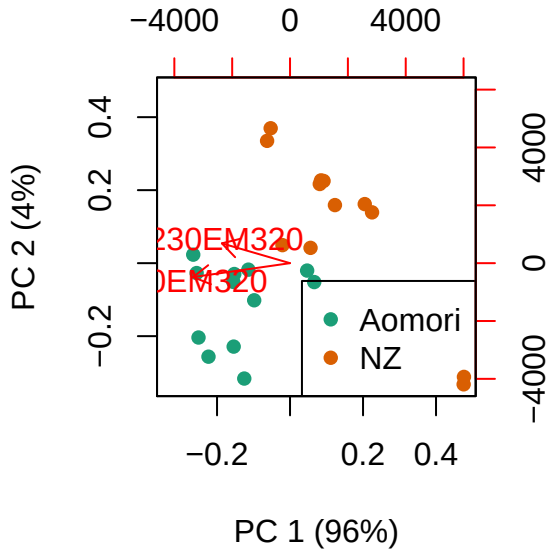
```
require(EEM)
data(applejuice)
applejuice_uf <- unfold(applejuice) # unfold list into matrix
# get country of apple production
country <- sapply(strsplit(names(applejuice), split = "-"), "[", 1)

# select peaks
local_peak <- findLocalMax(applejuice, n = 1)
```

```
##    EX  EM   value
## 1 230 320 6585.27
## 2 280 320 6826.21
```

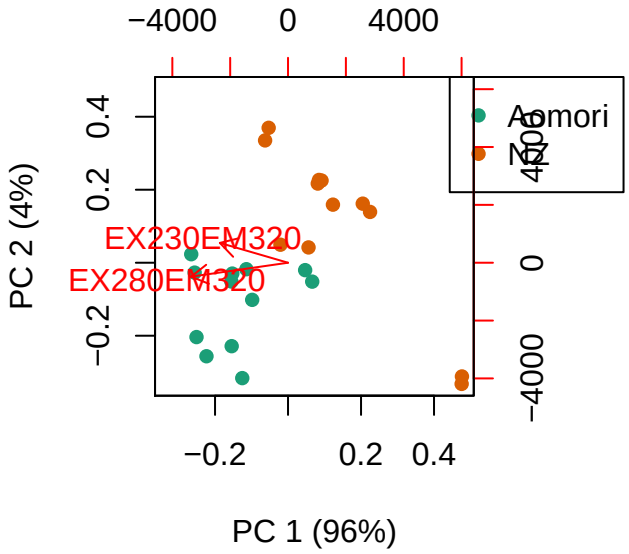
```
index <- colnames(applejuice_uf) %in% local_peak
applejuice_uf_selectedPeak <- applejuice_uf[,index, drop = FALSE]

# PCA
result <- prcomp(applejuice_uf_selectedPeak)

# create color palette for x points
library(RColorBrewer)
xcol <- brewer.pal(3, "Dark2")

# biplot 2: color the scores by group
biplot2(result, xlab = prcompname(result ,1), ylab = prcompname(result,2),
xlabs = country, xcol = xcol)
```



```
# biplot3: turn scores into points and color them by group
biplot3(result, xlab = prcompname(result ,1), ylab = prcompname(result,2),
xlabs = country, xcol = xcol)
```




```
# biplot4: same as biplot3 but move legend outside
biplot4(result, xlab = prcompname(result ,1), ylab = prcompname(result,2),
xlabs = country, xcol = xcol, legendinset = 0.07)
```



getText

```
string <- "country_cultivar_fruit_1"
getText(string, 2) # get the second group of string
```

```
## [1] "cultivar"
```

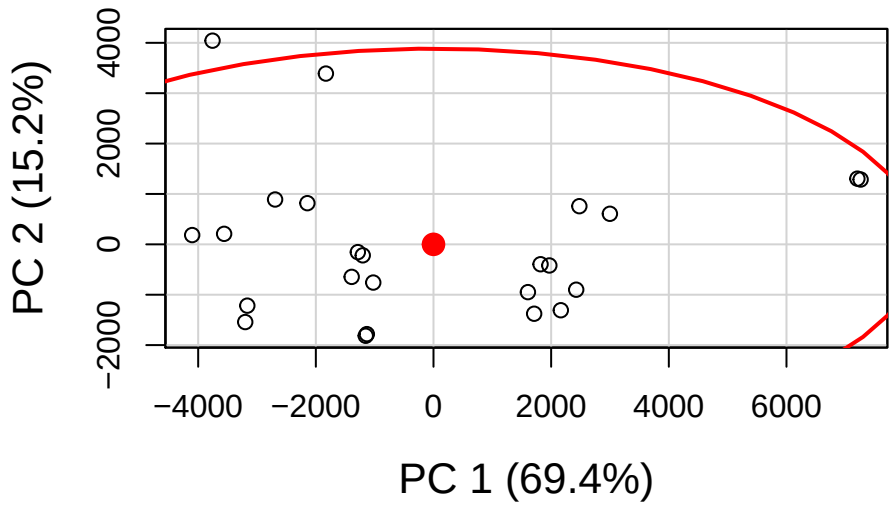
```
# different separator
string <- "country~cultivar~fruit~1"
getText(string, 2, sep = "~")
```

```
## [1] "cultivar"
```

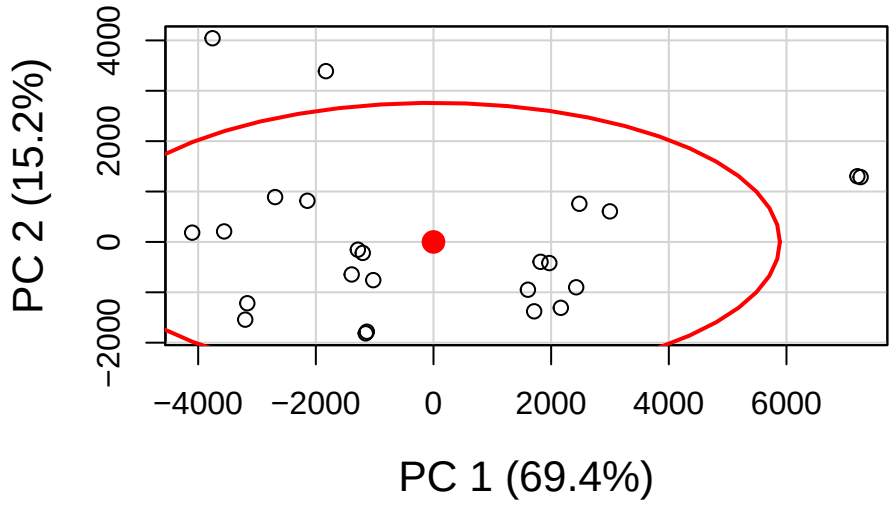
plotScore_ellipse

Plotting ellipse for PCA score plots

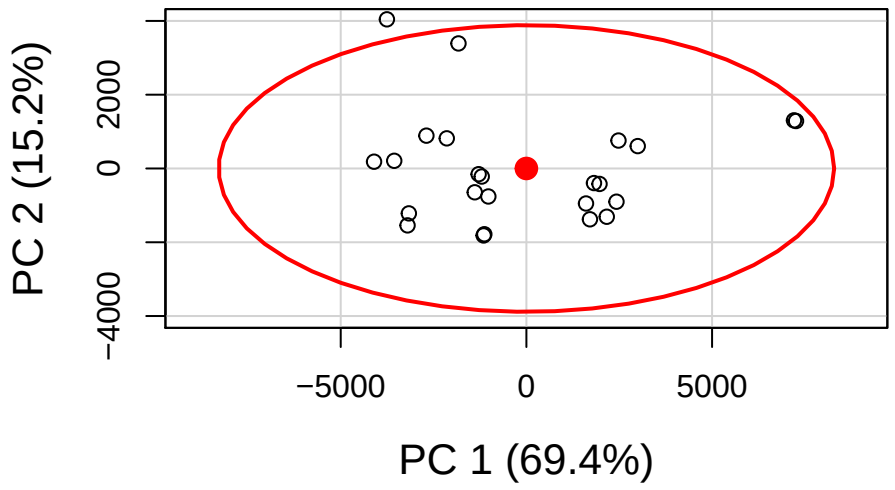
```
data(applejuice)
applejuice_uf <- unfold(applejuice)
PCA <- prcomp(applejuice_uf)
plotScore_ellipse(PCA)
```



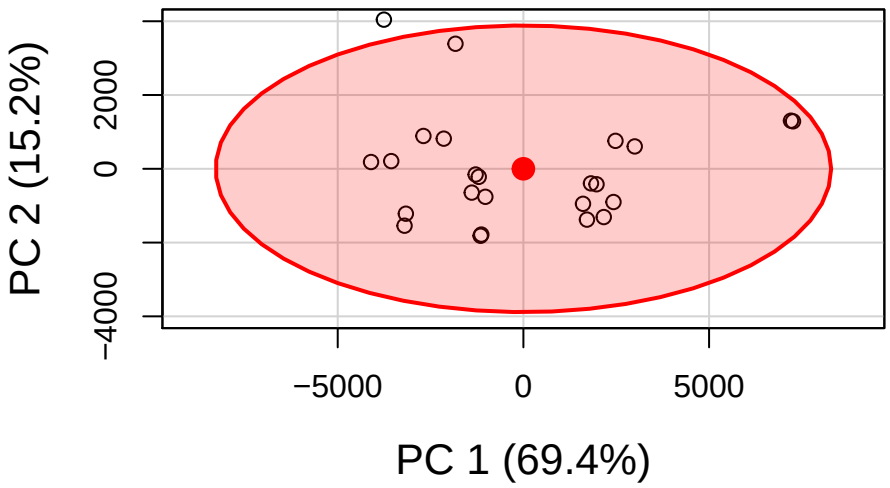

```
# change level
plotScore_ellipse(PCA, level = 0.8)
```




```
# manually set x,y ranges
plotScore_ellipse(PCA, xlim = c(-9000, 9000), ylim = c(-4000, 4000))
```




```
# fill in circles
plotScore_ellipse(PCA, fill = TRUE,
fill.alpha = 0.2, xlim = c(-9000, 9000), ylim = c(-4000, 4000))
```


PC 2 (15.2%)
2000
0
−4000
−5000
0
5000
PC 1 (69.4%)