eNote 9

Discriminant Analysis: LDA, QDA, k-NN, Bayes, PLSDA, cart, Random forests

Indhold

9 Discriminant Analysis: LDA, QDA, k-NN, Bayes, PLSDA, cart, Random forests 1
9.1 Reading material 2
9.1.1 LDA, QDA, k-NN, Bayes 2
9.1.2 Classification trees (CART) and random forests 3
9.2 Example: Iris data 4
9.2.1 Linear Discriminant Analysis, LDA 4
9.2.2 Quadratic Discriminant Analysis, QDA 6
9.2.3 Predicting NEW data 8
9.2.4 Bayes method 9
9.2.5 k-nearest neighbourgh 12
9.2.6 PLS-DA 13
9.2.7 Classification and Regression Trees - Cart 21
9.2.8 Random forests 21
9.3 Exercises 21

9.1 Reading material

9.1.1 LDA, QDA, k-NN, Bayes

Read in the Varmuza book: (not covering CARTS and random forests)

- Section 5.1, Intro, 2.5 pages
- Section 5.2, Linear Methods, 12 pages
- Section 5.3.3, Nearest Neighbourg (k-NN), 3 pages
- Section 5.7, Evaluation of classification, 3 pages

Alternatively read in the Wehrens book: (not covering CARTS and random forests)

- 7.1 Discriminant Analysis 104
  - 7.1.1 Linear Discriminant Analysis 105
  - 7.1.2 Crossvalidation 109
  - 7.1.3 Fisher LDA 111
  - 7.1.4 Quadratic Discriminant Analysis 114
  - 7.1.5 Model-Based Discriminant Analysis 116
  - 7.1.6 Regularized Forms of Discriminant Analysis 118
- 7.2 Nearest-Neighbour Approaches 122

- 11.3 Discrimination with Fat Data Matrices 243
  - 11.3.1 PCDA 244
  - 11.3.2 PLSDA 248

9.1.2 Classification trees (CART) and random forests

Read in the Varmuza book about classification (and regression) trees:

- Section 5.4 Classification Trees
- Section 5.8.1.5 Classification Trees
- (Section 4.8.3.3 Regression Trees)

Read in the Wehrens book:

- 7.3 Tree-Based Approaches 126-135
- 9.7 Integrated Modelling and Validation 195
  - (9.7.1 Bagging 196)
  - 9.7.2 Random Forests 197
  - (9.7.3 Boosting 202)

9.2 Example: Iris data

```
# we use the iris data:
data(iris3)

Iris <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
                   Sp = rep(c("s","c","v"), rep(50,3)))

# We make a test and training data:
set.seed(4897)
train <- sample(1:150, 75)
test <- (1:150)[-train]
Iris_train <- Iris[train,]
Iris_test <- Iris[test,]

# Distribution in three classes in training data:
table(Iris_train$Sp)



 c  s  v
23 24 28
```

9.2.1 Linear Discriminant Analysis, LDA

We use the lda function from the MASS package:

```
# PART 1: LDA
lda_train_LOO <- lda(Sp ~ Sepal.L. + Sepal.W. + Petal.L. + Petal.W.,
                     Iris_train, prior = c(1, 1, 1)/3, CV = TRUE)
```

The Species factor variable is expressed as the response in a usual model expression with the four measurement variables as the x's. The CV=TRUE option choice performs full LOO cross validation. The prior option works as:

the prior probabilities of class membership. If unspecified, the class proportions for the training set are used. If present, the probabilities should be specified in the order of the factor levels.

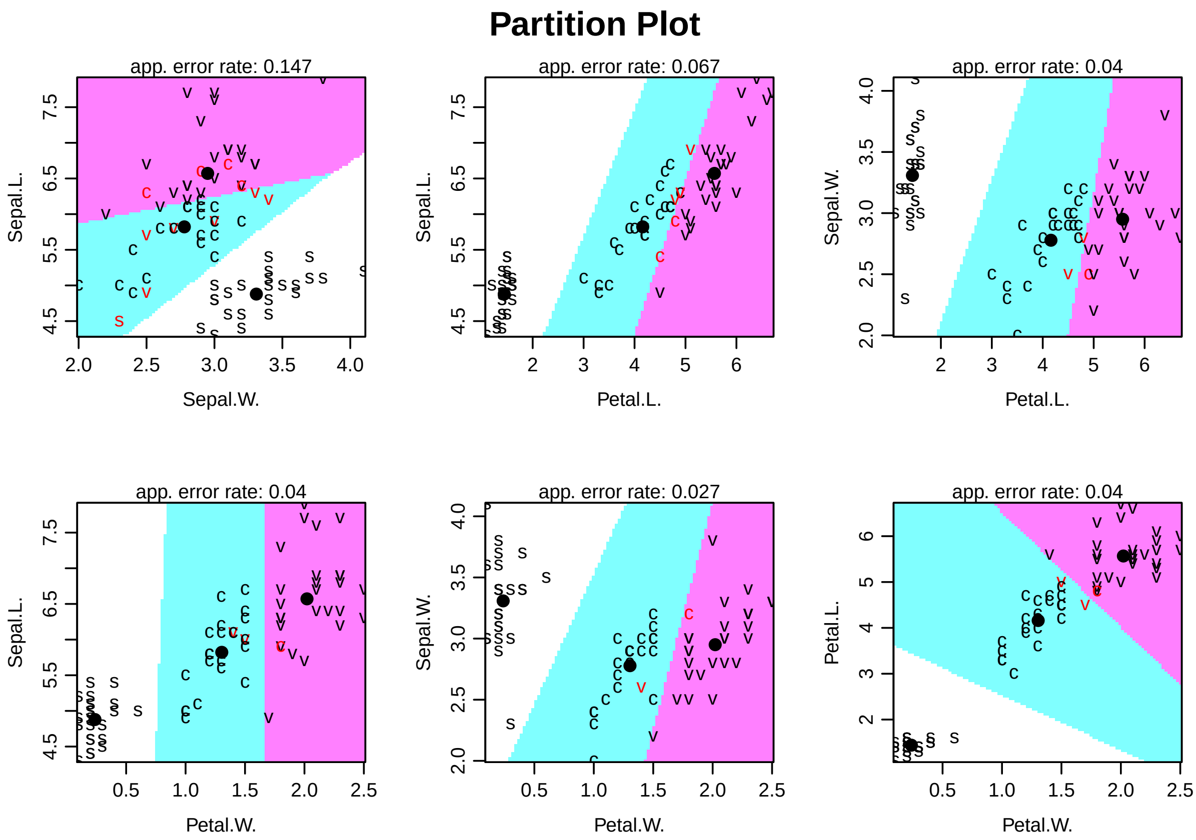
First we must assess the accuracy of the prediction based on the cross validation error, which is quantified simply as relative frequencies of errornous class predictions, either in total or detailed on the classes:

```
# Assess the accuracy of the prediction
# percent correct for each category:
ct <- table(Iris_train$Sp, lda_train_LOO$class)
diag(prop.table(ct, 1))


      c       s       v
0.91304 1.00000 1.00000


# total percent correct
sum(diag(prop.table(ct)))


[1] 0.97333
```

So the overal CV based error rate is $0.0267 = 2.7\%$.

Som nice plotting only works without the CV-stuff using the klaR-package:

```
library(klaR)
partimat(Sp ~ Sepal.L. + Sepal.W. + Petal.L. + Petal.W.,
         data = Iris_train, method = "lda", prior=c(1, 1, 1)/3)
```



9.2.2 Quadratic Discriminant Analysis, QDA

It goes very much like above:

```
# PART 2: QDA
# Most stuff from LDA can be reused:

qda_train_LOO <- qda(Sp ~ Sepal.L. + Sepal.W. + Petal.L. + Petal.W.,
                     Iris_train, prior = c(1, 1, 1)/3, CV = TRUE)

# Assess the accuracy of the prediction
# percent correct for each category:
ct <- table(Iris_train$Sp, qda_train_LOO$class)
ct
```

```
     c  s  v
  c 21  0  2
  s  0 24  0
  v  1  0 27
```

```
diag(prop.table(ct, 1))
```

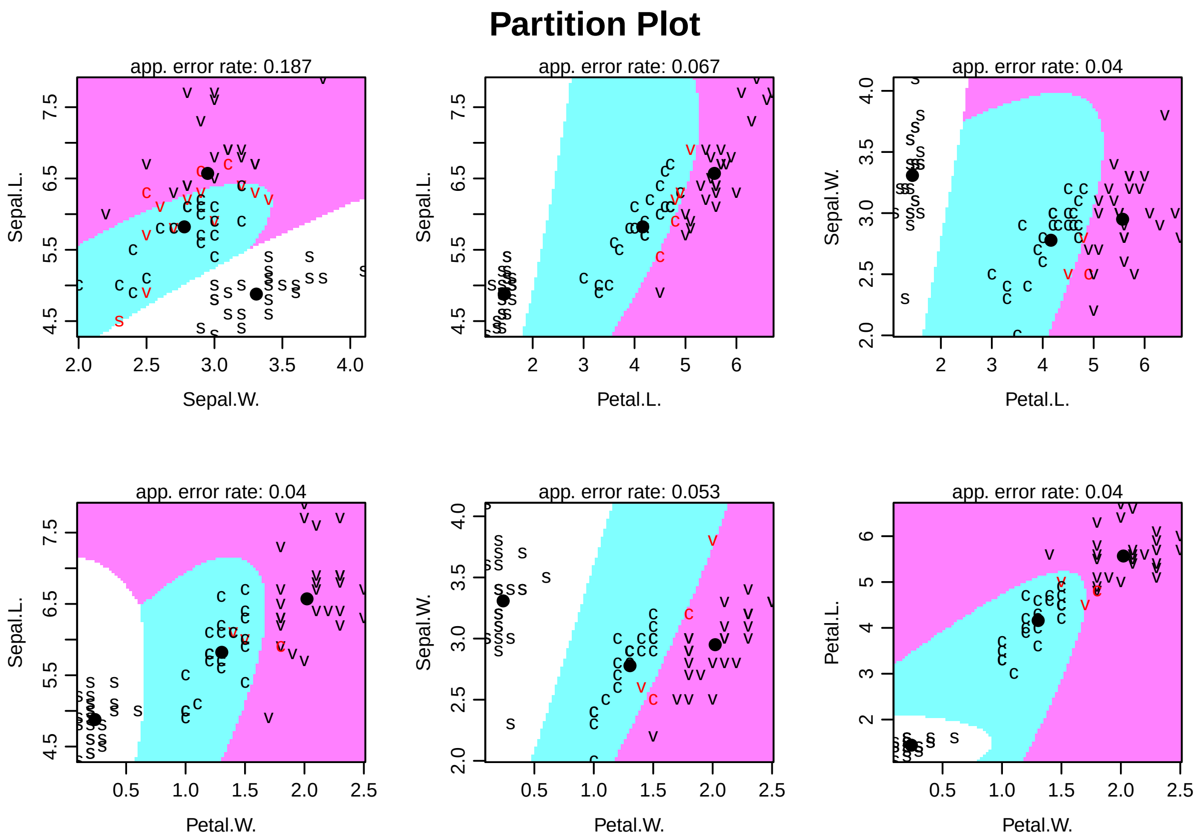
```
      c       s       v
0.91304 1.00000 0.96429
```

```
# total percent correct
sum(diag(prop.table(ct)))
```

```
[1] 0.96
```

For this example the QDA performs slightly worse than the LDA.

```
partimat(Sp ~ Sepal.L. + Sepal.W. + Petal.L. + Petal.W.,
         data = Iris_train, method = "qda", prior = c(1, 1, 1)/3)
```



9.2.3 Predicting NEW data

```
# And now predicting NEW data:
predict(qda_train, Iris_test)$class

 [1] s s s s s s s s s s s s s s s s s s s s s s s s s s c c c c c c c c c
[36] c c v c c c c c c v c c c c c c v v v v v v v v v v v v v c v v v
[71] v v v v v
Levels: c s v

# Find confusion table:
ct <- table(Iris_test$Sp, predict(qda_train, Iris_test)$class)
ct
```

```
   c  s  v
c 25  0  2
s  0 26  0
v  1  0 21
```

```
diag(prop.table(ct, 1))
```

```
      c       s       v
0.92593 1.00000 0.95455
```

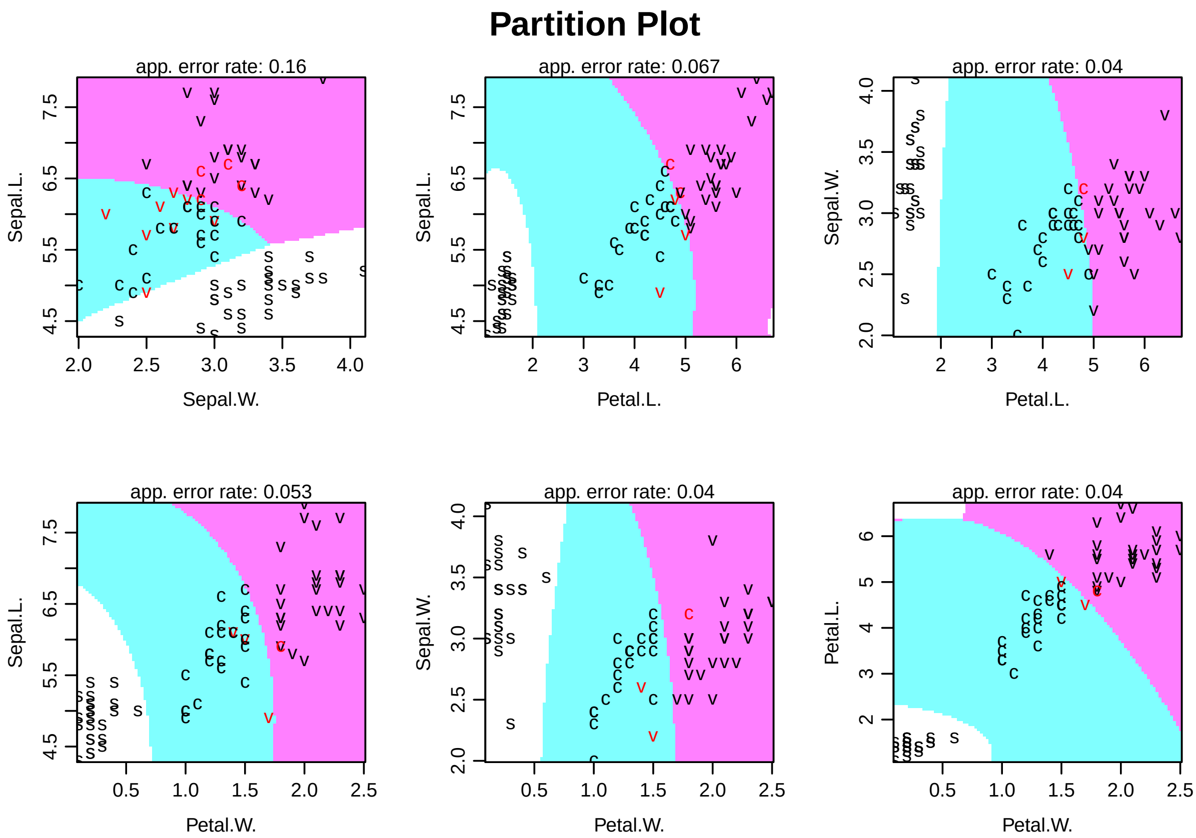
```
# total percent correct
sum(diag(prop.table(ct)))
```

```
[1] 0.96
```

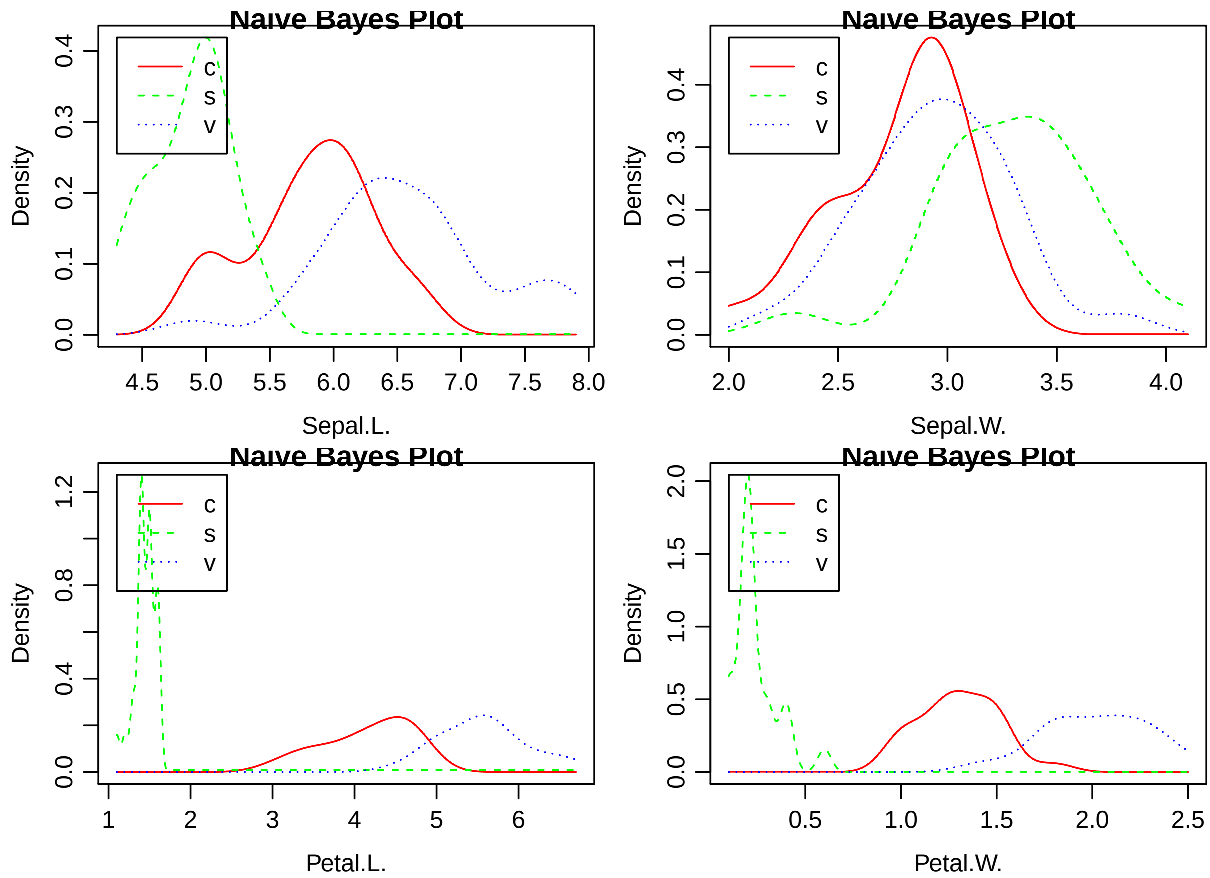
9.2.4 Bayes method

We can fit a density to the observed data and use that instead of the normal distributions implicitly used in LDA:

```
# Part 3: (Naive) Bayes method
# With "usekernel=TRUE" it fits a density to the observed data and uses that
# instead of the normal distribution

# Let's explore the results first:
# (Can take a while dur to the fitting of densities)
partimat(Sp ~ Sepal.L. + Sepal.W. + Petal.L. + Petal.W.,
         data = Iris_train, method = "naiveBayes",
         prior=c(1, 1, 1)/3, usekernel = TRUE)
```




```
bayes_fit <- suppressWarnings(NaiveBayes(Sp ~ Sepal.L. + Sepal.W. +
                                          Petal.L. + Petal.W.,
                          data = Iris_train, method = "naiveBayes",
                          prior = c(1, 1, 1)/3, usekernel = TRUE))
par(mfrow = c(2, 2))
plot(bayes_fit)
```

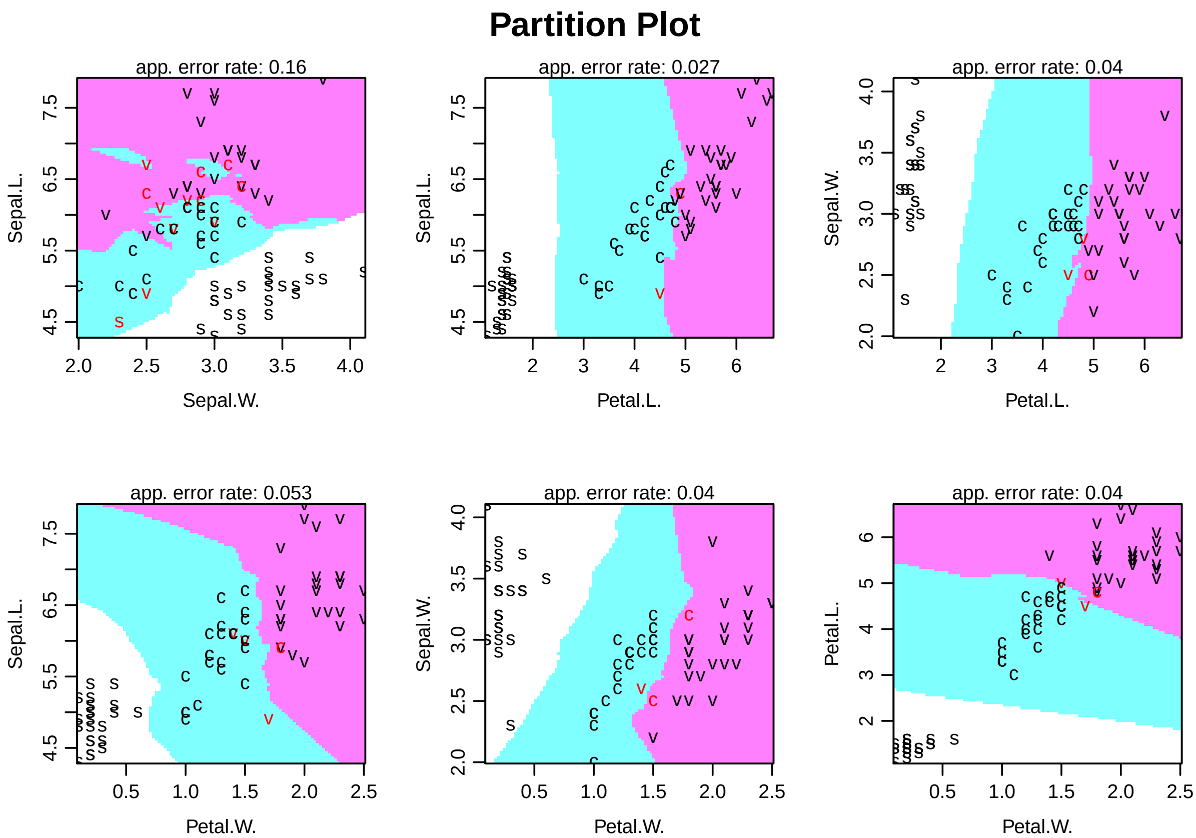



```
# And now predicting NEW data:
bayespred <- predict(bayes_fit, Iris_test)$class

# Find confusion table:
ct <- table(Iris_test$Sp, bayespred)
ct


   bayespred
     c  s  v
  c 25  0  2
  s  0 26  0
  v  1  0 21


diag(prop.table(ct, 1))


      c       s       v
0.92593 1.00000 0.95455
```

```
# total percent correct
sum(diag(prop.table(ct)))

[1] 0.96
```

9.2.5 k-nearest neighbourgh

```
# PART 4: k-nearest neighbourgh:

# Explorative plot WARNING: THIS may take some time to produce!!
partimat(Sp ~ Sepal.L. + Sepal.W. + Petal.L. + Petal.W.,
         data = Iris[train,], method = "sknn", kn = 3)
```



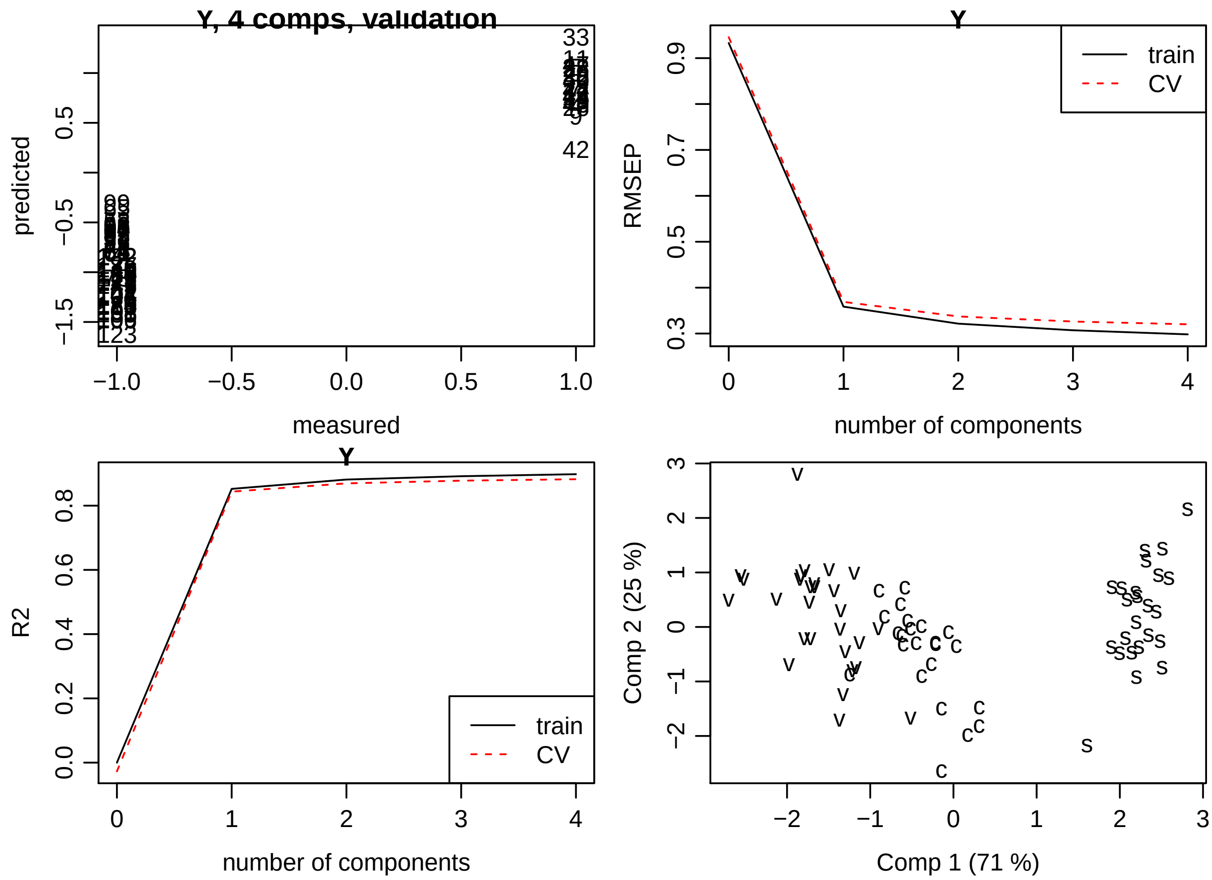
```
knn_fit_5 <- sknn(Sp ~ Sepal.L. + Sepal.W. + Petal.L. + Petal.W.,
                  data = Iris_train, method = "sknn", kn = 5)

# And now predicting NEW data:
knn_5_preds <- predict(knn_fit_5, Iris_test)$class

# Find confusion table:
ct <- table(Iris_test$Sp, knn_5_preds)
ct


   knn_5_preds
     c  s  v
  c 25  0  2
  s  0 26  0
  v  0  0 22


diag(prop.table(ct, 1))


      c       s       v
0.92593 1.00000 1.00000


# total percent correct
sum(diag(prop.table(ct)))


[1] 0.97333
```

9.2.6 PLS-DA

```
# PART 5: PLS-DA
# We have to use the "usual" PLS-functions

# Define the response vector (2 classes) OR matrix (>2) classes:
```

```
# Let's try with K=2: Group 1: s, Group -1: c and v
Iris_train$Y <- -1
Iris_train$Y[Iris_train$Sp == "s"] <- 1
table(Iris_train$Y)



-1  1
51 24


Iris_train$X <- as.matrix(Iris_train[, 1:4])
```

```
# From here use standard PLS1 predictions, e.g.:
library(pls)
# Pls:

mod_pls <- plsr(Y ~X , ncomp = 4, data =Iris_train,
                validation = "LOO", scale = TRUE, jackknife = TRUE)

# Initial set of plots:
par(mfrow = c(2, 2))
plot(mod_pls, labels = rownames(Iris_train), which = "validation")
plot(mod_pls, "validation", estimate = c("train", "CV"),
     legendpos = "topright")
plot(mod_pls, "validation", estimate = c("train", "CV"),
     val.type = "R2", legendpos = "bottomright")
pls::scoreplot(mod_pls, labels = Iris_train$Sp)
```



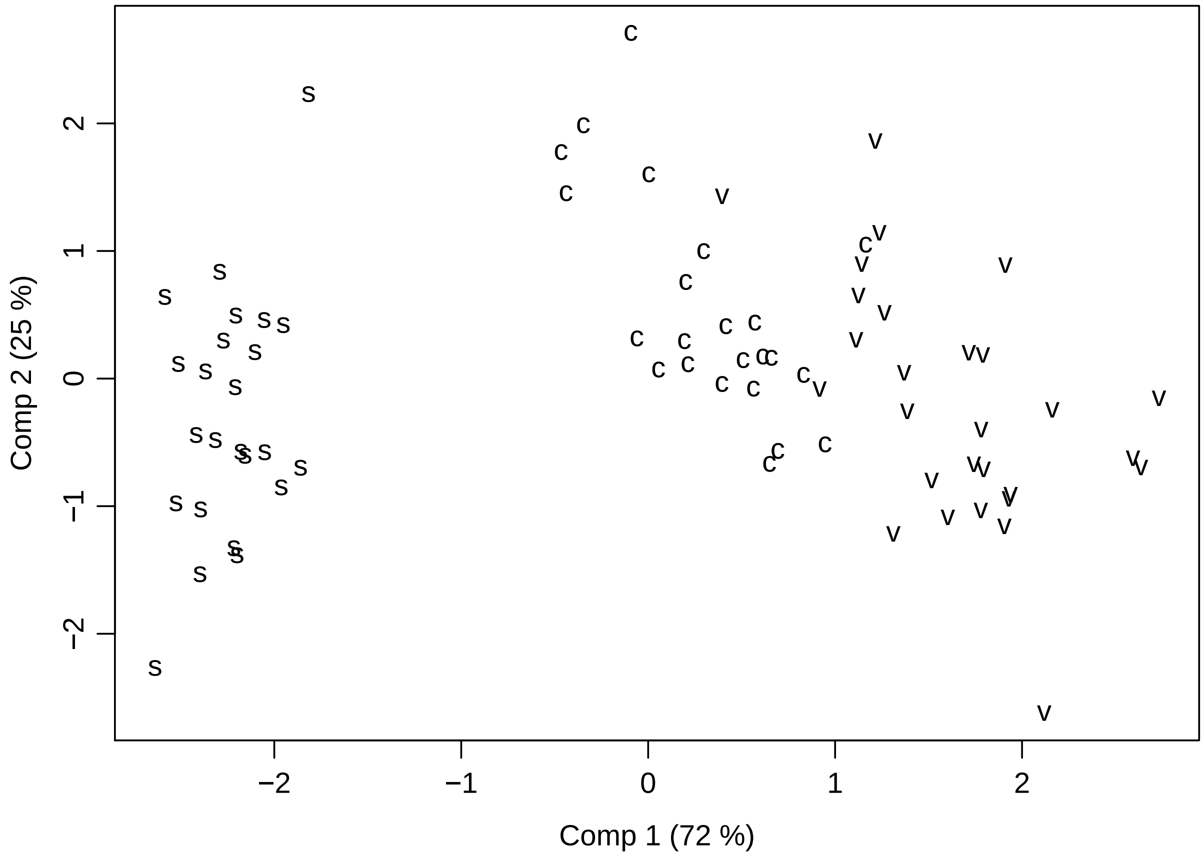
Be carefull about interpretations due to the binary setting You should do a CV based confusion table for each component, really:

```
preds <- array(dim = c(length(Iris_train[, 1]), 4))

for (i in 1:4) preds[, i] <- predict(mod_pls, ncomp = i)
preds[preds<0] <- -1
preds[preds>0] <- 1

# Look at the results from each of the components:

for (i in 1:4) {
ct <- table(Iris_train$Y, preds[,1])
CV_error <- 1-sum(diag(prop.table(ct)))
print(CV_error)
}

[1] 0
```

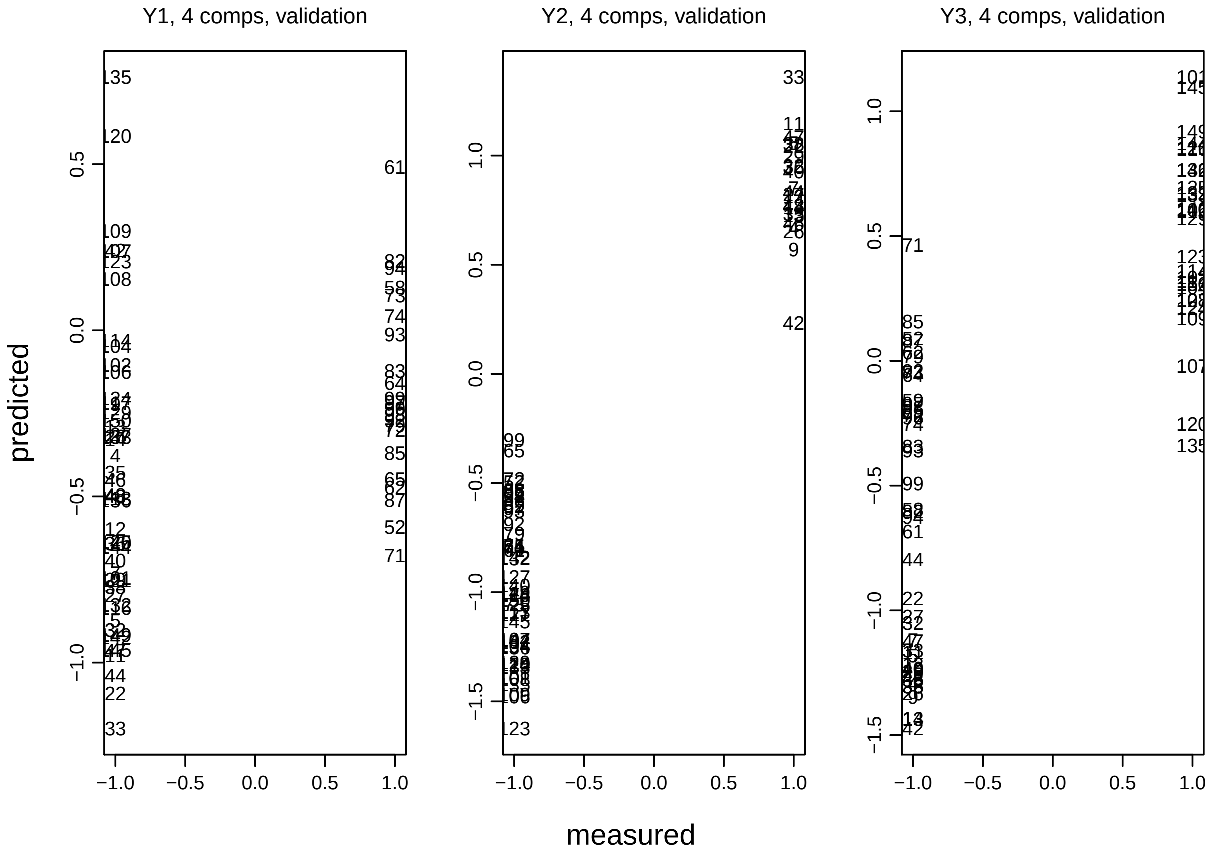
```
[1] 0
[1] 0
[1] 0
```

The prediction of new data would be handled similarly (not shown).

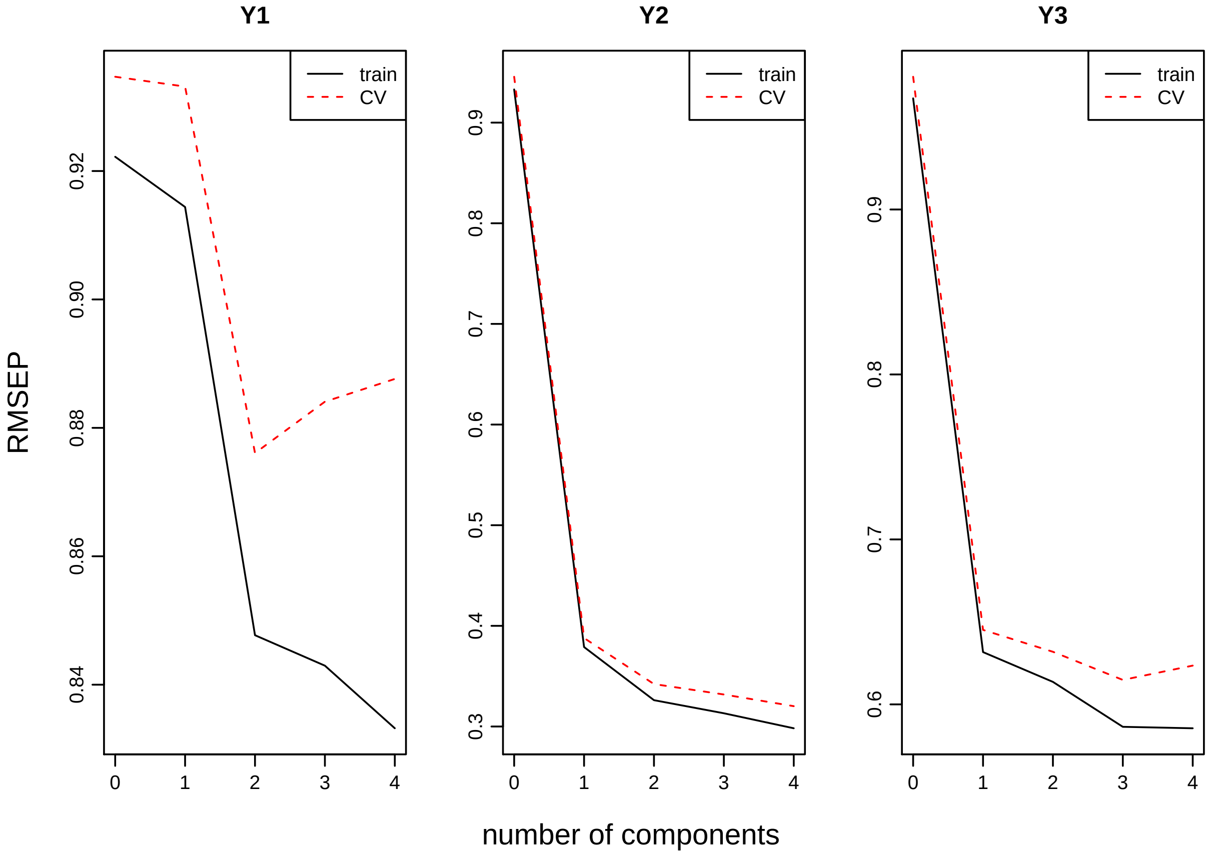
```
# WHAT if in this case K=3 classes:
# Look at Varmuza, Sec. 5.2.2.3
K=3
Iris_train$Y=matrix(rep(1,length(Iris_train[,1])*K),ncol=K)

Iris_train$Y[Iris_train$Sp!="c",1]=-1
Iris_train$Y[Iris_train$Sp!="s",2]=-1
Iris_train$Y[Iris_train$Sp!="v",3]=-1

mod_pls <- plsr(Y ~ X, ncomp = 4, data = Iris_train,
                validation="LOO", scale = TRUE, jackknife = TRUE)

# Initial set of plots:
par(mfrow=c(1, 1))
pls::scoreplot(mod_pls, labels = Iris_train$Sp)
```



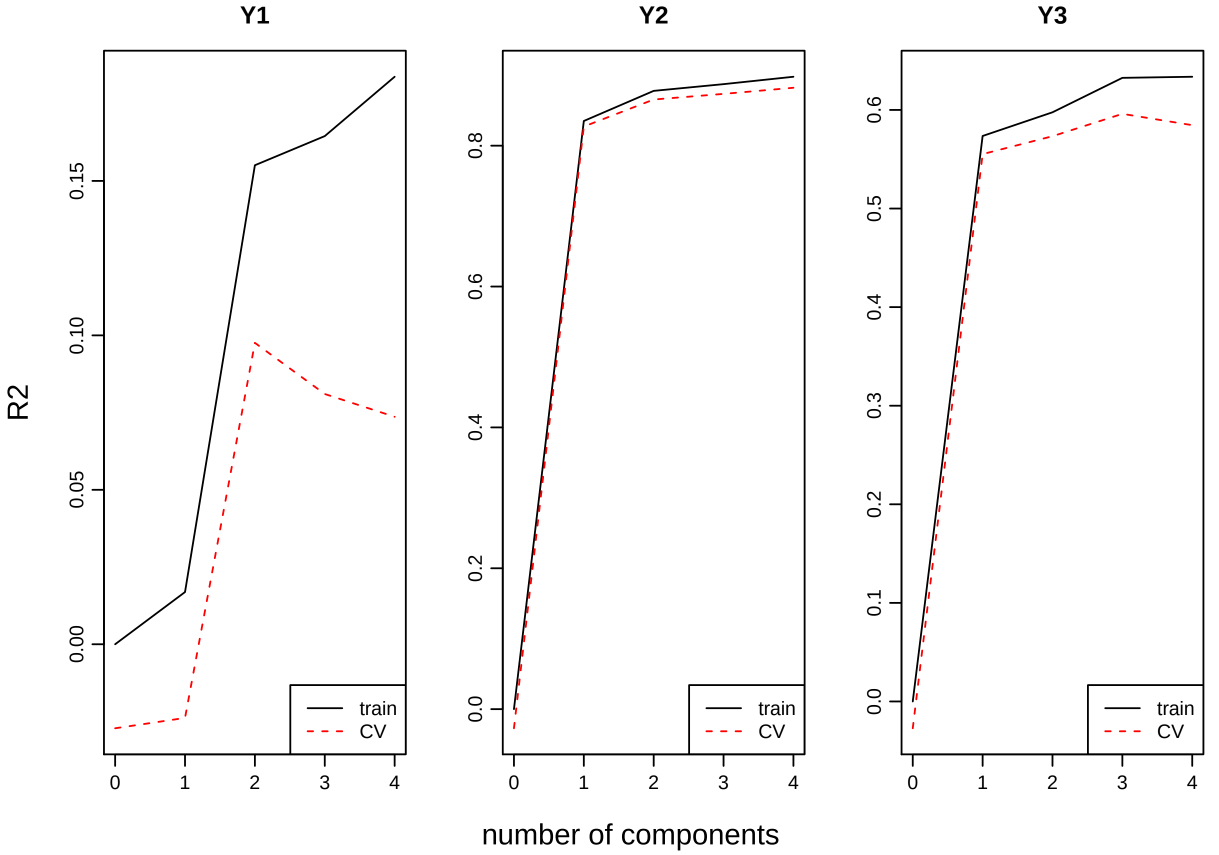

```
plot(mod_pls, labels = rownames(Iris_train), which = "validation")
```




```r
plot(mod_pls, "validation", estimate = c("train", "CV"),
     legendpos = "topright")
```




```
plot(mod_pls, "validation", estimate = c("train", "CV"),
     val.type = "R2", legendpos = "bottomright")
```




```
# Predictions from PLS need to be transformed to actual classifications:
# Select the largest one:
preds <- array(dim = c(length(Iris_train[, 1]), 4))
for (i in 1:4) preds[,i]<- apply(predict(mod_pls, ncomp = i),
                                 1, which.max)

preds2 <- array(dim = c(length(Iris_train[, 1]), 4))
preds2[preds==1] <- "c"
preds2[preds==2] <- "s"
preds2[preds==3] <- "v"

# Look at the results from each of the components:
for (i in 1:4) {
  ct <- table(Iris_train$Sp, preds2[, i])
  CV_error <- 1 - sum(diag(prop.table(ct)))
  print(CV_error)
}
```

```
[1] 0.96
[1] 0.24
[1] 0.2
[1] 0.17333
```

9.2.7 Classification and Regression Trees - Cart

9.2.8 Random forests

9.3 Exercises

|||| Exercise 1 Exercise: Wine data

Use the wine data previously used.

a) Predict the wine-class using the different methods suggested.

b) Try also a PCA based version of some of the methods.

c) What are the results? Make your own test- and training data. Apply CV on the training data and predict the test data.

d) By looking at various plots (in addition to prediction ability) consider which models seem to be mostly appropriate for the data.

Exercise 2 Exercise: Random forest exercise