ML supervised practical

IMohorianu

04/03/2021

```r
setwd("~/Dropbox/CSCI/training/BBS/MachineLearning/")
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```r
library(ggfortify)
library(readr)
library(car)
```

```
## Loading required package: carData
```

```r
library(gridExtra)
library(grid)
library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:car':
##
##     recode

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(reshape2)
library(caTools)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
##
## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':
##
##     nasa
```

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.6.2
```

```r
library(C50)
library(tree)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(class)
library(gmodels)
library(dendextend)
```

```
## Warning: package 'dendextend' was built under R version 3.6.2

##
## ---------------------
## Welcome to dendextend version 1.14.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------

##
## Attaching package: 'dendextend'

## The following object is masked from 'package:stats':
##
##     cutree
```

```r
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
library(caret)
```

```
## Loading required package: lattice
```

The Iris data

```r
data(iris)      ##loads the dataset, which can be accessed under the variable name iris
summary(iris)   ##presents the 5 figure summary of the dataset
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##
##
##
```

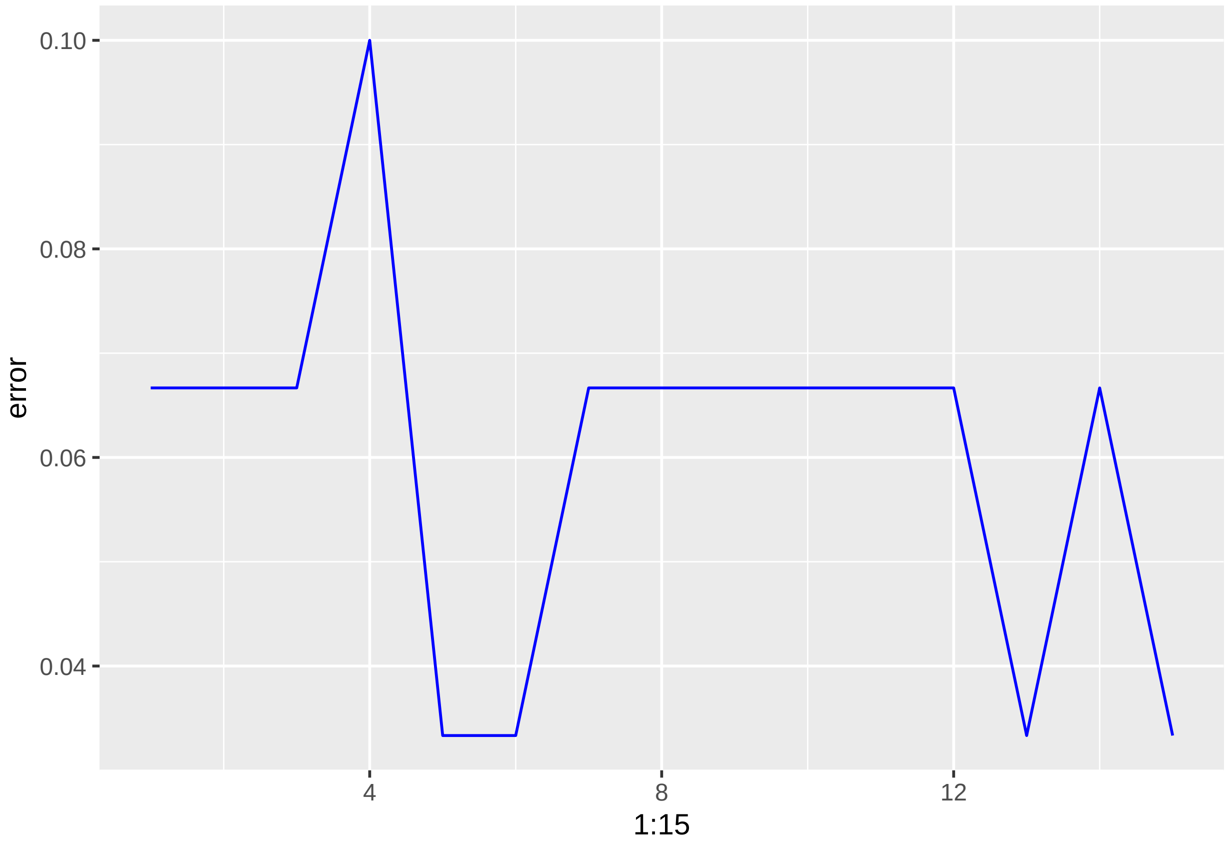
k nearest neighbours

Divide the Iris dataset into training and test dataset to apply KNN classification. 80% of the data is used for training while the KNN classification is tested on the remaining 20% of the data.

```r
iris[,1:4] <- scale(iris[,1:4])
setosa<- rbind(iris[iris$Species=="setosa",])
versicolor<- rbind(iris[iris$Species=="versicolor",])
virginica<- rbind(iris[iris$Species=="virginica",])

ind <- sample(1:nrow(setosa), nrow(setosa)*0.8)
iris.train<- rbind(setosa[ind,], versicolor[ind,], virginica[ind,])
iris.test<- rbind(setosa[-ind,], versicolor[-ind,], virginica[-ind,])
iris[,1:4] <- scale(iris[,1:4])
```

Then train and evaluate. Determine the optimal number of neighbours.

```r
error <- c()
for (i in 1:15)
{
  knn.fit <- knn(train = iris.train[,1:4], test = iris.test[,1:4], cl = iris.train$Species, k = i)
  error[i] = 1- mean(knn.fit == iris.test$Species)
}

ggplot(data = data.frame(error), aes(x = 1:15, y = error)) +
  geom_line(color = "Blue")
```



```r
iris_test_pred1 <- knn(train = iris.train[,1:4],
                       test = iris.test[,1:4],
                       cl = iris.train$Species,k = 10,prob=TRUE)
table(iris.test$Species,iris_test_pred1)
```

```
##             iris_test_pred1
##              setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         0
##   virginica       0          2         8
```

```r
CrossTable(x = iris.test$Species, y = iris_test_pred1,prop.chisq=FALSE)
```

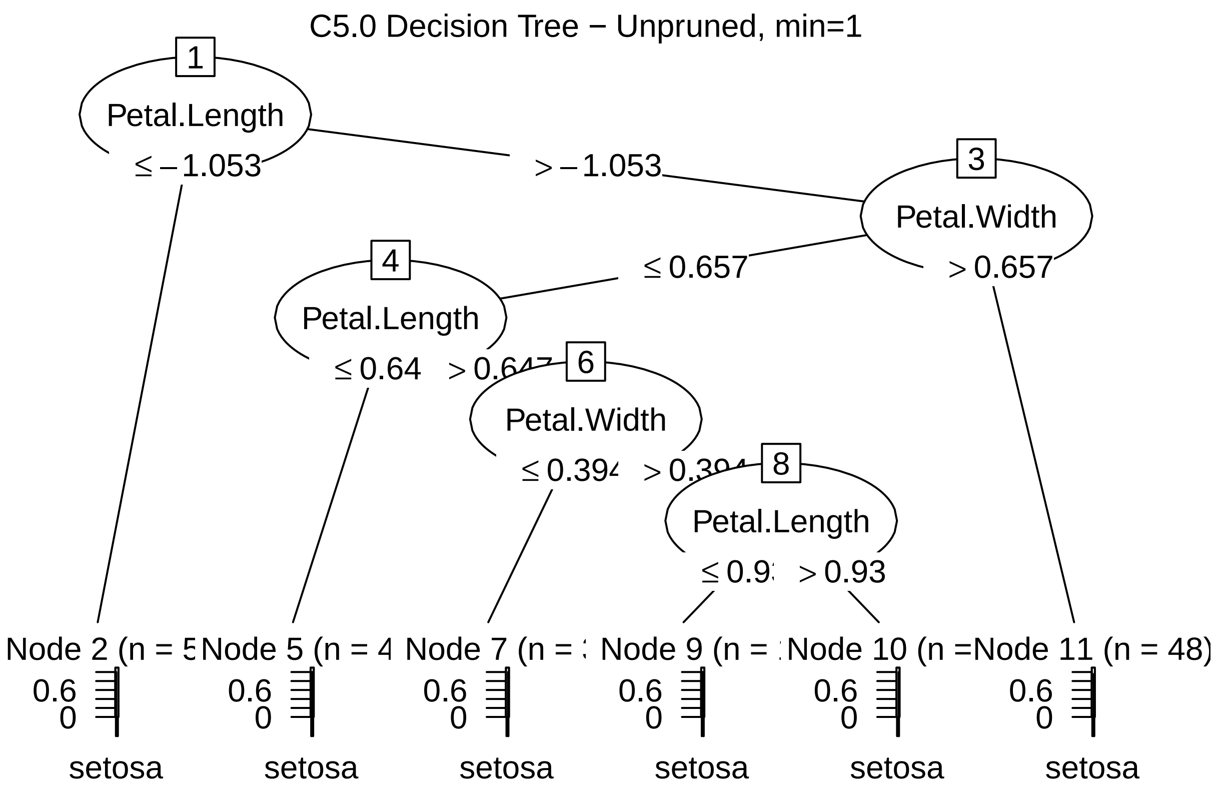
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  30
##
##
##                   | iris_test_pred1
## iris.test$Species |     setosa | versicolor |  virginica |  Row Total |
## ------------------|------------|------------|------------|------------|
##            setosa |         10 |          0 |          0 |         10 |
##                   |      1.000 |      0.000 |      0.000 |      0.333 |
##                   |      1.000 |      0.000 |      0.000 |            |
##                   |      0.333 |      0.000 |      0.000 |            |
## ------------------|------------|------------|------------|------------|
##        versicolor |          0 |         10 |          0 |         10 |
##                   |      0.000 |      1.000 |      0.000 |      0.333 |
##                   |      0.000 |      0.833 |      0.000 |            |
##                   |      0.000 |      0.333 |      0.000 |            |
## ------------------|------------|------------|------------|------------|
##         virginica |          0 |          2 |          8 |         10 |
##                   |      0.000 |      0.200 |      0.800 |      0.333 |
##                   |      0.000 |      0.167 |      1.000 |            |
##                   |      0.000 |      0.067 |      0.267 |            |
## ------------------|------------|------------|------------|------------|
##      Column Total |         10 |         12 |          8 |         30 |
##                   |      0.333 |      0.400 |      0.267 |            |
## ------------------|------------|------------|------------|------------|
##
##
```

Decision Trees and Random Forrest

```
library(C50)
input <- iris[,1:4]
output <- iris[,5]
model1 <- C5.0(input, output, control = C5.0Control(noGlobalPruning = TRUE,minCases=1))
summary(model1)
```

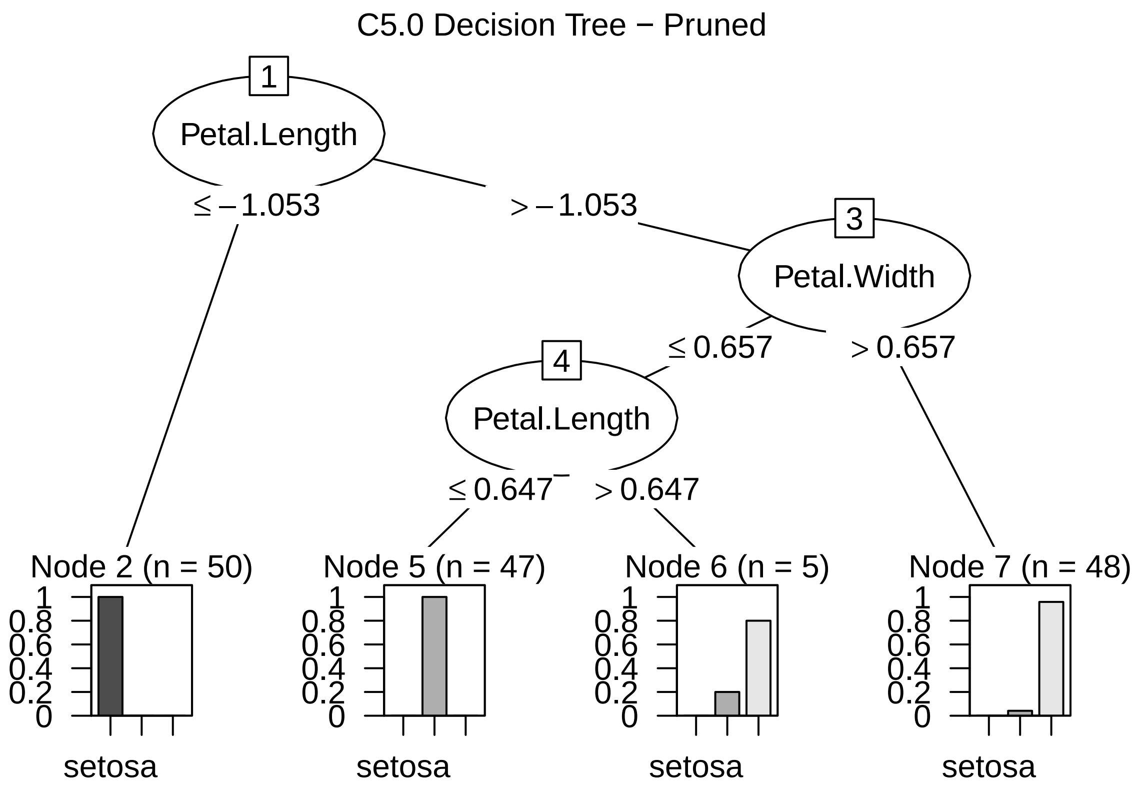
```
##
## Call:
## C5.0.default(x = input, y = output, control = C5.0Control(noGlobalPruning
##  = TRUE, minCases = 1))
##
##
## C5.0 [Release 2.07 GPL Edition]      Thu Mar  4 17:45:53 2021
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 150 cases (5 attributes) from undefined.data
##
## Decision tree:
##
## Petal.Length <= -1.052513: setosa (50)
## Petal.Length > -1.052513:
## :...Petal.Width > 0.656838: virginica (46/1)
##     Petal.Width <= 0.656838:
##     :...Petal.Length <= 0.6469162: versicolor (48/1)
##         Petal.Length > 0.6469162:
##         :...Petal.Width <= 0.3944527: virginica (3)
##             Petal.Width > 0.3944527:
##             :...Petal.Length <= 0.9301544: versicolor (2)
##                 Petal.Length > 0.9301544: virginica (1)
##
##
## Evaluation on training data (150 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##       6    2( 1.3%)   <<
##
##
##     (a)   (b)   (c)    <-classified as
##    ----  ----  ----
##      50                (a): class setosa
##            49     1    (b): class versicolor
##             1    49    (c): class virginica
##
##
##  Attribute usage:
##
##  100.00% Petal.Length
##   66.67% Petal.Width
##
##
## Time: 0.0 secs
```

```
plot(model1, main="C5.0 Decision Tree - Unpruned, min=1")
```

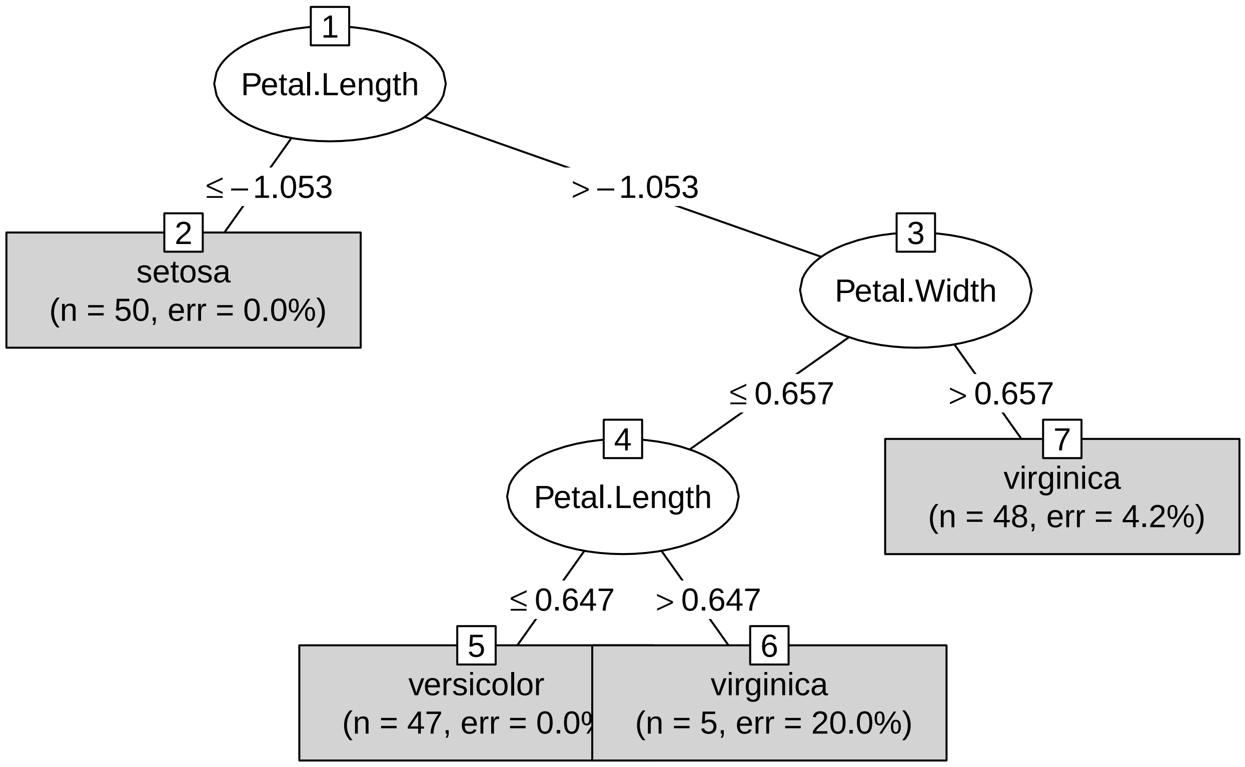


We can play with the parameters of the classifier to see better/simpler/more complete/more complex trees. Here's a simpler one:

```r
model2 <- C5.0(input, output, control = C5.0Control(noGlobalPruning = FALSE))
plot(model2, main="C5.0 Decision Tree - Pruned")
```



```r
plot(model2, type='simple')
```



```r
summary(model2)
```

```
##
## Call:
## C5.0.default(x = input, y = output, control = C5.0Control(noGlobalPruning
##  = FALSE))
##
##
## C5.0 [Release 2.07 GPL Edition]      Thu Mar  4 17:45:54 2021
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 150 cases (5 attributes) from undefined.data
##
## Decision tree:
##
## Petal.Length <= -1.052513: setosa (50)
## Petal.Length > -1.052513:
## :...Petal.Width > 0.656838: virginica (46/1)
##     Petal.Width <= 0.656838:
##     :...Petal.Length <= 0.6469162: versicolor (48/1)
##         Petal.Length > 0.6469162: virginica (6/2)
##
##
## Evaluation on training data (150 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##       4    4( 2.7%)   <<
##
##
##     (a)   (b)   (c)    <-classified as
##    ----  ----  ----
##      50                (a): class setosa
##            47     3    (b): class versicolor
##             1    49    (c): class virginica
##
##
##  Attribute usage:
##
##  100.00% Petal.Length
##   66.67% Petal.Width
##
##
## Time: 0.0 secs
```

```
#We can "zoom into" the usage of features for creation of the model:
C5imp(model2,metric='usage')
```

```
##              Overall
## Petal.Length  100.00
## Petal.Width    66.67
## Sepal.Length    0.00
## Sepal.Width     0.00
```

```
#Now we have a model. Can we predict the class from the numerical attributes?
newcases <- iris[c(1:3,51:53,101:103),]
newcases
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1    -0.89767388  1.01560199   -1.3357516  -1.3110521     setosa
## 2    -1.13920048 -0.13153881   -1.3357516  -1.3110521     setosa
## 3    -1.38072709  0.32731751   -1.3923993  -1.3110521     setosa
## 51    1.39682886  0.32731751    0.5336209   0.2632600 versicolor
## 52    0.67224905  0.32731751    0.4203256   0.3944526 versicolor
## 53    1.27606556  0.09788935    0.6469162   0.3944526 versicolor
## 101   0.55148575  0.55674567    1.2700404   1.7063794  virginica
## 102  -0.05233076 -0.81982329    0.7602115   0.9192234  virginica
## 103   1.51759216 -0.13153881    1.2133927   1.1816087  virginica
```

```
predicted <- predict(model2, newcases, type="class")
predicted
```

```
## [1] setosa     setosa     setosa     versicolor versicolor versicolor virginica
## [8] virginica  virginica
## Levels: setosa versicolor virginica
```

Random Forest

```
library(randomForest)
ind <- sample(2,nrow(iris),replace=TRUE,prob=c(0.7,0.3))
```

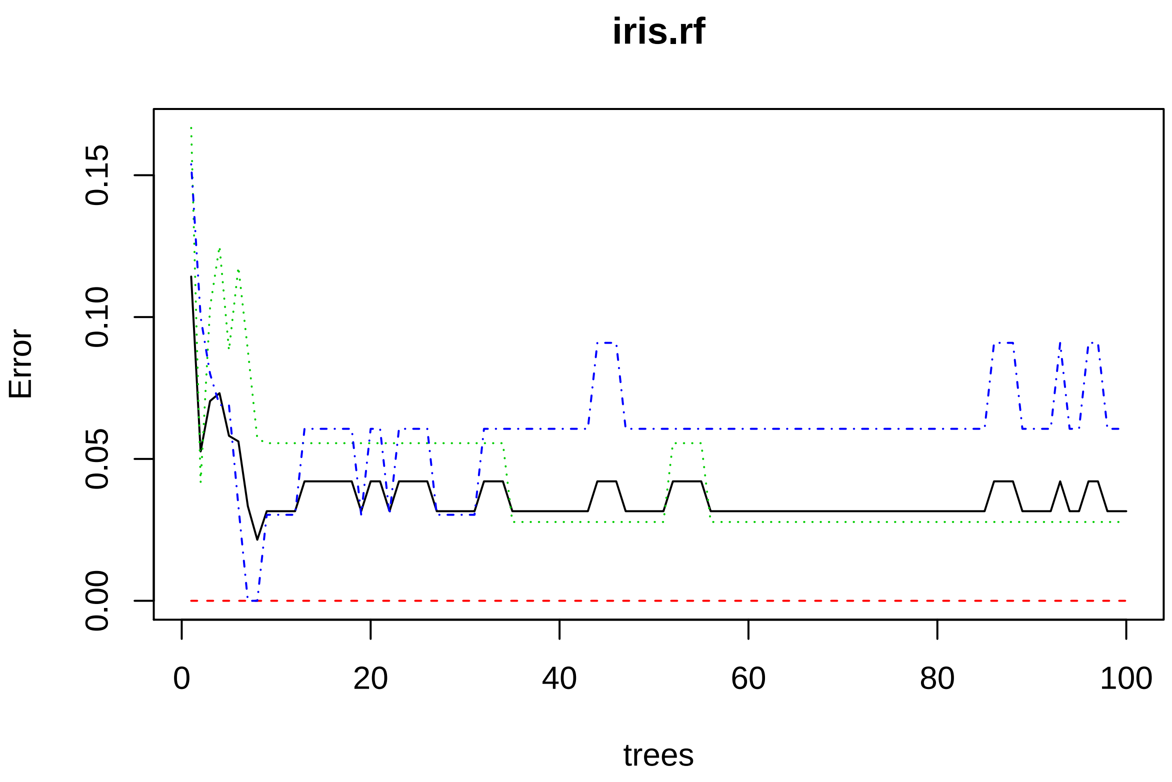
```r
trainData <- iris[ind==1,]
testData <- iris[ind==2,]
iris.rf <- randomForest(Species~.,data=trainData,ntree=100,proximity=TRUE)
table(predict(iris.rf),trainData$Species)
```

```
##
##                setosa versicolor virginica
##   setosa           26          0         0
##   versicolor        0         35         2
##   virginica         0          1        31
```

```r
print(iris.rf)
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = trainData, ntree = 100,      proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 3.16%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         26          0         0  0.00000000
## versicolor      0         35         1  0.02777778
## virginica       0          2        31  0.06060606
```

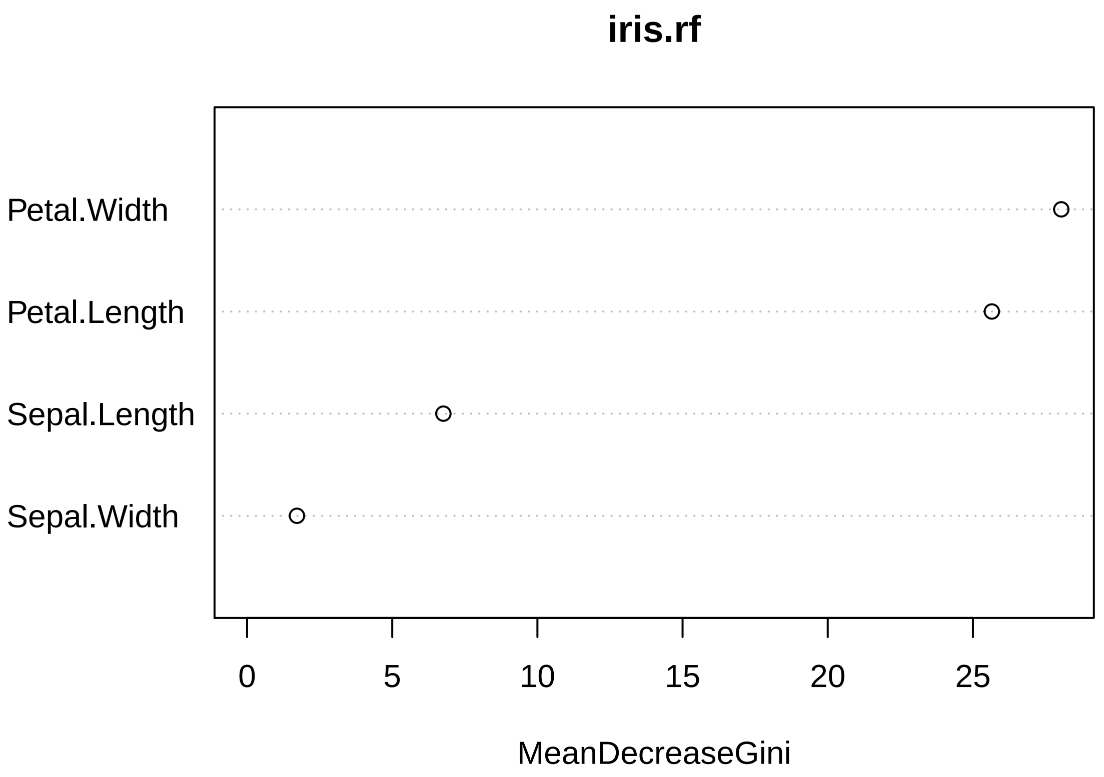
```r
plot(iris.rf)
```



```r
importance(iris.rf)
```

```
##                MeanDecreaseGini
## Sepal.Length          6.760831
## Sepal.Width           1.718308
## Petal.Length         25.652968
## Petal.Width          28.043893
```

```
varImpPlot(iris.rf)
```



```
iris.pred<-predict(iris.rf,newdata=testData)
table(iris.pred, testData$Species)
```

```
##
## iris.pred    setosa versicolor virginica
##   setosa         24          0         0
##   versicolor      0         12         1
##   virginica       0          2        16
```

As observed for the previous examples, the discriminative features are the petal length and the petal width.

SVMs

```
library(e1071)
library(caTools)
my.split = sample.split(iris$Species, SplitRatio = .8)
training_set = subset(iris, my.split == TRUE)
test_set = subset(iris, my.split == FALSE)
nrow(training_set)
```

```
## [1] 120
```

```
training_set[,1:4] = scale(training_set[,1:4])
test_set[,1:4] = scale(test_set[,1:4])
```

```
classifier1 = svm(formula = Species~., data = training_set, type = 'C-classification', kernel = 'radial
classifier2 = svm(formula = Species~ Petal.Width + Petal.Length, data = training_set, type = 'C-classifi

classifier1
```

```
##
## Call:
## svm(formula = Species ~ ., data = training_set, type = "C-classification",
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  46
```

```
classifier2
```

```
##
## Call:
## svm(formula = Species ~ Petal.Width + Petal.Length, data = training_set,
##     type = "C-classification", kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  30
```

```
test_pred1 = predict(classifier1, type = 'response', newdata = test_set[-5])
test_pred2 = predict(classifier2, type = 'response', newdata = test_set[-5])
# Making Confusion Matrix
cm1 = table(test_set[,5], test_pred1)
cm2 = table(test_set[,5], test_pred2)

cm1
```

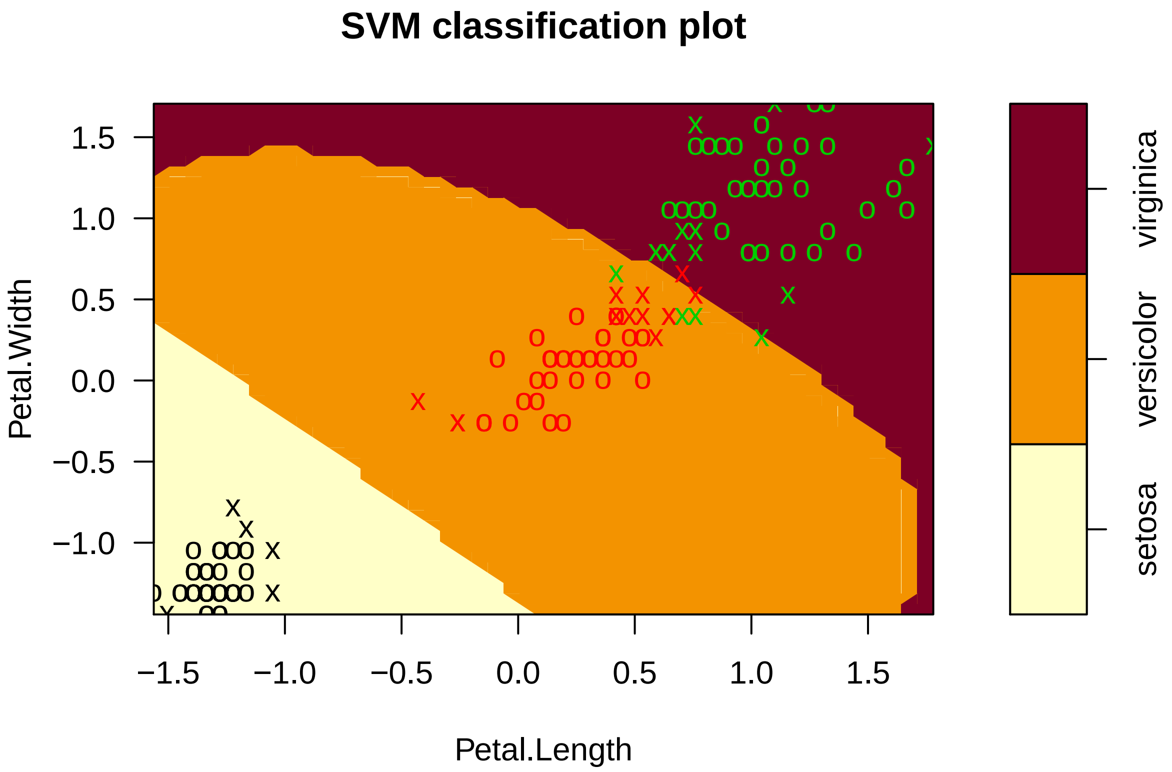
```
##             test_pred1
##              setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         0
##   virginica       0          1         9
```

```
cm2
```

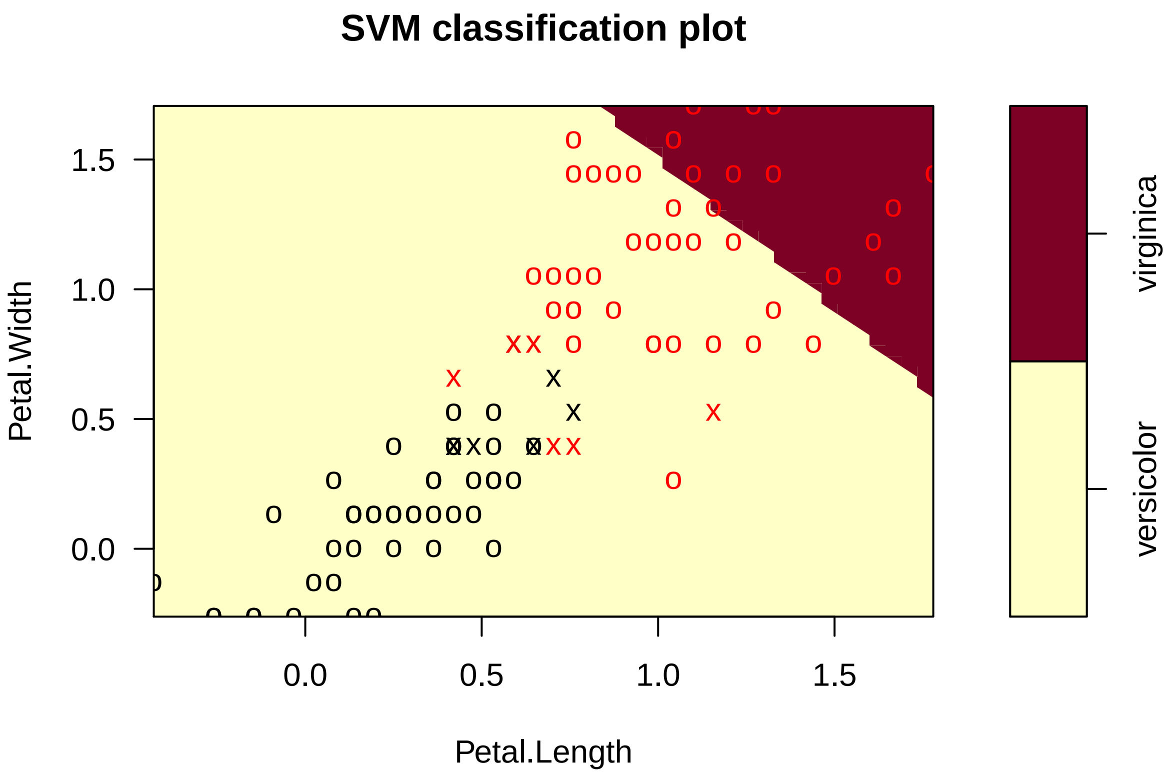
```
##             test_pred2
##              setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0          9         1
##   virginica       0          1         9
```

The accuracy for both model looks solid. Also notice that as we had deduced, only Petal Length and Width is important to make this model accurate and our second classifier proves it!

```r
m2 <- svm(Species ~ Petal.Width + Petal.Length, data = iris)
plot(m2, iris, Petal.Width ~ Petal.Length,
      slice = list(Sepal.Width = 3, Sepal.Length = 4))
```



```r
iris.part = subset(iris, Species != 'setosa')
iris.part$Species = factor(iris.part$Species)
#iris.part = iris.part[, c(1,2,5)]
svm.fit = svm(formula=Species~., data=iris.part, type='C-classification', kernel='linear')
plot(svm.fit, iris.part, Petal.Width ~ Petal.Length, slice = list(Sepal.Width = 3, Sepal.Length = 4))
```



Brain blood barrier data; using the classiiers for regressions

k-nn can also be applied to the problem of regression as we will see in the following example. The BloodBrain dataset in the caret package contains data on 208 chemical compounds, organized in two objects:

logBBB - a vector of the log ratio of the concentration of a chemical compound in the brain and the concentration in the blood. bbbDescr - a data frame of 134 molecular descriptors of the compounds.

```r
data(BloodBrain)
str(bbbDescr)
```

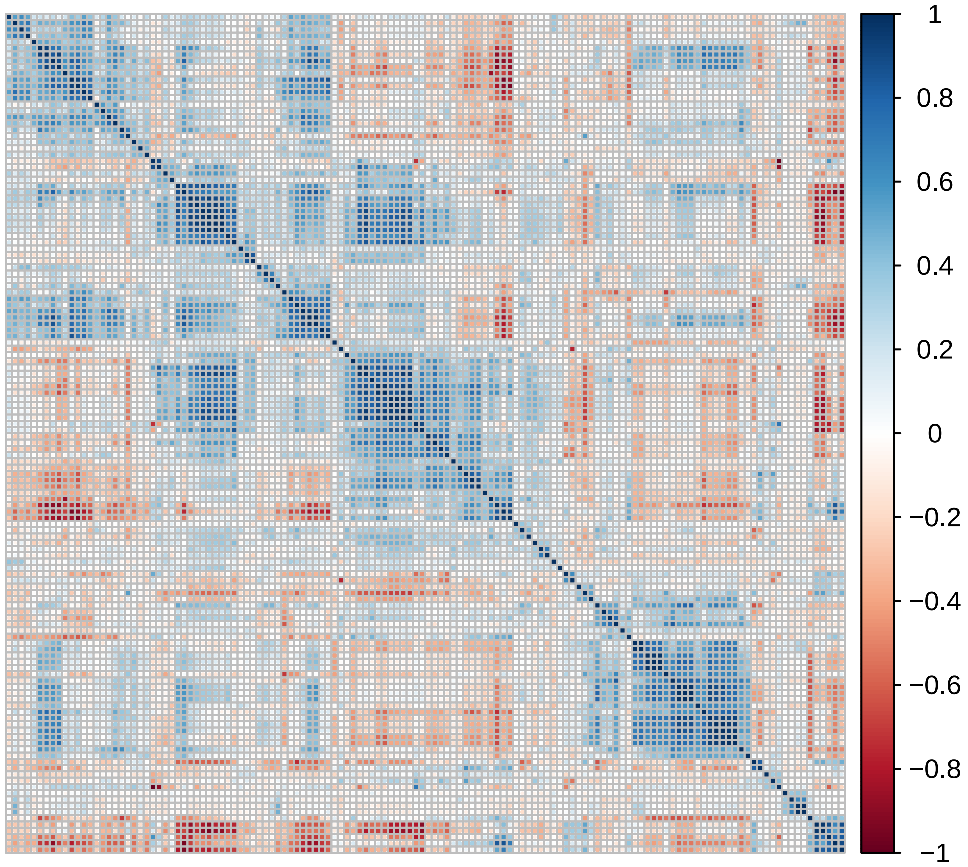
```
## 'data.frame':    208 obs. of  134 variables:
##  $ tpsa               : num  12 49.3 50.5 37.4 37.4 ...
##  $ nbasic             : int  1 0 1 0 1 1 1 1 1 1 ...
##  $ negative           : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ vsa_hyd            : num  167.1 92.6 295.2 319.1 299.7 ...
##  $ a_aro              : int  0 6 15 15 12 11 6 12 12 6 ...
##  $ weight             : num  156 151 366 383 326 ...
##  $ peoe_vsa.0         : num  76.9 38.2 58.1 62.2 74.8 ...
##  $ peoe_vsa.1         : num  43.4 25.5 124.7 124.7 118 ...
##  $ peoe_vsa.2         : num  0 0 21.7 13.2 33 ...
##  $ peoe_vsa.3         : num  0 8.62 8.62 21.79 0 ...
##  $ peoe_vsa.4         : num  0 23.3 17.4 0 0 ...
##  $ peoe_vsa.5         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ peoe_vsa.6         : num  17.24 0 8.62 8.62 8.62 ...
##  $ peoe_vsa.0.1       : num  18.7 49 83.8 83.8 83.8 ...
##  $ peoe_vsa.1.1       : num  43.5 0 49 68.8 36.8 ...
##  $ peoe_vsa.2.1       : num  0 0 0 0 0 ...
##  $ peoe_vsa.3.1       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ peoe_vsa.4.1       : num  0 0 5.68 5.68 5.68 ...
##  $ peoe_vsa.5.1        : num  0 13.567 2.504 0 0.137 ...
##  $ peoe_vsa.6.1        : num  0 7.9 2.64 2.64 2.5 ...
##  $ a_acc               : int  0 2 2 2 2 2 2 2 0 2 ...
##  $ a_acid              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ a_base              : int  1 0 1 1 1 1 1 1 1 1 ...
##  $ vsa_acc             : num  0 13.57 8.19 8.19 8.19 ...
##  $ vsa_acid            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ vsa_base            : num  5.68 0 0 0 0 ...
##  $ vsa_don             : num  5.68 5.68 5.68 5.68 5.68 ...
##  $ vsa_other           : num  0 28.1 43.6 28.3 19.6 ...
##  $ vsa_pol             : num  0 13.6 0 0 0 ...
##  $ slogp_vsa0          : num  18 25.4 14.1 14.1 14.1 ...
##  $ slogp_vsa1          : num  0 23.3 34.8 34.8 34.8 ...
##  $ slogp_vsa2          : num  3.98 23.86 0 0 0 ...
##  $ slogp_vsa3          : num  0 0 76.2 76.2 76.2 ...
##  $ slogp_vsa4          : num  4.41 0 3.19 3.19 3.19 ...
##  $ slogp_vsa5          : num  32.9 0 9.51 0 0 ...
##  $ slogp_vsa6          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ slogp_vsa7          : num  0 70.6 148.1 144 140.7 ...
##  $ slogp_vsa8          : num  113.2 0 75.5 75.5 75.5 ...
##  $ slogp_vsa9          : num  33.3 41.3 28.3 55.5 26 ...
##  $ smr_vsa0            : num  0 23.86 12.63 3.12 3.12 ...
##  $ smr_vsa1            : num  18 25.4 27.8 27.8 27.8 ...
##  $ smr_vsa2            : num  4.41 0 0 0 0 ...
##  $ smr_vsa3            : num  3.98 5.24 8.43 8.43 8.43 ...
##  $ smr_vsa4            : num  0 20.8 29.6 21.4 20.3 ...
##  $ smr_vsa5            : num  113.2 70.6 235.1 235.1 234.6 ...
##  $ smr_vsa6            : num  0 5.26 76.25 76.25 76.25 ...
##  $ smr_vsa7            : num  66.2 33.3 0 31.3 0 ...
##  $ tpsa.1              : num  16.6 49.3 51.7 38.6 38.6 ...
##  $ logp.o.w.           : num  2.948 0.889 4.439 5.254 3.8 ...
##  $ frac.anion7.        : num  0 0.001 0 0 0 0 0.001 0 0 0 ...
##  $ frac.cation7.       : num  0.999 0 0.986 0.986 0.986 0.986 0.996 0.946 0.999 0.976 ...
##  $ andrewbind          : num  3.4 -3.3 12.8 12.8 10.3 10 10.4 15.9 12.9 9.5 ...
##  $ rotatablebonds      : int  3 2 8 8 8 8 8 7 4 5 ...
##  $ mlogp               : num  2.5 1.06 4.66 3.82 3.27 ...
##  $ clogp               : num  2.97 0.494 5.137 5.878 4.367 ...
##  $ mw                  : num  155 151 365 382 325 ...
##  $ nocount             : int  1 3 5 4 4 4 4 3 2 4 ...
##  $ hbdnr               : int  1 2 1 1 1 1 2 1 1 0 ...
##  $ rule.of.5violations : int  0 0 1 1 0 0 0 0 1 0 ...
##  $ alert               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ prx                 : int  0 1 6 2 2 2 1 0 0 4 ...
##  $ ub                  : num  0 3 5.3 5.3 4.2 3.6 3 4.7 4.2 3 ...
##  $ pol                 : int  0 2 3 3 2 2 2 3 4 1 ...
##  $ inthb               : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ adistm              : num  0 395 1365 703 746 ...
##  $ adistd              : num  0 10.9 25.7 10 10.6 ...
##  $ polar_area          : num  21.1 117.4 82.1 65.1 66.2 ...
##  $ nonpolar_area       : num  379 248 638 668 602 ...
##  $ psa_npsa            : num  0.0557 0.4743 0.1287 0.0974 0.11 ...
##  $ tcsa                : num  0.0097 0.0134 0.0111 0.0108 0.0118 0.0111 0.0123 0.0099 0.0106 0.0115
##  $ tcpa                : num  0.1842 0.0417 0.0972 0.1218 0.1186 ...
##  $ tcnp                : num  0.0103 0.0198 0.0125 0.0119 0.013 0.0125 0.0162 0.011 0.0109 0.0122 ..
##  $ ovality             : num  1.1 1.12 1.3 1.3 1.27 ...
##  $ surface_area        : num  400 365 720 733 668 ...
##  $ volume              : num  656 555 1224 1257 1133 ...
##  $ most_negative_charge: num  -0.617 -0.84 -0.801 -0.761 -0.857 ...
##  $ most_positive_charge: num  0.307 0.497 0.541 0.48 0.455 ...
##  $ sum_absolute_charge : num  3.89 4.89 7.98 7.93 7.85 ...
##  $ dipole_moment       : num  1.19 4.21 3.52 3.15 3.27 ...
##  $ homo                : num  -9.67 -8.96 -8.63 -8.56 -8.67 ...
##  $ lumo                : num  3.4038 0.1942 0.0589 -0.2651 0.3149 ...
##  $ hardness            : num  6.54 4.58 4.34 4.15 4.49 ...
##  $ ppsa1               : num  349 223 518 508 509 ...
##  $ ppsa2               : num  679 546 2066 2013 1999 ...
##  $ ppsa3               : num  31 42.3 64 61.7 61.6 ...
##  $ pnsa1               : num  51.1 141.8 202 225.4 158.8 ...
##  $ pnsa2               : num  -99.3 -346.9 -805.9 -894 -623.3 ...
##  $ pnsa3               : num  -10.5 -44 -43.8 -42 -39.8 ...
##  $ fpsa1               : num  0.872 0.611 0.719 0.693 0.762 ...
##  $ fpsa2               : num  1.7 1.5 2.87 2.75 2.99 ...
##  $ fpsa3               : num  0.0774 0.1159 0.0888 0.0842 0.0922 ...
##  $ fnsa1               : num  0.128 0.389 0.281 0.307 0.238 ...
##  $ fnsa2               : num  -0.248 -0.951 -1.12 -1.22 -0.933 ...
##  $ fnsa3               : num  -0.0262 -0.1207 -0.0608 -0.0573 -0.0596 ...
##  $ wpsa1               : num  139.7 81.4 372.7 372.1 340.1 ...
##  $ wpsa2               : num  272 199 1487 1476 1335 ...
##  $ wpsa3               : num  12.4 15.4 46 45.2 41.1 ...
##  $ wnsa1               : num  20.4 51.8 145.4 165.3 106 ...
##  $ wnsa2               : num  -39.8 -126.6 -580.1 -655.3 -416.3 ...
##   [list output truncated]
```

Before proceeding the data set must be partitioned into a training and a test set.

```
set.seed(42)
trainIndex <- createDataPartition(y=logBBB, times=1, p=0.8, list=F)
descrTrain <- bbbDescr[trainIndex,]
concRatioTrain <- logBBB[trainIndex]
descrTest <- bbbDescr[-trainIndex,]
concRatioTest <- logBBB[-trainIndex]
```

Analyse the correlation between features

```
cm <- cor(descrTrain)
corrplot(cm, order="hclust", tl.pos="n")
```



The number of variables exhibiting a pair-wise correlation coefficient above 0.75 can be determined:

```
highCorr <- findCorrelation(cm, cutoff=0.75)
length(highCorr)
```

```
## [1] 65
```

```
anyNA(descrTrain)
```

```
## [1] FALSE
```

```
nearZeroVar(descrTrain)
```

```
## [1]  3 16 17 22 25 50 60
```

We know there are issues with scaling, and the presence of highly correlated predictors and near zero variance predictors. These problems are resolved by pre-processing. First we define the procesing steps.

```
transformations <- preProcess(descrTrain,
                              method=c("center", "scale", "corr", "nzv"),
                              cutoff=0.75)

descrTrain <- predict(transformations, descrTrain)
```

k nearest neighbours

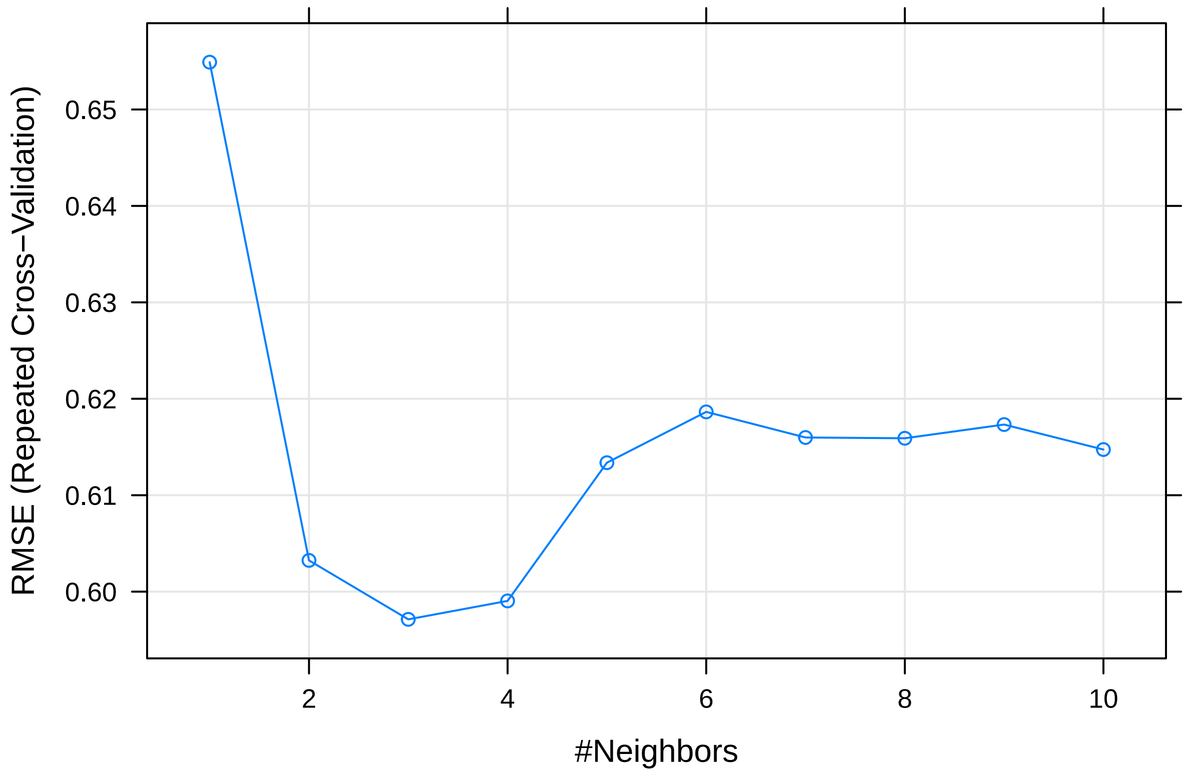
Search for the optimum number of clusters

```
set.seed(42)
seeds <- vector(mode = "list", length = 26)
```

```r
for(i in 1:25) seeds[[i]] <- sample.int(1000, 10)
seeds[[26]] <- sample.int(1000,1)

knnTune <- train(descrTrain,
                 concRatioTrain,
                 method="knn",
                 tuneGrid = data.frame(.k=1:10),
                 trControl = trainControl(method="repeatedcv",
                                          number = 5,
                                          repeats = 5,
                                          seeds=seeds,
                                          preProcOptions=list(cutoff=0.75))
                 )

knnTune
```

```
## k-Nearest Neighbors
##
## 168 samples
## 127 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 136, 133, 134, 134, 135, 135, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    1  0.6549043  0.4108920  0.4766928
##    2  0.6032405  0.4528603  0.4576061
##    3  0.5971264  0.4502719  0.4540638
##    4  0.5990394  0.4427879  0.4548592
##    5  0.6133729  0.4188855  0.4696289
##    6  0.6186412  0.4098024  0.4738693
##    7  0.6159861  0.4130907  0.4738798
##    8  0.6159065  0.4137798  0.4716903
##    9  0.6173266  0.4091199  0.4724567
##   10  0.6147349  0.4169201  0.4691612
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 3.
```

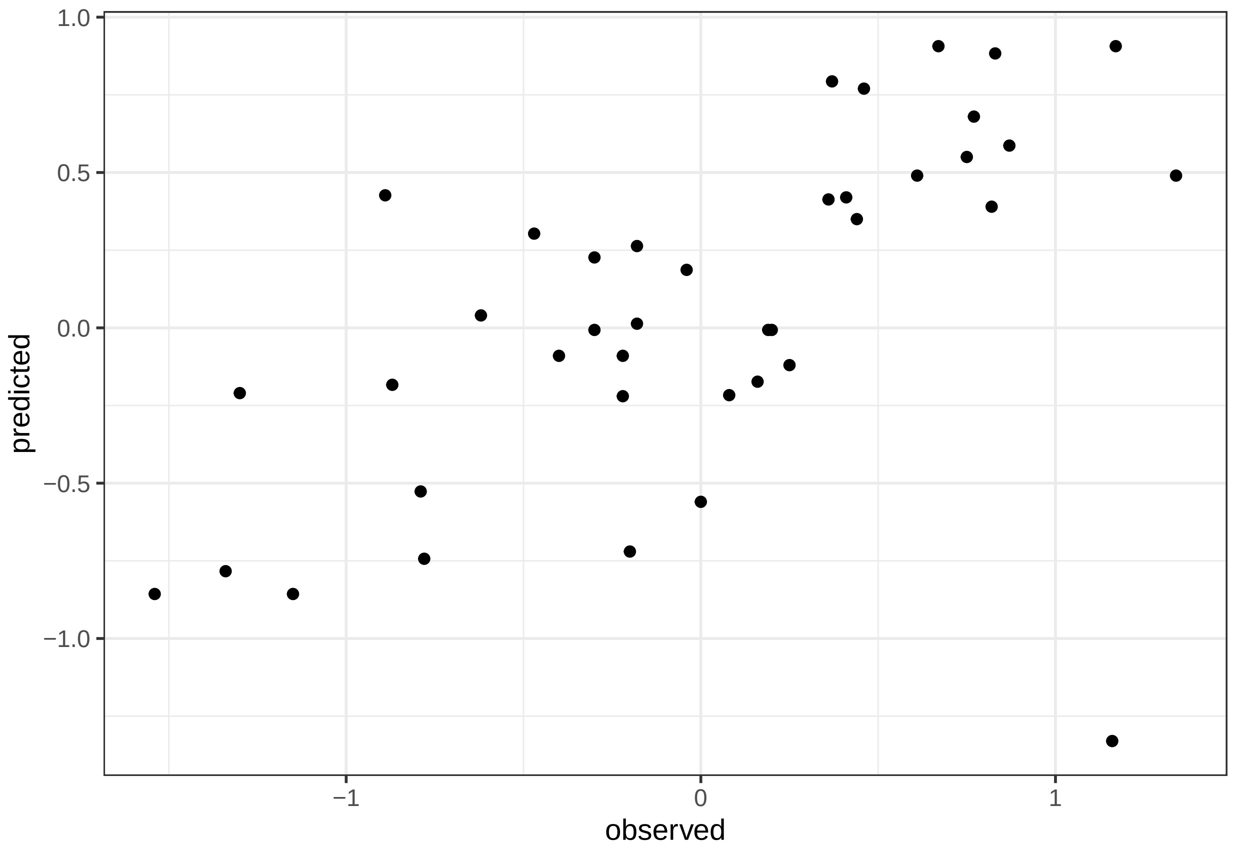
```r
plot(knnTune)
```



Use model to make predictions

Before attempting to predict the blood/brain concentration ratios of the test samples, the descriptors in the test set must be transformed using the same pre-processing procedure that was applied to the descriptors in the training set.

```
descrTest <- predict(transformations, descrTest)

test_pred <- predict(knnTune, descrTest)

#Prediction performance can be visualized in a scatterplot.
qplot(concRatioTest, test_pred) +
  xlab("observed") +
  ylab("predicted") +
  theme_bw()
```



```
cor(concRatioTest, test_pred)
```

```
## [1] 0.5719185
```

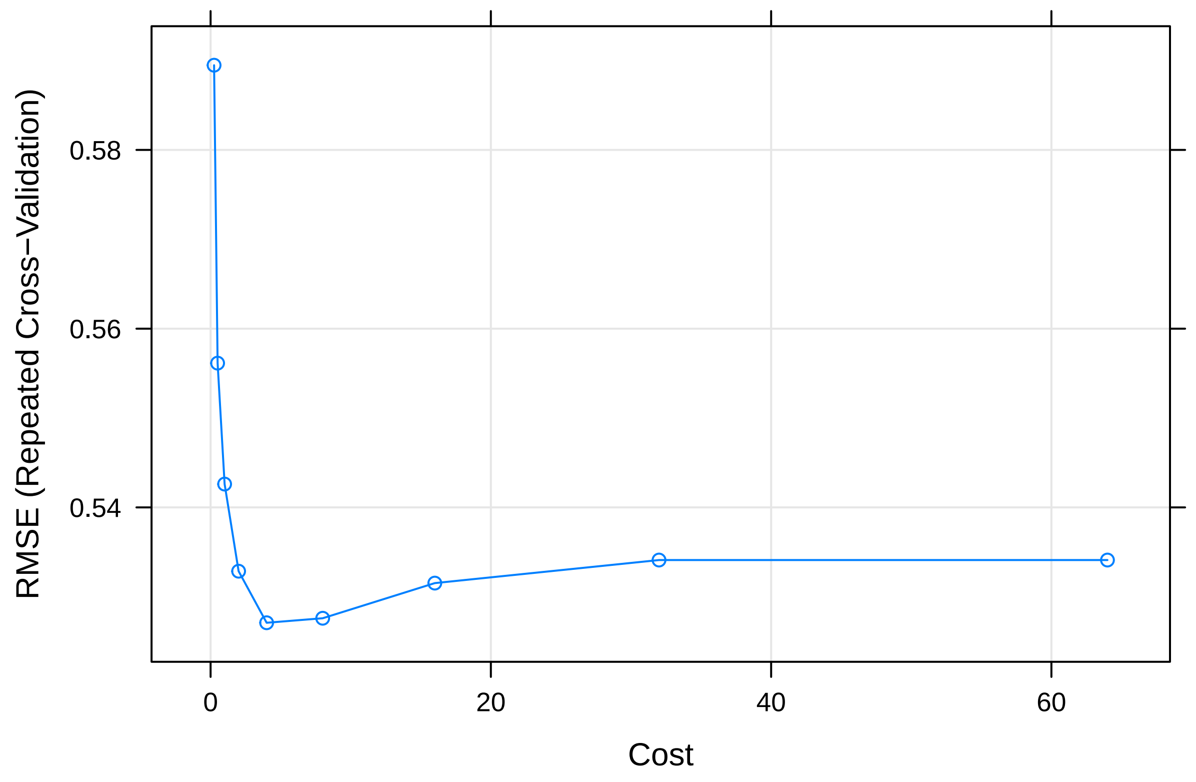
SVMs

```
data(BloodBrain)

set.seed(42)
trainIndex <- createDataPartition(y=logBBB, times=1, p=0.8, list=F)
descrTrain <- bbbDescr[trainIndex,]
concRatioTrain <- logBBB[trainIndex]
descrTest <- bbbDescr[-trainIndex,]
concRatioTest <- logBBB[-trainIndex]

transformations <- preProcess(descrTrain,
                              method=c("center", "scale", "corr", "nzv"),
                              cutoff=0.75)
descrTrain <- predict(transformations, descrTrain)

set.seed(42)
seeds <- vector(mode = "list", length = 26)
for(i in 1:25) seeds[[i]] <- sample.int(1000, 50)
seeds[[26]] <- sample.int(1000,1)
```

```
svmTune2 <- train(descrTrain,
                  concRatioTrain,
                  method ="svmRadial",
                  tuneLength = 9,
                  trControl = trainControl(method="repeatedcv",
                                           number = 5,
                                           repeats = 5,
                                           seeds=seeds,
                                           preProcOptions=list(cutoff=0.75)
                                           )
)

svmTune2
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 168 samples
## 127 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 136, 134, 134, 134, 134, 134, ...
## Resampling results across tuning parameters:
##
##   C      RMSE       Rsquared   MAE
##    0.25  0.5894761  0.5150058  0.4455836
##    0.50  0.5561418  0.5437051  0.4193837
##    1.00  0.5426114  0.5526850  0.4106434
##    2.00  0.5328636  0.5653459  0.4041801
##    4.00  0.5270938  0.5747292  0.3962617
##    8.00  0.5275978  0.5744446  0.3920354
##   16.00  0.5315302  0.5687043  0.3926010
##   32.00  0.5341106  0.5650992  0.3941705
##   64.00  0.5341106  0.5650992  0.3941705
##
## Tuning parameter 'sigma' was held constant at a value of 0.005208852
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.005208852 and C = 4.
```

```
plot(svmTune2)
```

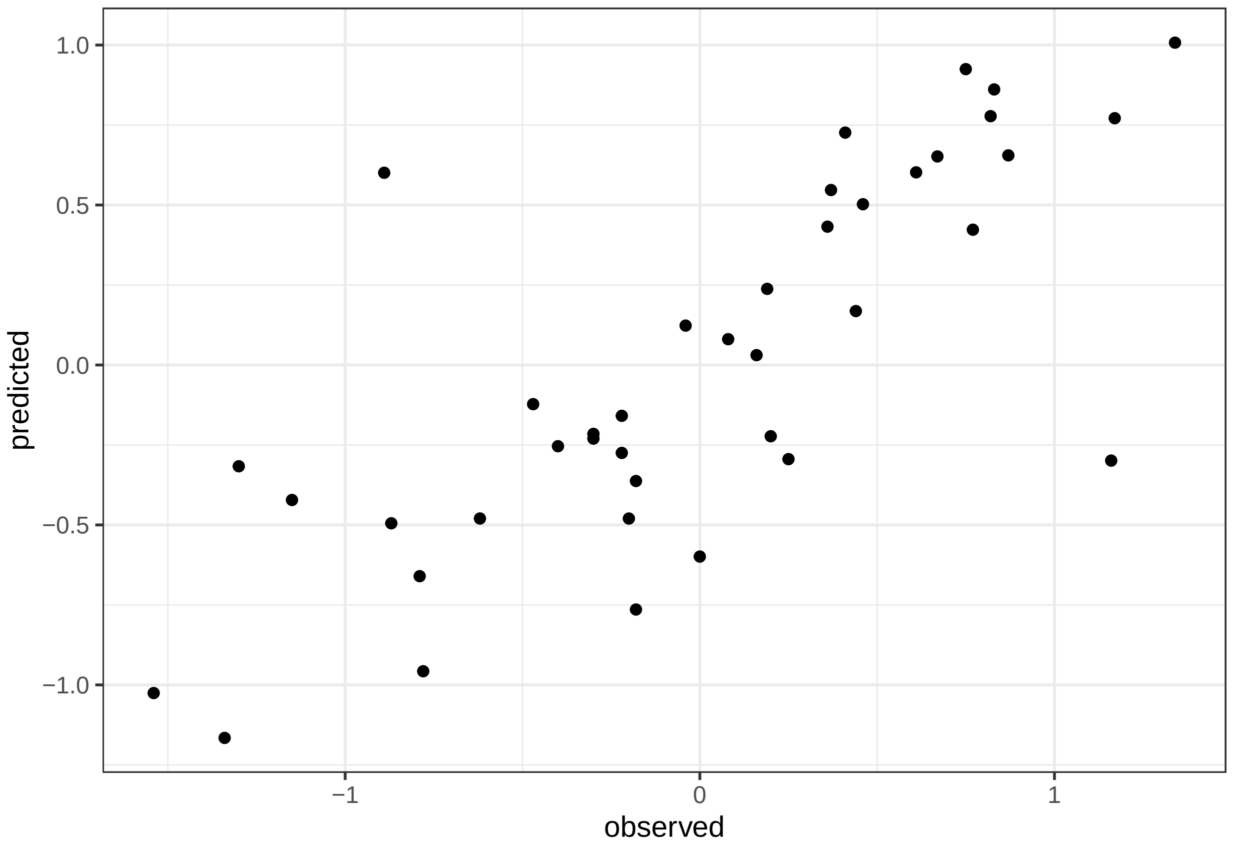


```
svmTune3 <- train(descrTrain,
                  concRatioTrain,
                  method ="svmLinear",
                  tuneLength = 9,
                  trControl = trainControl(method="repeatedcv",
                                           number = 5,
                                           repeats = 5,
                                           seeds=seeds,
                                           preProcOptions=list(cutoff=0.75)
                                           )
)

svmTune3
```

```
## Support Vector Machines with Linear Kernel
##
## 168 samples
## 127 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 134, 134, 135, 134, 135, 133, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.7185052  0.3945419  0.5384248
##
## Tuning parameter 'C' was held constant at a value of 1
```

Use model to predict outcomes, after first pre-processing the test set.

```r
descrTest <- predict(transformations, descrTest)
test_pred <- predict(svmTune2, descrTest)

qplot(concRatioTest, test_pred) +
  xlab("observed") +
  ylab("predicted") +
  theme_bw()
```



```r
cor(concRatioTest, test_pred)
```

```
## [1] 0.7678866
```