Machine Learning

Ravi Kumar Tiwari

14 June 2016

Introduction

1. Definition: It is a method of teaching computers to make predictions based on data
2. Types of machine learning:
    - Supervised learning: Learning from data in which output values are known
    - Unsupervised learning: Learning from data in which output values are unknown
3. Machine Learning Applications:
    - Prediction: Fuel Consumption of automobile based on their weight, House Prices based on locality, size
    - Forecasting: Linear Regression
    - Classification: Predicting flower species based on sepal and petal measurement Decision Tree, SVM, Logistic Regression
    - Clustering: Finding similar species of flowers based on their sepal and petal measurements, K-means, Hierarchical

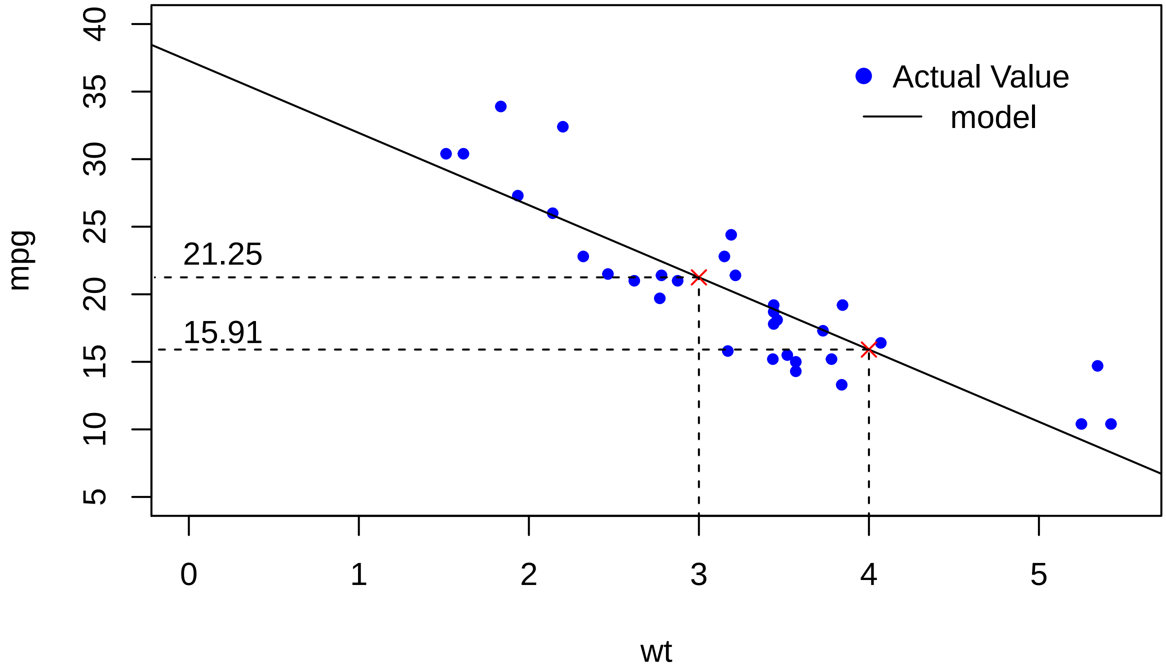
Supervised Learning

1. Linear Regression

This method involves finding a straight line (y = ax + b) that best describes the relationship between the dependent and the independent variables. This best fit line is then used to predict the value of the dependent variable for any given values of the independent variables.

1.1 Example

A linear model that captures the relationship between mpg (miles per gallon) and wt (weight of the car) in the mtcars dataset



1.2 R Codes

```
## Build the linear model
lmModel <- lm(mpg ~ wt, data = mtcars)

## Use the linear model to make prediction
predValue <- predict(lmModel, data.frame(wt = 3))
predValue <- predict(lmModel, data.frame(wt = c(3,4)))
predValue <- predict(lmModel, data.frame(wt = mtcars$wt))
```

```
## Access the model parameters
coef(lmModel)
```

1.3 Interpretation of the model parameter

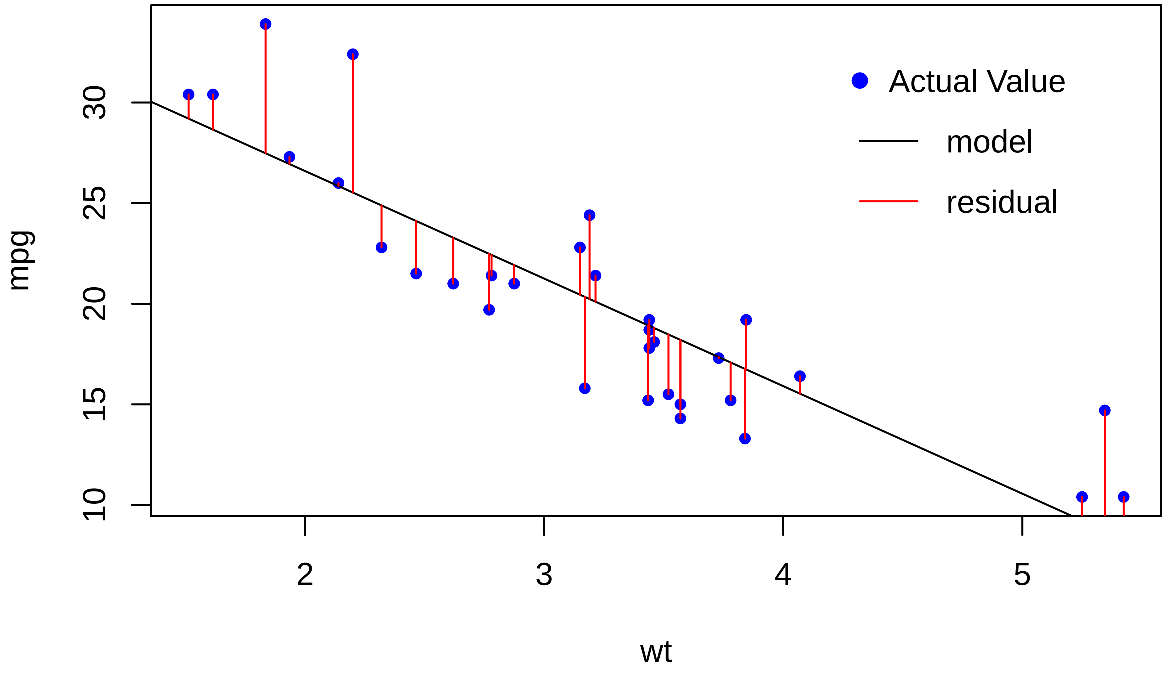
```
coef(lmModel)
```

```
## (Intercept)          wt
##   37.285126   -5.344472
```

The intercept is the model prediction for the case when the independent variable is 0. The slope is the change in the dependent variable when the independent variable changes by 1 unit

1.4 Model Assessment

1. Visual Inspection



2. R-squared value

```
sumModel <- summary(lmModel)
sumModel$r.squared
```

```
## [1] 0.7528328
```

1.5 Extension of linear model

```
## More than one predictors
lmModel2 <- lm(mpg ~ wt + hp + disp, data = mtcars) # wt, hp, and disp will be used as
# predictor
lmModel3  <- lm(mpg~ ., data = mtcars)   # All the variable will be used

## subset selection:
## 1) Identify the best model that contains a given number of predictors
## 2) Identify the overall best model

library(leaps)  # subset selection library
fwdSelection <- regsubsets(mpg ~ ., data = mtcars, method = "forward")
sumFwdSel <- summary(fwdSelection)
sumFwdSel$outmat  # 1) Shows the predictors to be included when their numbers are fixed
which.max(sumFwdSel$adjr2) # 2) overall best model has the highest adjusted
# r-squared value
```

1.6 Forward model selection: Output

1. Predictors to be included in the model when their numbers are fixed

```
sumFwdSel$outmat
```

```
##          cyl disp hp  drat wt  qsec vs  am  gear carb
## 1  ( 1 ) " " " "  " " " "  "*" " "  " " " " " "  " " 
## 2  ( 1 ) "*" " "  " " " "  "*" " "  " " " " " "  " " 
## 3  ( 1 ) "*" " "  "*" " "  "*" " "  " " " " " "  " " 
## 4  ( 1 ) "*" " "  "*" " "  "*" " "  " " "*" " "  " " 
## 5  ( 1 ) "*" " "  "*" " "  "*" "*"  " " "*" " "  " " 
## 6  ( 1 ) "*" "*"  "*" " "  "*" "*"  " " "*" " "  " " 
## 7  ( 1 ) "*" "*"  "*" "*"  "*" "*"  " " "*" " "  " " 
## 8  ( 1 ) "*" "*"  "*" "*"  "*" "*"  " " "*" "*"  " " 
```

2. Overall best model

```
n <- which.max(sumFwdSel$adjr2)
coef(fwdSelection, n)
```

```
## (Intercept)         cyl        disp          hp          wt        qsec 
## 20.05169952 -0.50206577  0.01396099 -0.01956054 -3.99773180  0.81017782 
##          am 
##  2.94074955 
```

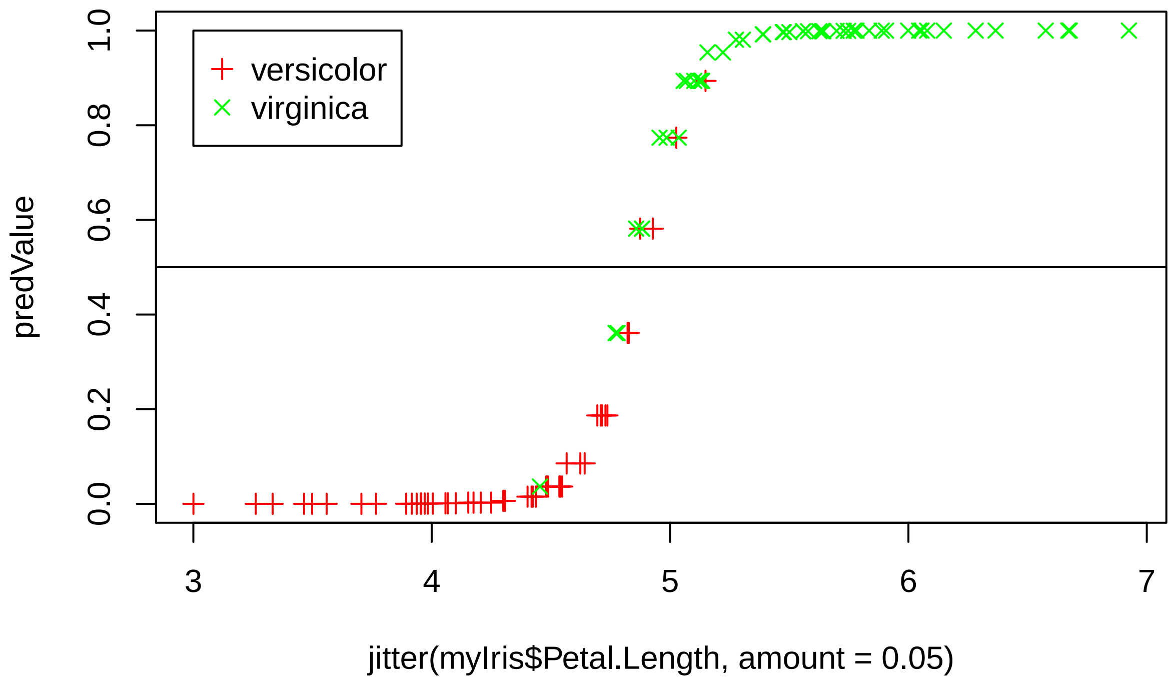
1.7 Challenge

Use backward selection model to find the best model for mpg

2. Logistic Regression

Fit the predictor values to a function whose value lies between 0 and 1. Choose a cut-off value to separates the function output values in two regions corresponding to two classes. A new observation class is decided by the region in which the function values corresponding to this observation lies.

Example



Codes

```
inSetosa <- iris$Species == "setosa"
myIris <- iris[!inSetosa,]
myIris$Species <- factor(myIris$Species, levels = c("versicolor", "virginica"))
glmModel <- glm(Species ~ Petal.Length, data = myIris, family = binomial(link="logit"))
predValue <- predict(glmModel, myIris, type = "response")
```

Model Assessment

```
prediction <- ifelse(predValue > 0.5, "virginica", "versicolor")
table(prediction, myIris$Species)
```

```
##
## prediction   versicolor virginica
##   versicolor         46         3
##   virginica           4        47
```

3. Tree based algorithm

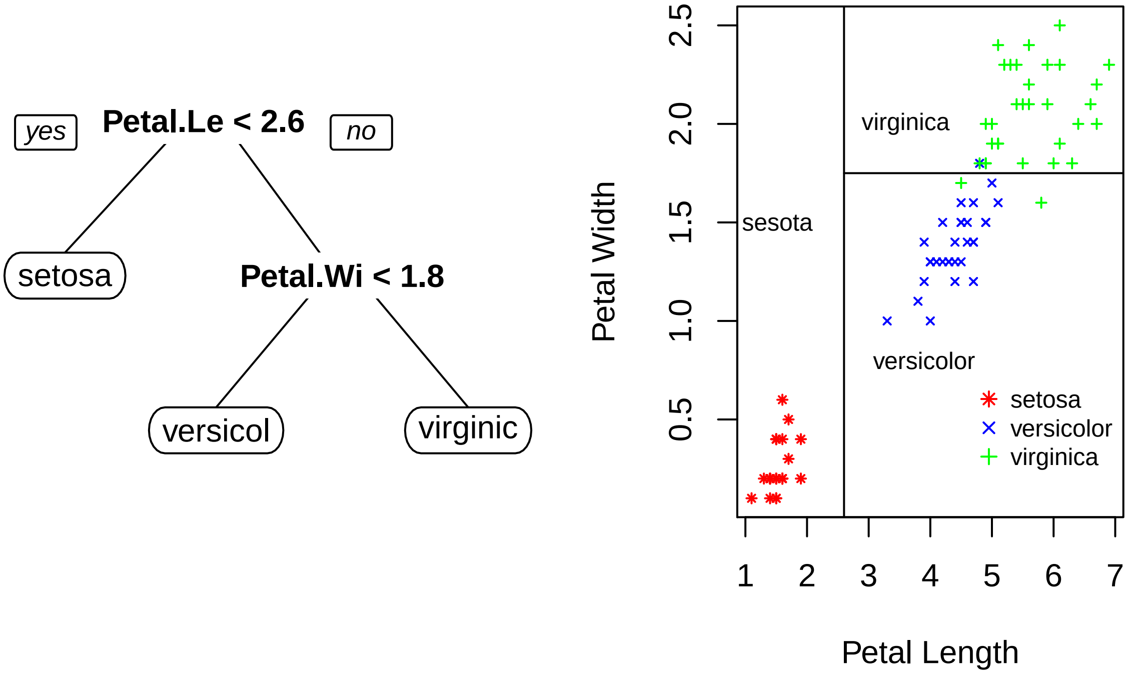
3.1 Decision Tree

Working principle Divide the data set into several small regions such that the response variables are (nearly) homogeneous in those regions. The predictd value of a new observation is the most dominant class of the region to which the observation belongs.

Example Find the decision rule to predict the species of iris dataset based on Sepal.Length, Sepal.Width, Petal.Length, and Petal.Width

```
iris[c(1,100,150),]
```

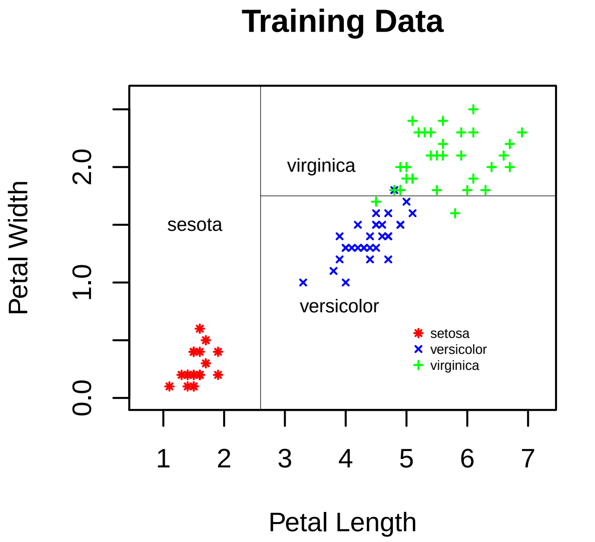
```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 100          5.7         2.8          4.1         1.3 versicolor
## 150          5.9         3.0          5.1         1.8  virginica
```

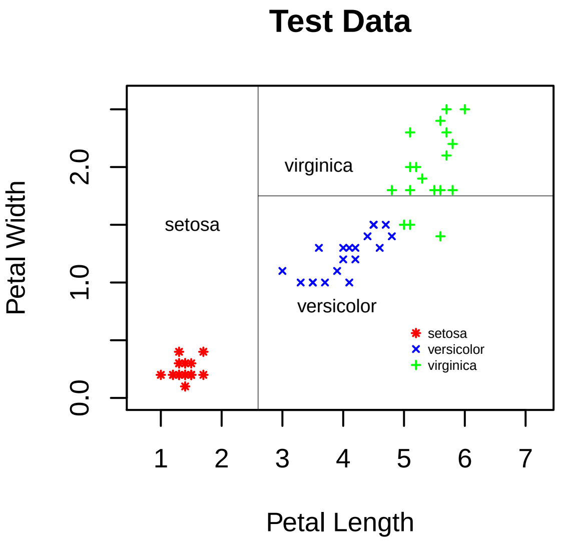


```
## Load the required libraries
library(rpart)
library(rpart.plot)  # For decision tree visualization
```

```r
## create the data partition
set.seed(1)
inTrain <- sample(c(TRUE, FALSE), size = nrow(iris), replace = TRUE, prob = c(0.6,0.4))
trainData <- iris[inTrain,]
testData <- iris[!inTrain,1:4]
testClass <- iris[!inTrain,5]

## Create the tree model
treeModel <- rpart(Species ~ ., data = trainData)

## Use the tree model to predict the class of the test data
predTrainClass <- predict(treeModel, newdata = trainData, type = "class")
predTestClass <- predict(treeModel, newdata = testData, type = "class")

## Find out the performance of the decision tree
table(predTrainClass, trainData$Species)  # Confusion Matrix
mean(predTrainClass == trainData$Species) # Prediction Accuracy

table(predTestClass, testClass)           # Confusion Matrix
mean(predTestClass == testClass)          # Prediction Accuracy
```

Codes





	setosa	**versicolor**	**virginica**
setosa	27	0	0
versicolor	0	29	2
virginica	0	1	30

	setosa	**versicolor**	**virginica**
setosa	23	0	0
versicolor	0	20	3
virginica	0	0	15

Add some challenge

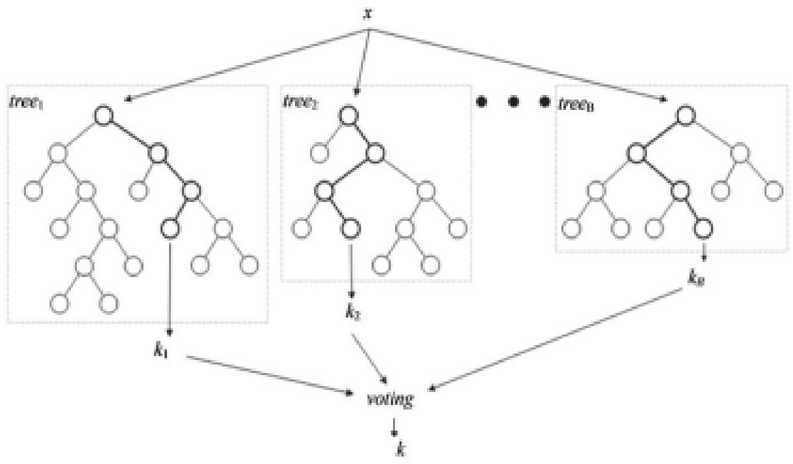
Problem with decision tree The decision is very easy to interpret. However, it has got low prediction accuracy. One way to enhance the prediction accuracy is to first build a lot of trees using the bootstrapped samples and use their mean as the prediction. In many cases, the trees formed in such a way are highly correlated as a result the averaging does not improve the result much. In order to decorrelate the trees, during the tree formation only some of the variables are considered when deciding which varibles to choose to split the tree on. In order to decorrelate the trees a random sample of m predictors (mtry) is chosen as split candidates from the full set of p predictors.

3.2 Random Forest

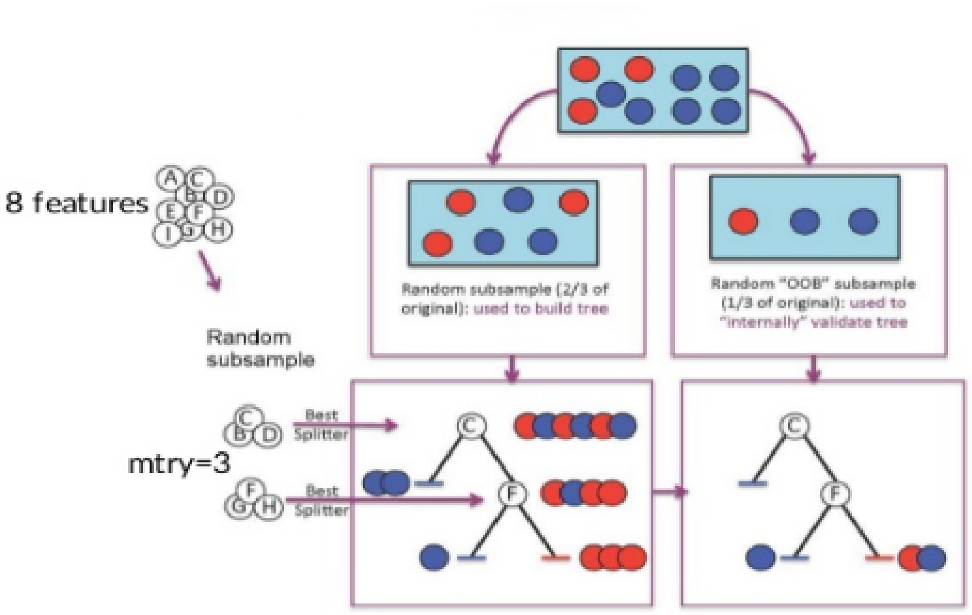
It fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging for prediction.

Important parameters of the random forest are: 1) ntree, 2) mtry

1. ntree



2. mtry


8 features
A C B D E F I G H
Random subsample
C B D
Best Splitter
mtry=3
F G H
Best Splitter
Random subsample (2/3 of original): used to build tree
Random "OOB" subsample (1/3 of original): used to "internally" validate tree
C
F
C
F

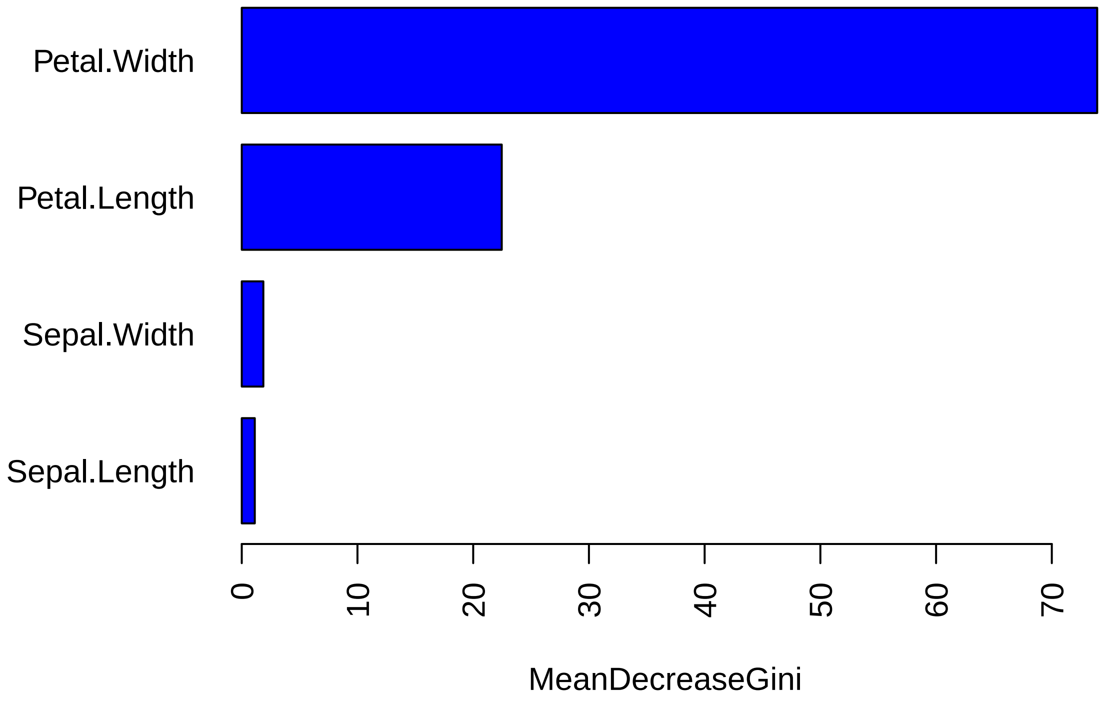
Codes

```
library(randomForest)
set.seed(1)
rfModel <- randomForest(Species ~ ., data=iris, mtry=4, ntree=20)
predClass <- predict(rfModel, newdata = iris)
table(predClass, iris$Species)
rfModel$importance
```

Prediction Accuracy

```
##
## predClass    setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         49         0
##   virginica       0          1        50
```

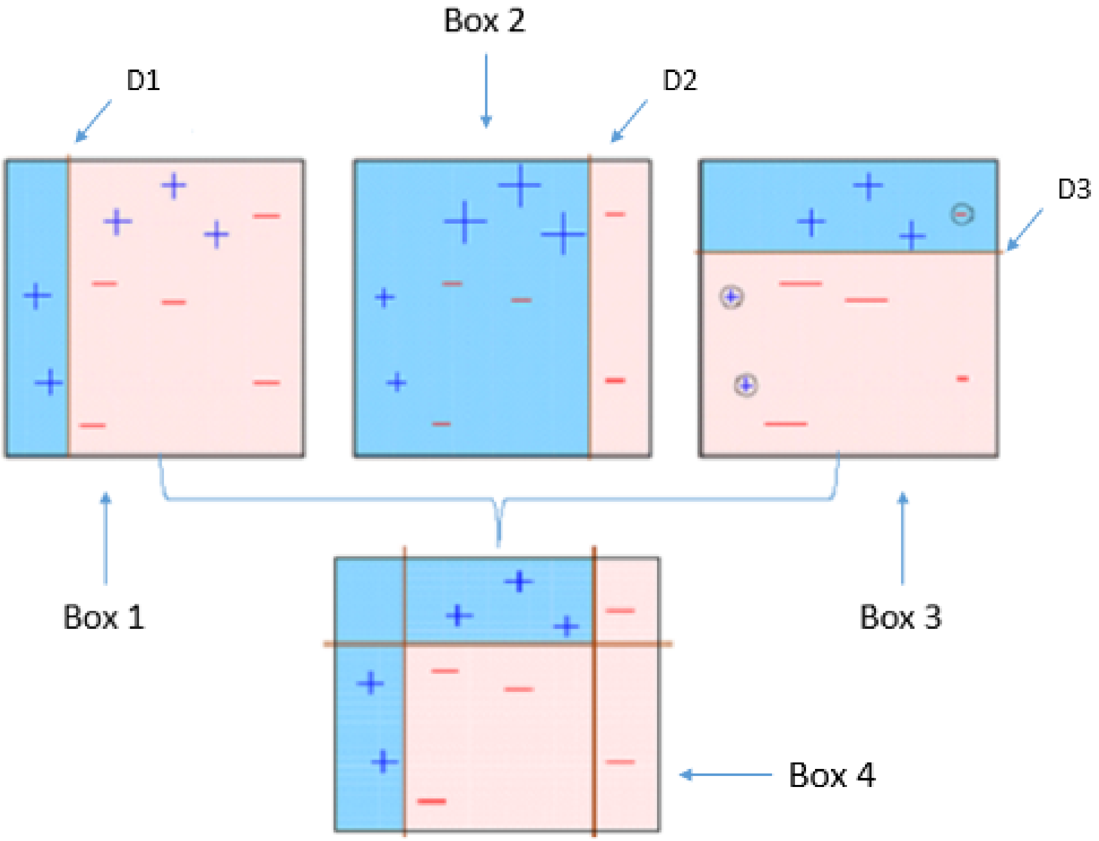
Variable Importance



3.3 Boosted Tree

Combines a number of weak classifiers using proper weight to form a strong classifier

Illustration The argument n.teees = 5000 indicates that we want 5000 trees, and the option interaction.depth = 4 limits the depth of each tree



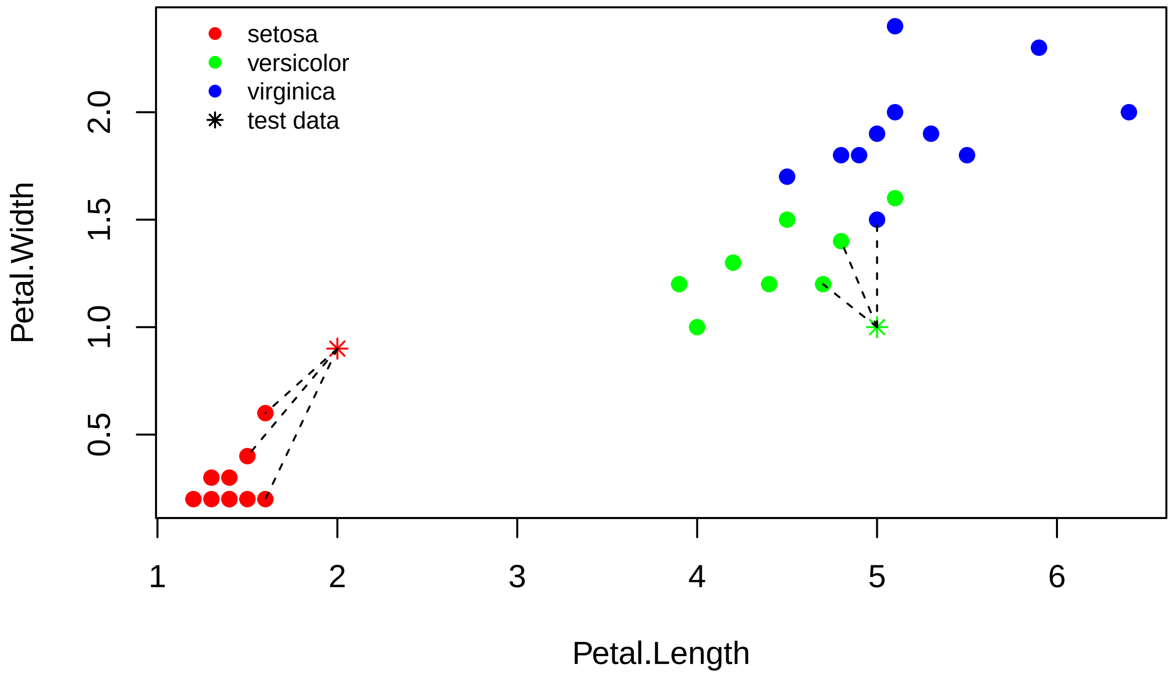
Codes

Assesment

4. KNN

It assumes that the members of a given class have similar characteristics. So, a given observation is assigned the class of its nearest neighbours (number of nearest neighbour to be decided by the user)

Example



codes

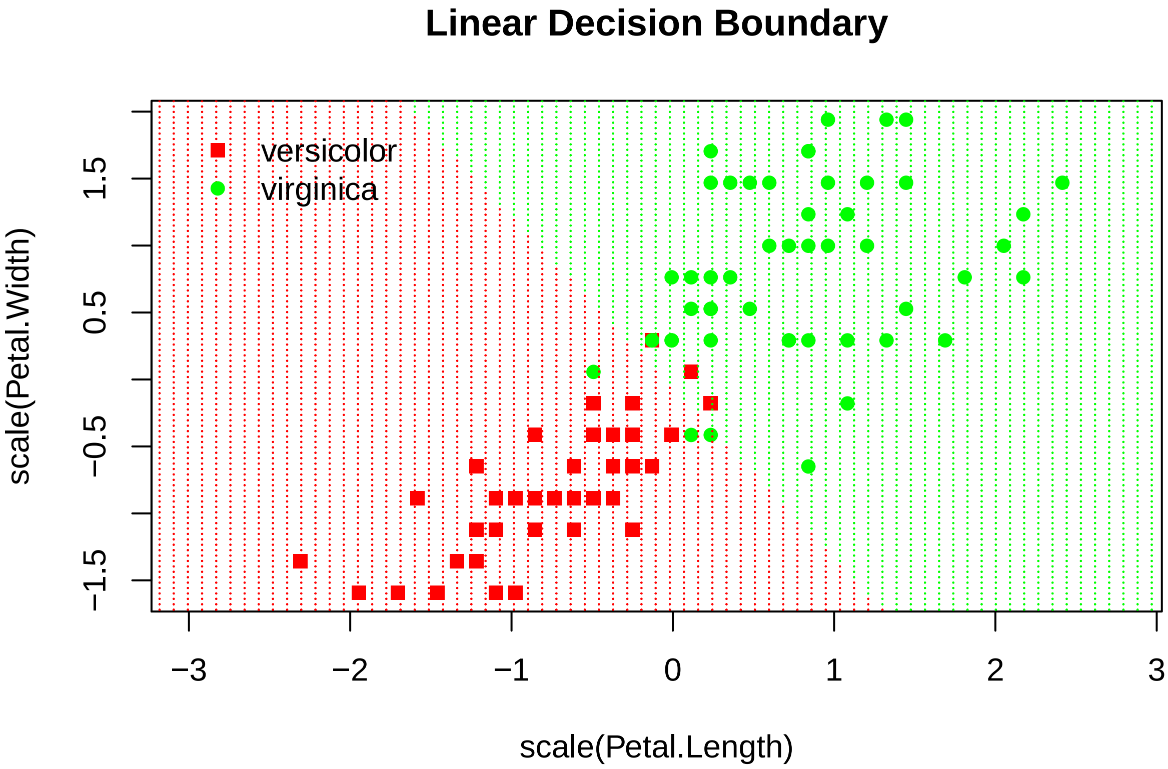
```
library(class)
myIris <- iris[,3:5]

set.seed(100)
inTrain <- sample(c(TRUE, FALSE), size = nrow(myIris), replace = TRUE, prob = c(0.2,0.8))
trainData <- myIris[inTrain,1:2]
trainClass <- myIris[inTrain,3]
testData <- myIris[!inTrain,1:2]
testClass <- myIris[!inTrain,3]

predClass <- knn(trainData, testData, cl = trainClass, k = 3)
table(predClass, testClass)
```

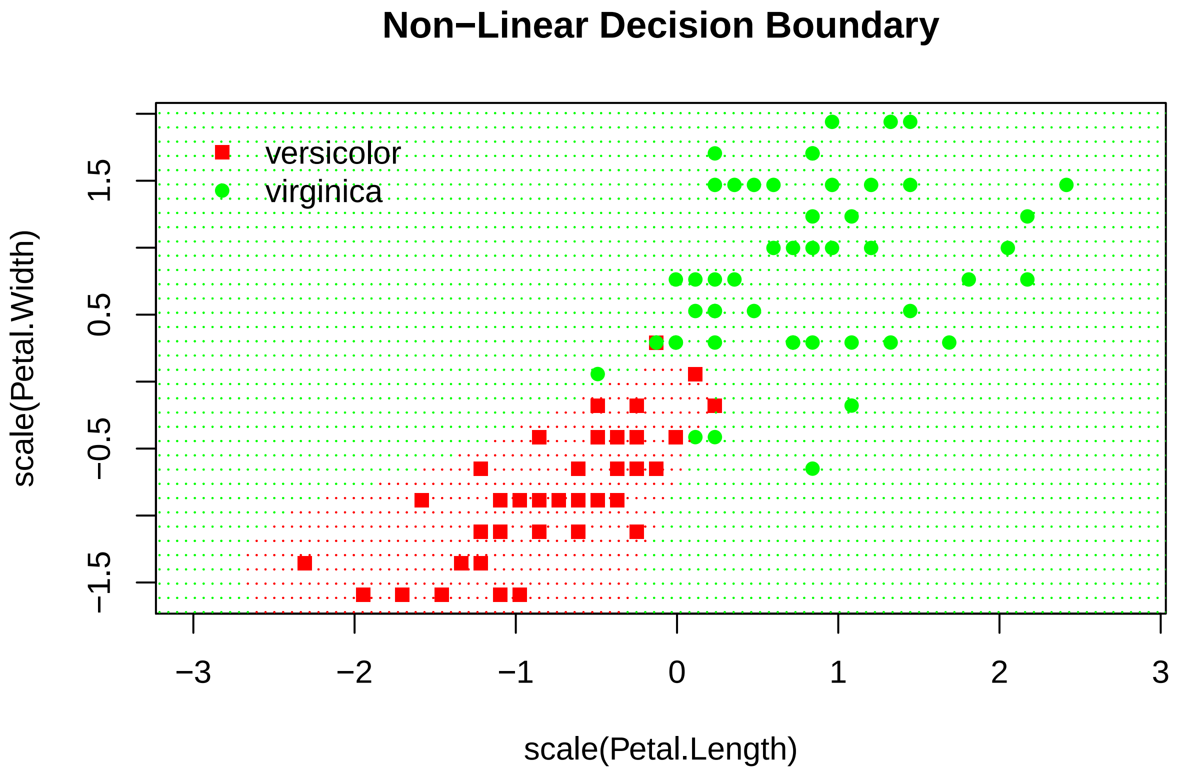
```
##               testClass
## predClass    setosa versicolor virginica
##   setosa         40          0         0
##   versicolor      0         40         1
##   virginica       0          2        38
```

5. SVM

It classifies a test observation depending on which side of a hyperplane it lies. The hyperplane is chosen to correctly separate most of the training observations into two classes.

Example





Codes

```
library(e1071)
inSetosa <- iris$Species == "setosa"
myIris <- iris[!inSetosa, c("Petal.Length", "Petal.Width", "Species")]
myIris$Species <- factor(myIris$Species, levels = c("versicolor", "virginica"))
svmModel <- svm(Species ~ ., data = myIris, kernal = "linear",
                scale = FALSE)
summary(svmModel)
prediction <- predict(svmModel, myIris[, 1:2])
table(prediction, myIris$Species)
```

Assessment

```
prediction <- predict(svmModel, myIris[, 1:2])
table(prediction, myIris$Species)
```

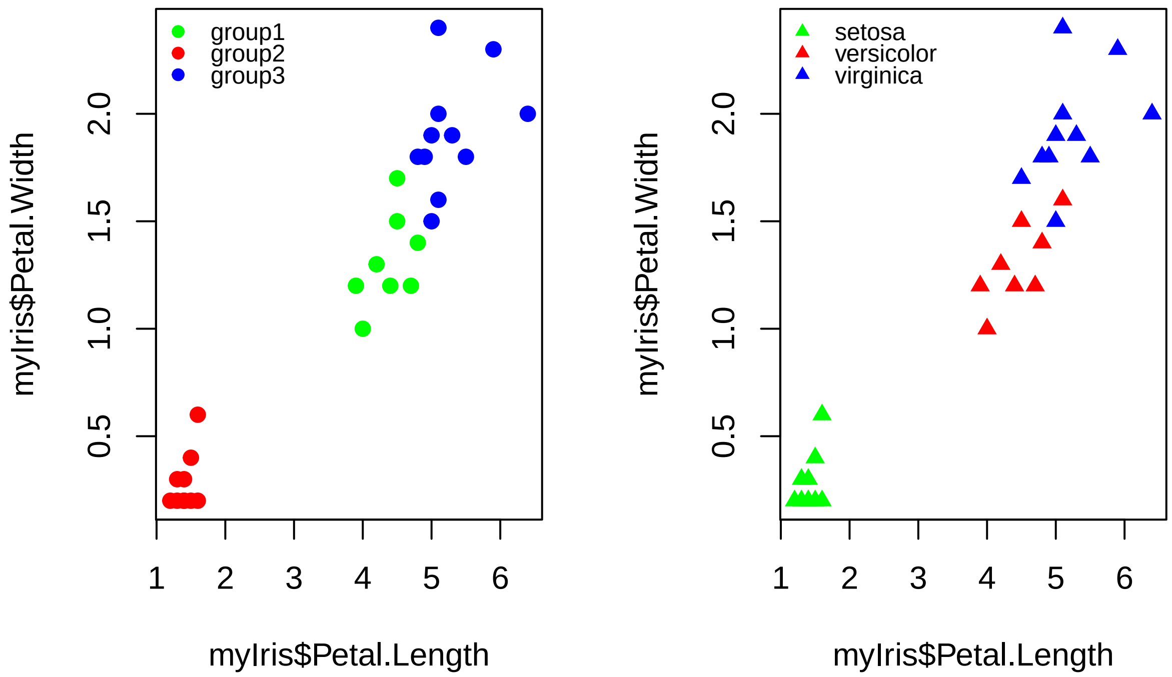
```
##
## prediction   versicolor virginica
##   versicolor         47         2
##   virginica           3        48
```

```
mean(prediction==myIris$Species)
```

```
## [1] 0.95
```

Unsupervised Learning

1. kmeans clustering

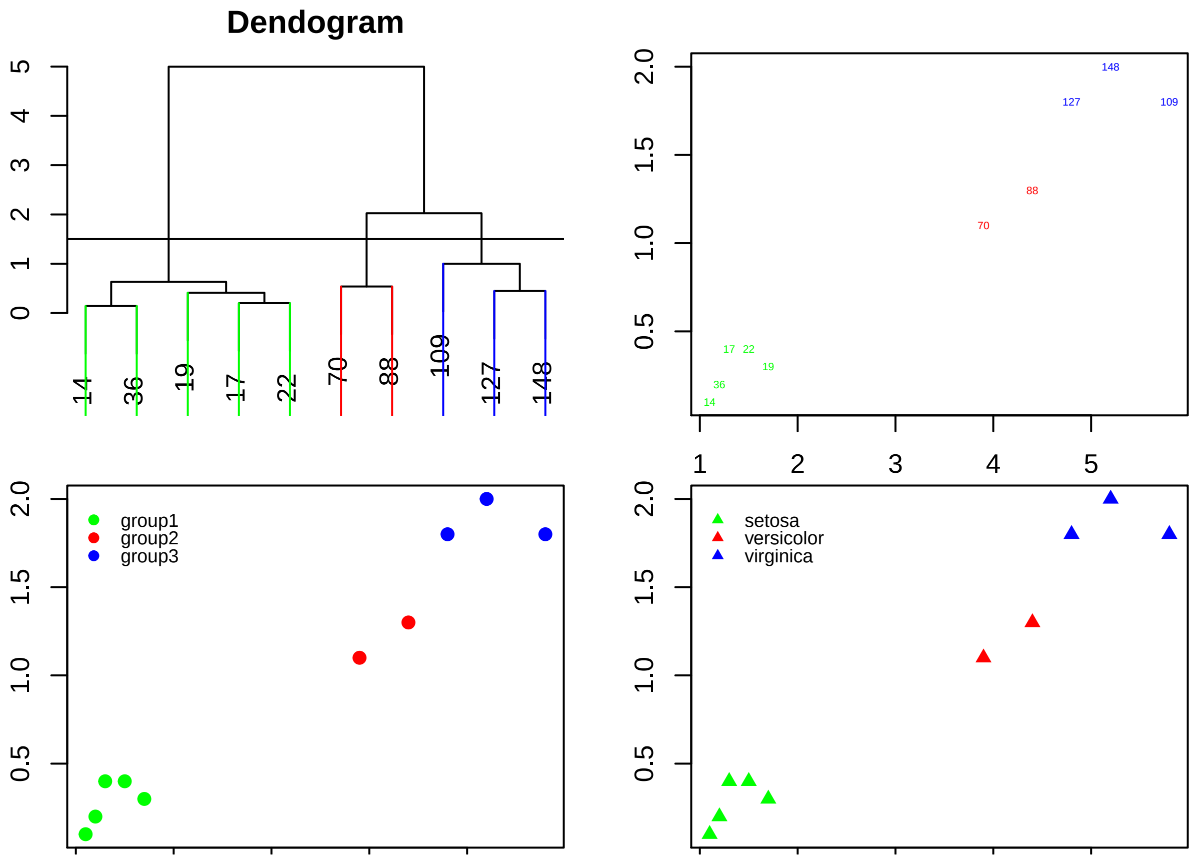


Codes

```
set.seed(100)
index <- sample(c(TRUE, FALSE), nrow(iris), p = c(0.2, 0.8), replace = TRUE)
myIris <- iris[index,3:4]
group <- iris$Species[index]
set.seed(100)
predGroup <- kmeans(myIris, centers = 3, nstart = 10)
predGroupC <- ifelse(predGroup$cluster==1, "setosa", ifelse(predGroup$cluster==2,
                                                    "versicolor", "virginnica"))
predGroupC <- factor(predGroupC)
table(predGroupC, group)
```

```
##               group
## predGroupC    setosa versicolor virginica
##   setosa           0          7         1
##   versicolor      10          0         0
##   virginnica       0          1        10
```

2. Hierarchichal Clustering



Codes

```
set.seed(4)
index <- sample(c(TRUE, FALSE), nrow(iris), p = c(0.05, 0.95), replace = TRUE)
myIris <- iris[index,3:4]
disM <- dist(myIris)
irisClust <- hclust(disM)
clusters <- cutree(irisClust, k = 3)
```

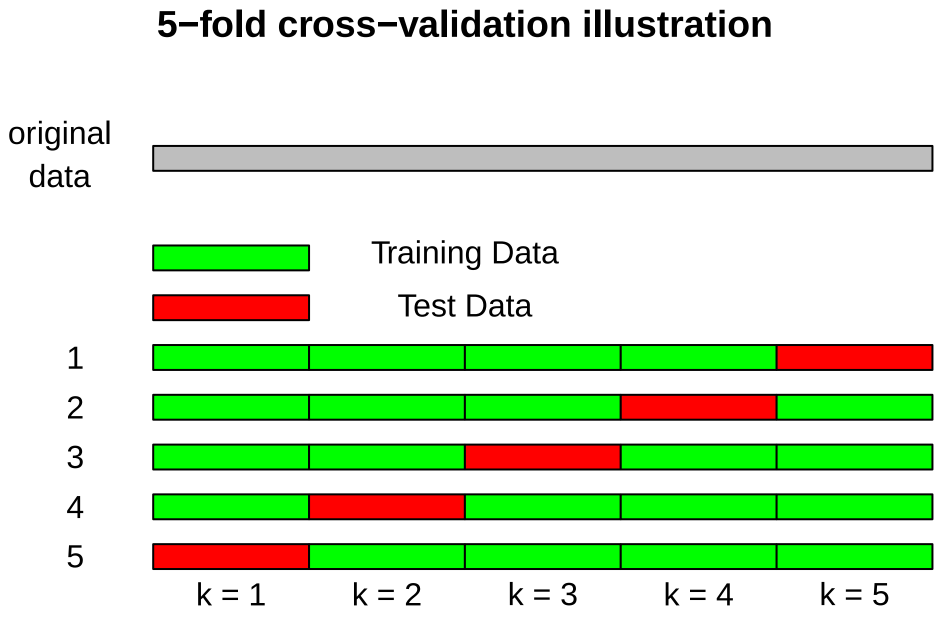
Resampling Methods

Resampling methods involve repeatedly drawing samples from the original data and refitting it to the model of interest. These methods are very useful in getting additional information about the model.

1. k-fold Cross-validation

This method involves randomly splitting dataset into k-folds of equal size. Out of k-fold, one group of observation is held-out and the remaining k-1 groups of observations are used to train the model.

This method is very useful to estimate 1. test-error associated with a given learning method in order to evaluate its performance (model assessment) 2. choose appropriate level of flexibility (model selection)

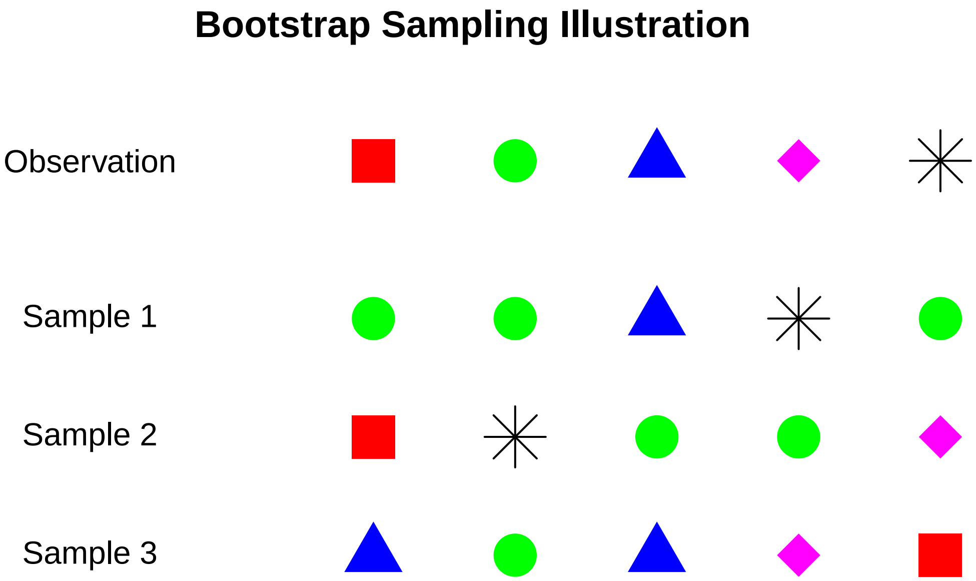


Leave-one-out cross-validation (LOOCV) is a special case of k-fold cross-validation where k = n, where n is the number of observations.

2. Bootstrap Sampling

This method involves repeatedly withdrawing samples from the original data set with replacement. The sample size of the withdrawn sample is kept the same as that of the original data.

The n trees in the random forest are fitted using n bootstrapped samples obtained from the original observation. Bootstrapped sampling is also used to measure the accuracy of the fitted parameters.


Bootstrap Sampling Illustration
Observation
Sample 1
Sample 2
Sample 3