

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

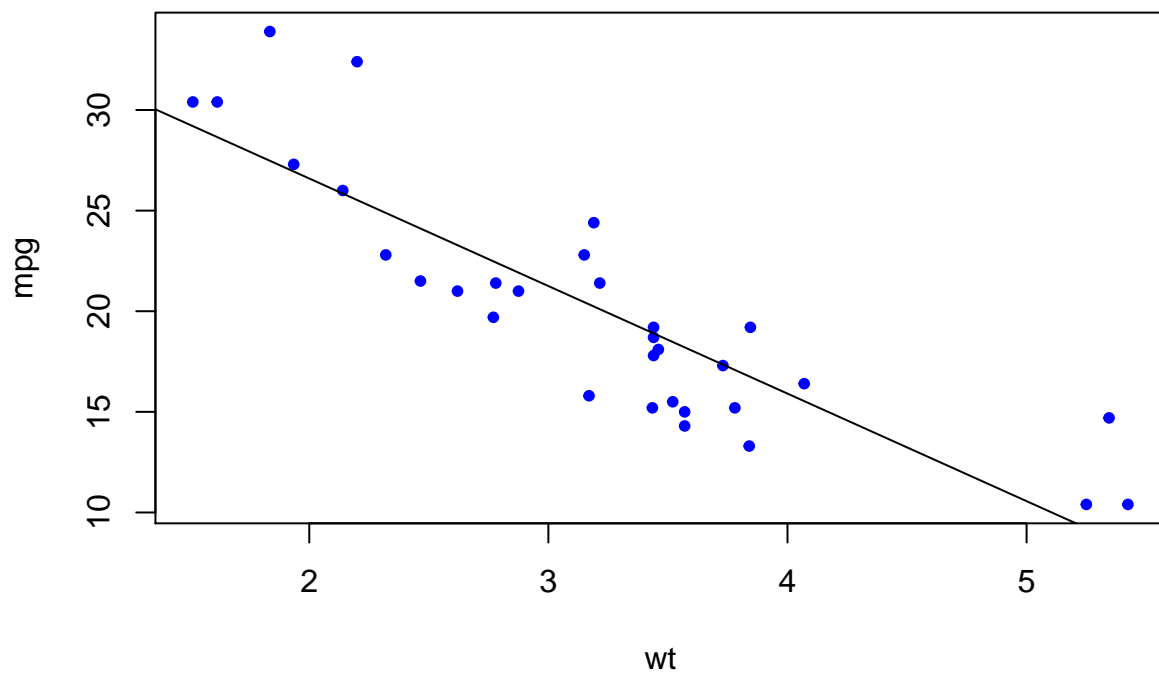
Linear Regression

Working principle

Find a straight line that best describes the relationship between the dependent and the independent variables. In order to obtain the predicted value of the dependent variable, plug in the the values of the independent variable in the equation of the line.

Example

Build a linear model to describe the relationship between mpg (miles per gallon) and wt (weight of the car) in the mtcars dataset



Codes

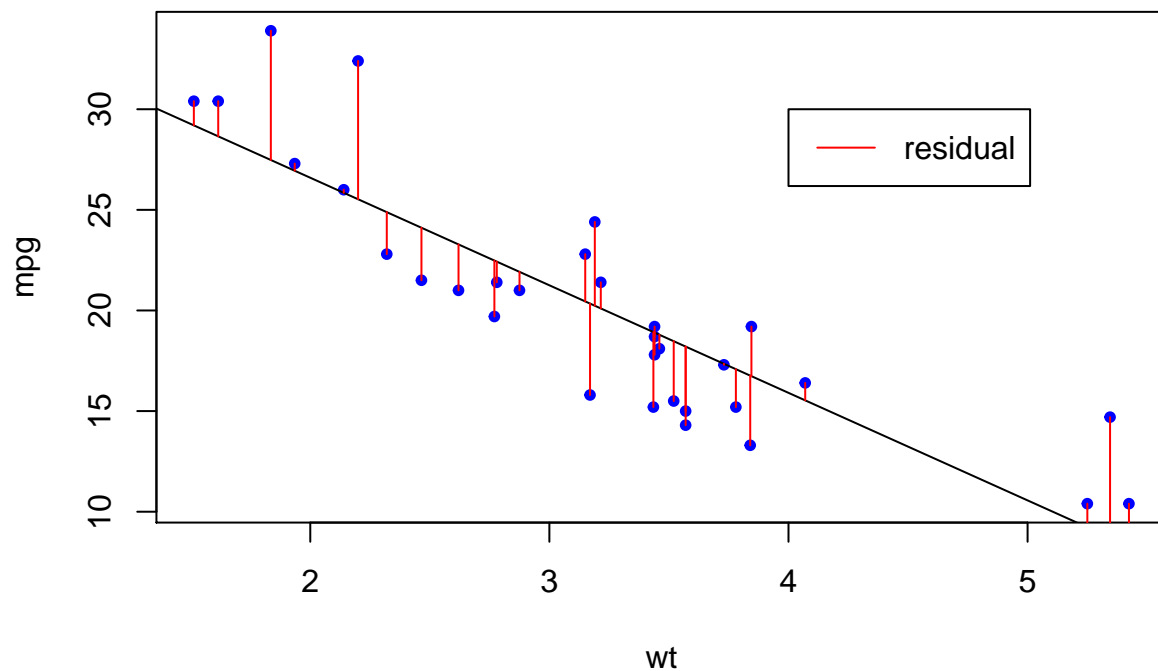
```
## Build the linear model object
lmModel <- lm(mpg ~ wt, data = mtcars)

## Obtain the model parameters
sumModel <- summary(lmModel)
sumModel$coefficients
```

```
## Prediction using the model
predValue <- predict(lmModel, data.frame(wt = c(3.5, 4.2)))
predValue <- predict(lmModel, data.frame(wt = mtcars$wt))
```

Model Assessment

1. Visual Inspection



2. R-squared value

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

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

3. F-statistics

```
sumModel$fstatistic
```

```
##      value      numdf      dendif
## 91.37533   1.00000  30.00000
```

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$outmat # 1) Included predictor in the Best Model when the number of predictors is fixed
which.max(sumFwdSel$adjr2) # 2) overall best model has the highest adjusted r-squared value
```

Output

1. Included predictors in the best model when the number of predictors 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
```

Challenge

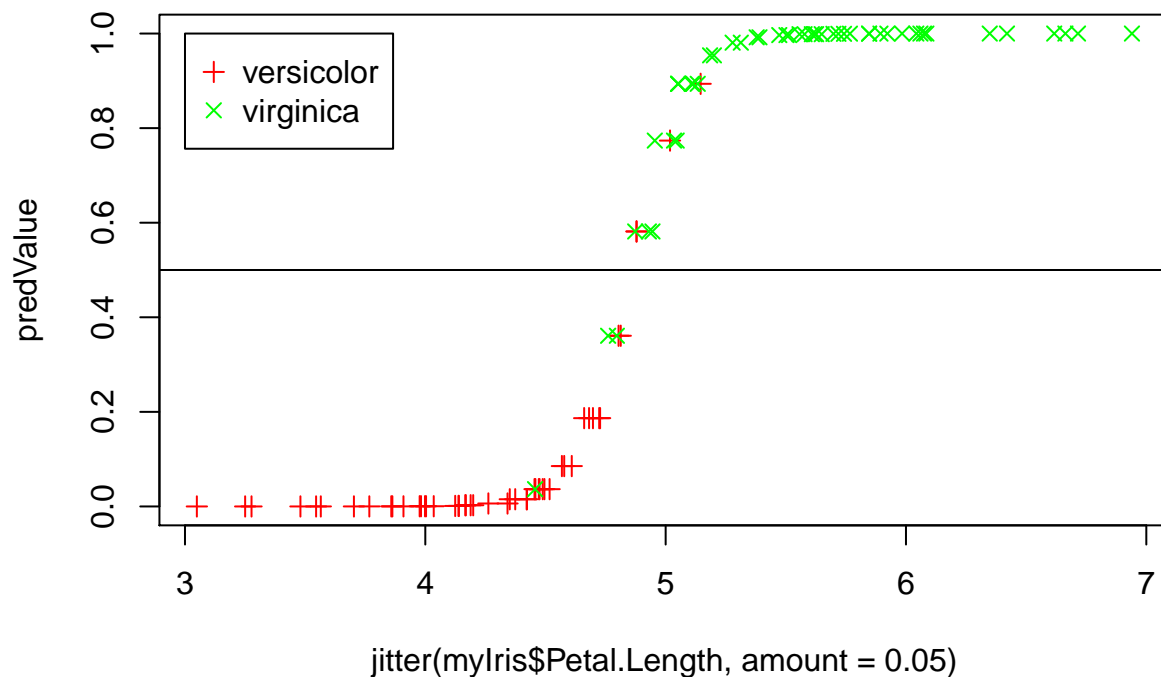
Use backward selection model to find the best model for mpg

Logistic Regression

Working Principle

Fit the predictor values to a function whose value lies between 0 and 1. Choose a cut-off value to separate 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
```

Tree based algorithm

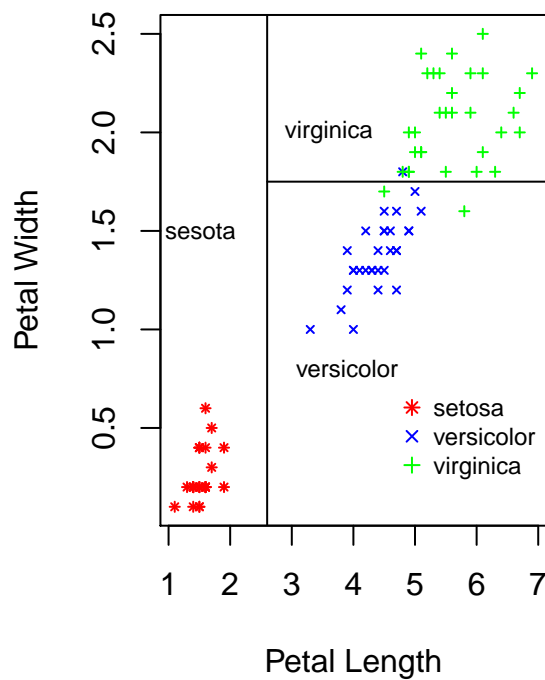
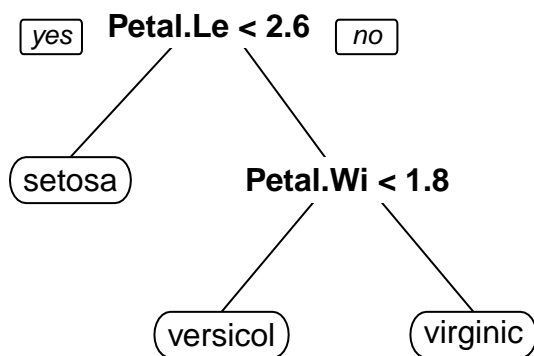
Decision Tree

Working principle Divide the data set into several small regions such that the response variables are (nearly) homogeneous in those regions. The predicted 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
```

```

## 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)

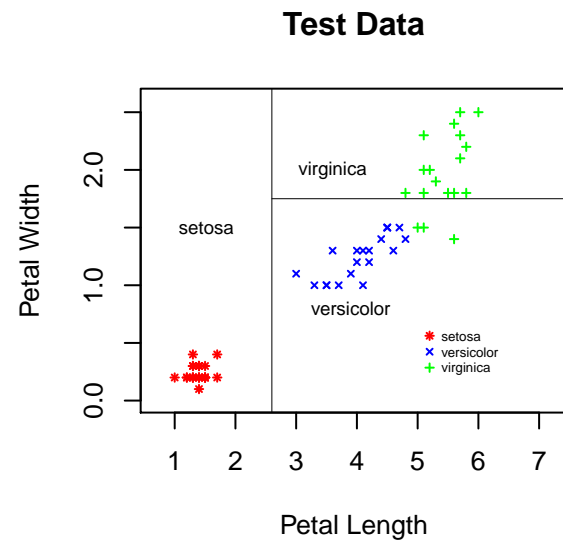
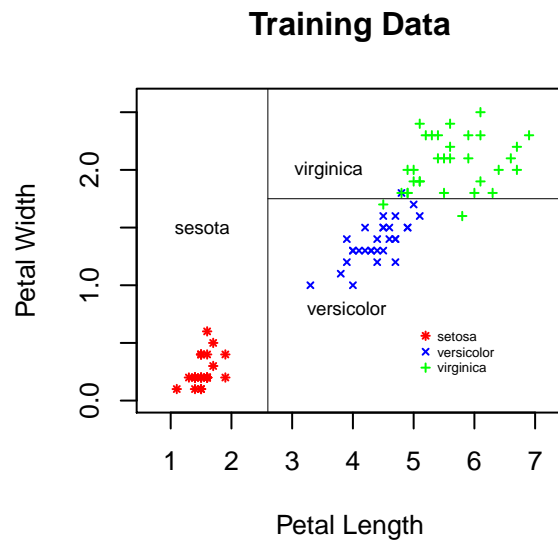
## Use the tree model to predict the class of the test data
predTrainClass <- predict(treeModel, newdata = trainData, type = "class")
predTestClass <- predict(treeModel, newdata = testData, type = "class")

## 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

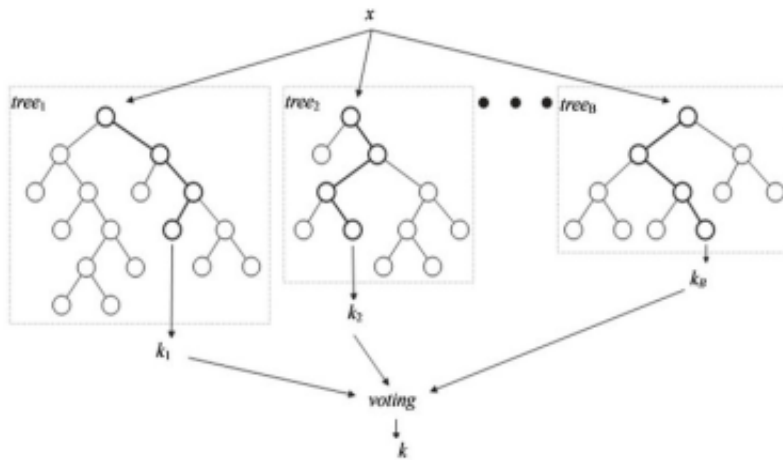
Problem with decision tree The decision is very easy to interpret. However, it has got low prediction accuracy. One way to enhance the prediction accuracy is to first build a lot of trees using the bootstrapped samples and use their mean as the prediction. In many cases, the trees formed in such a way 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 variables to choose to split the tree on. In order to decorrelate the trees a random sample of m predictors (m try) is chosen as split candidates from the full set of p predictors.

Random Forest

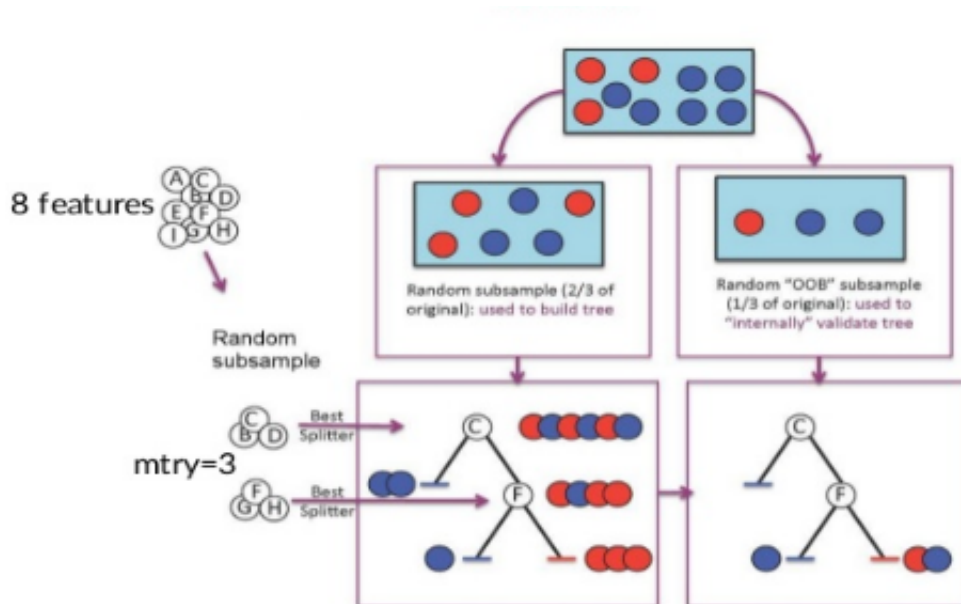
It fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging for prediction.

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

1. `ntree`



2. `mtry`



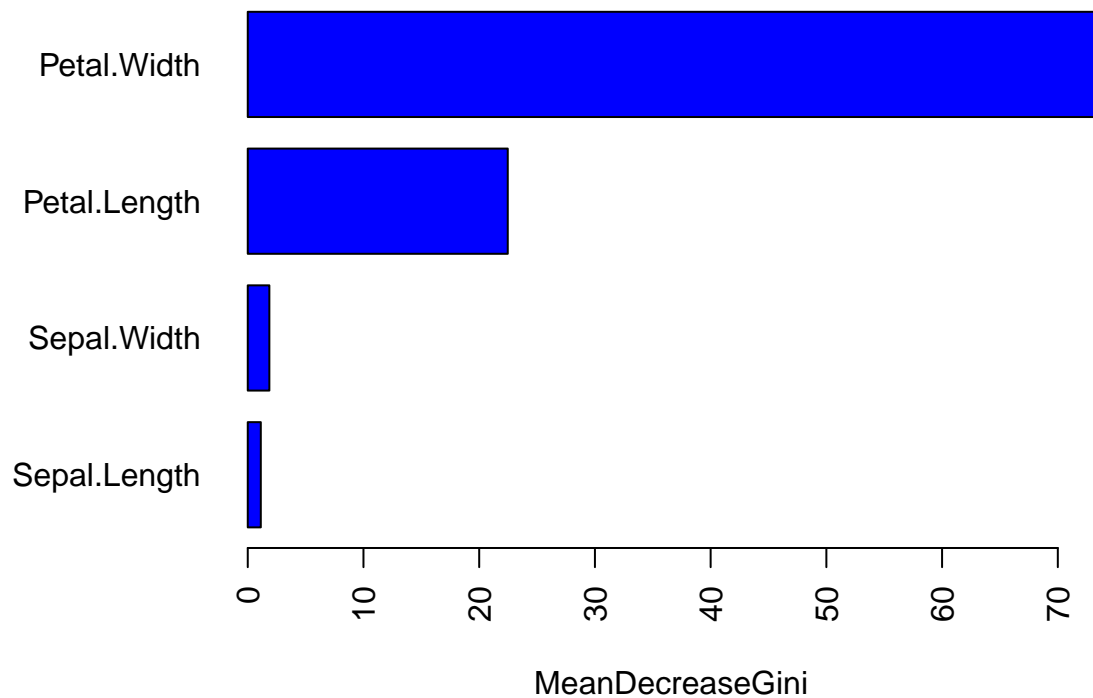
Codes

```
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
```

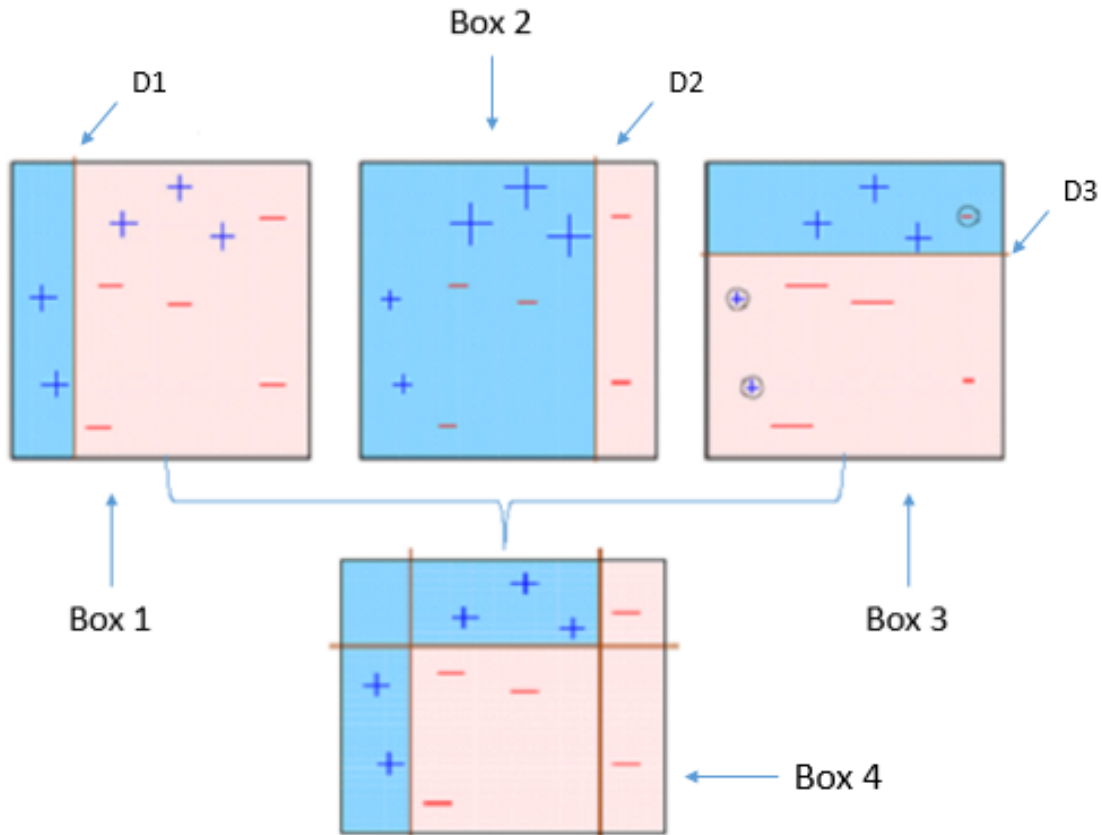
Variable Importance



Boosted Tree

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

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



Codes

Assesment

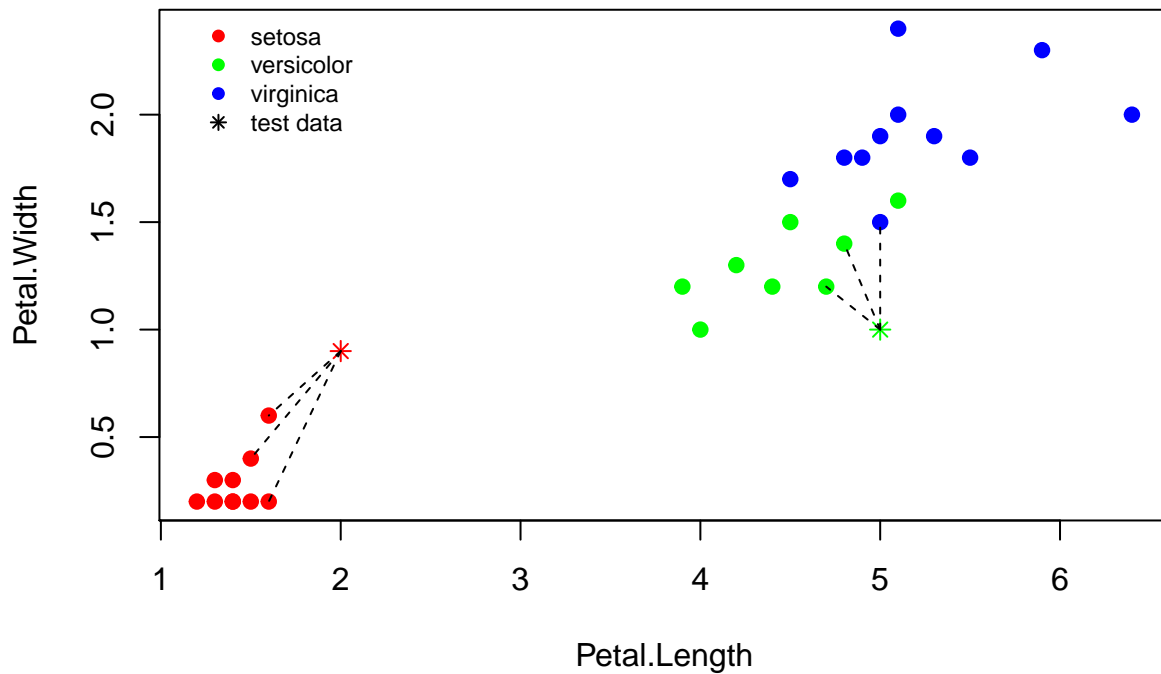
KNN

make sure, you scale the data and also with an example tell why it is important to scale the data

Working principle

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)
```

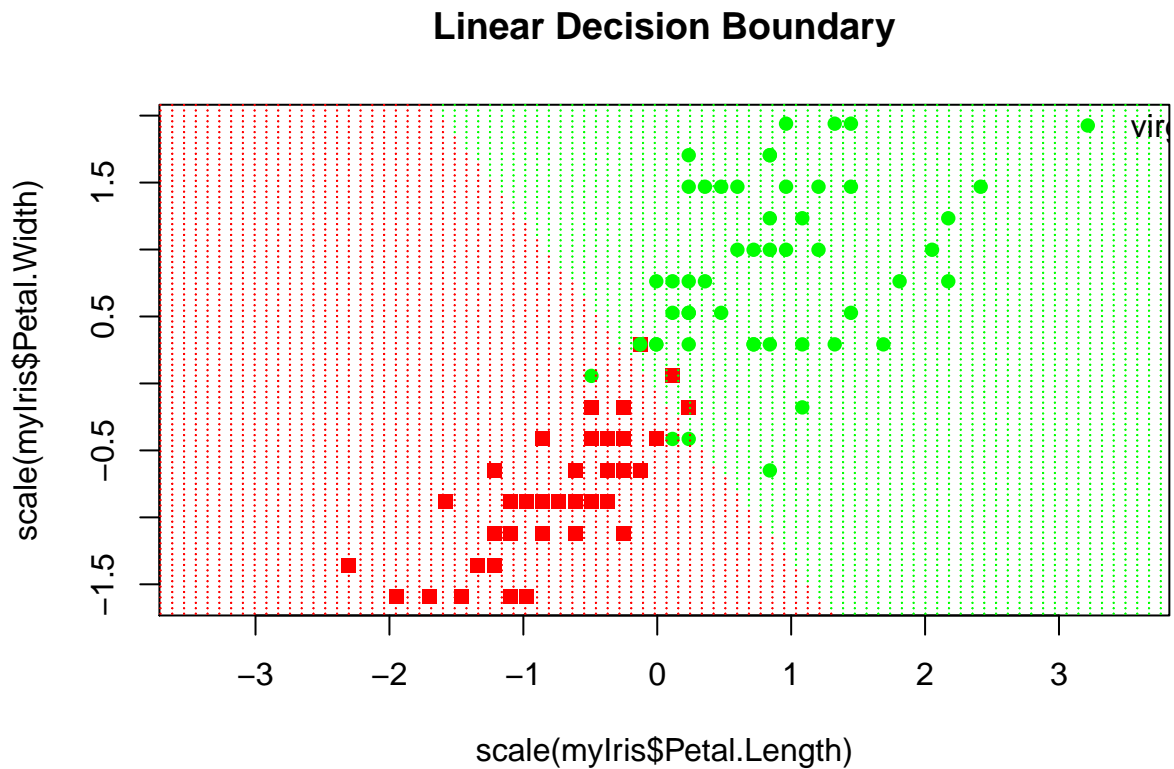
```
##           testClass
## predClass  setosa versicolor virginica
##   setosa      40         0         0
## versicolor   0         40         1
##  virginica    0          2        38
```

SVM

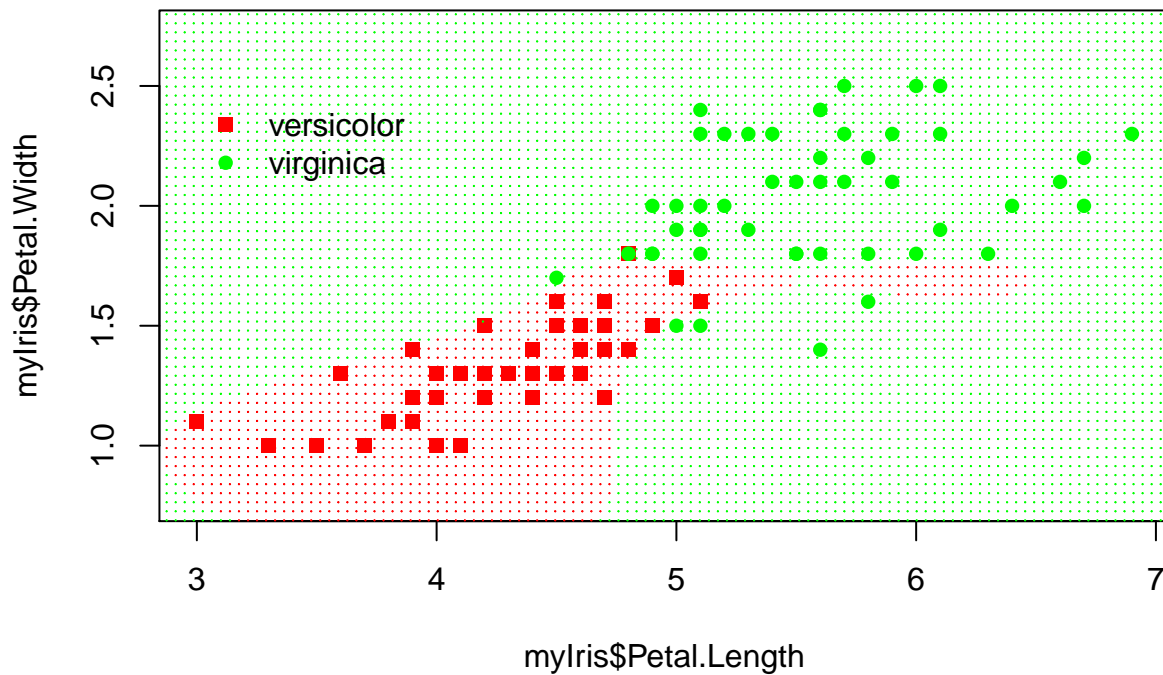
Working Principle

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



Non-Linear Decision Boundary



Codes

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

Assessment

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

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

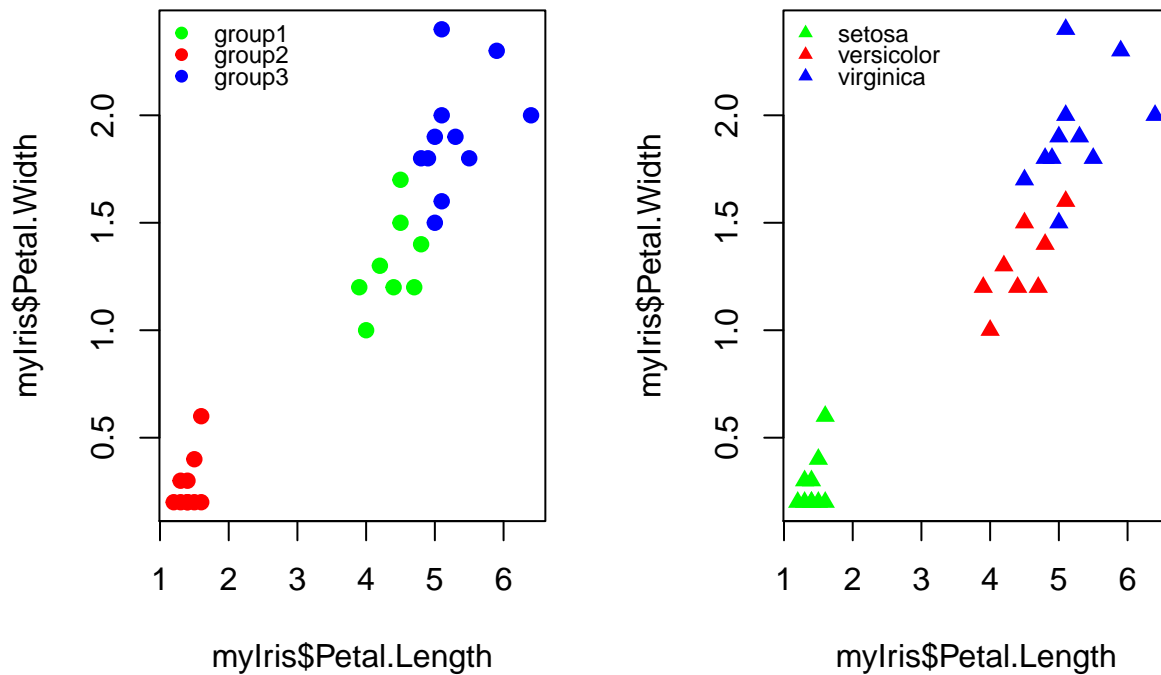


```
mean(prediction==myIris$Species)
```

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

Unsupervised Learning

kmeans clustering

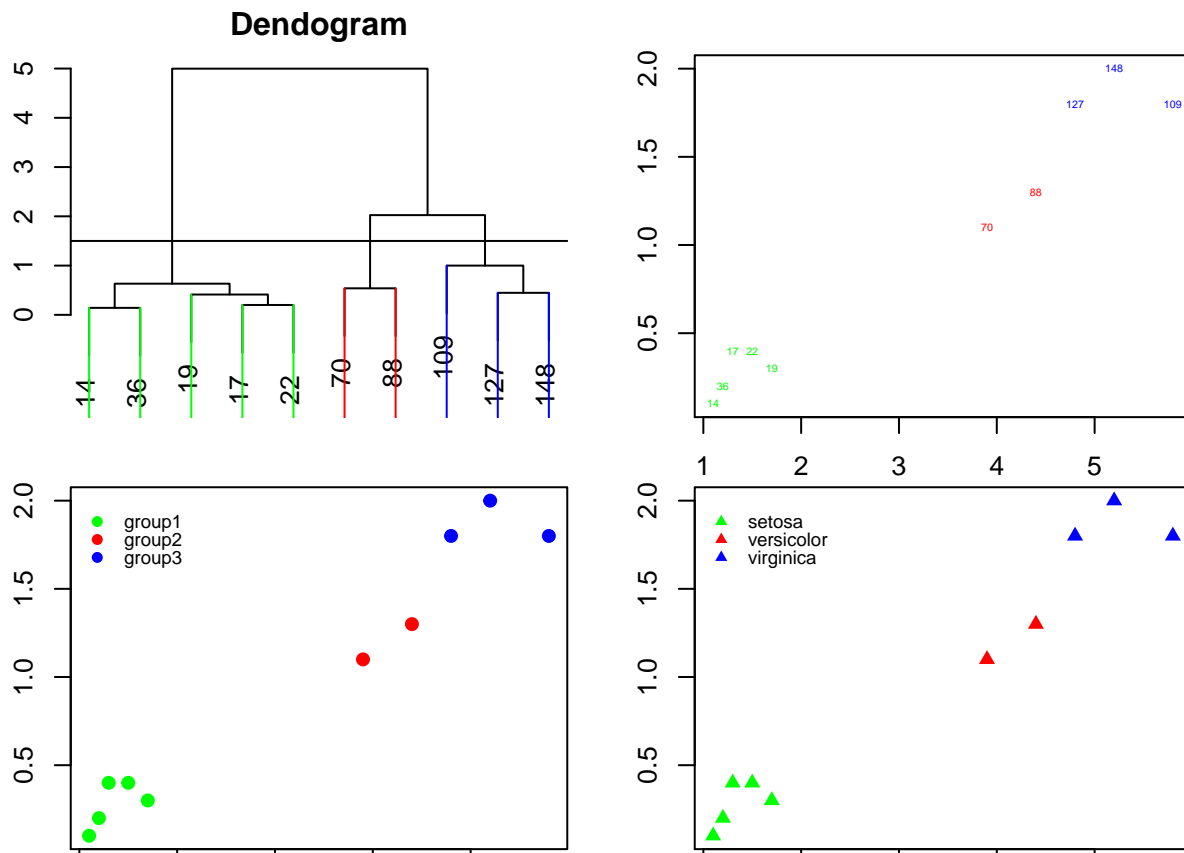


Codes

```
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", "virginica"))
predGroupC <- factor(predGroupC)
table(predGroupC, group)
```

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

Hierarchical 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

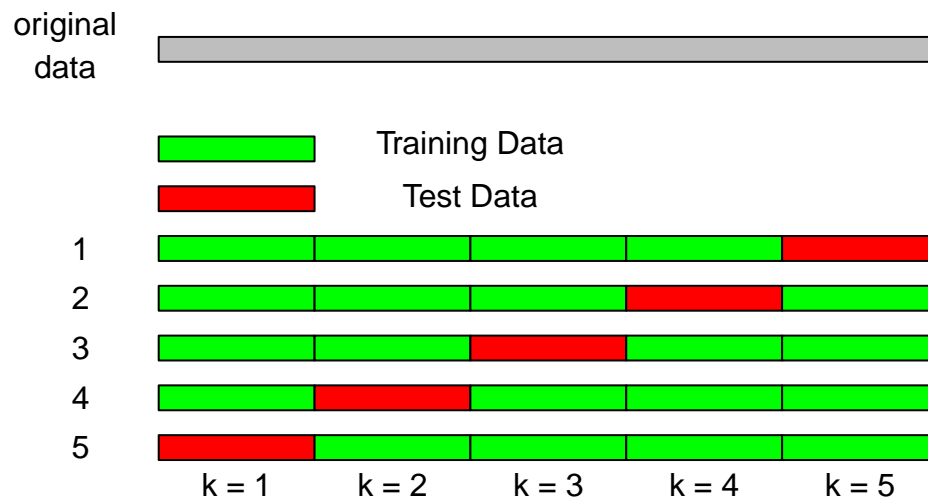
Resampling methods involve repeatedly drawing samples from the original data and refitting it to the model of interest. These methods are very useful in getting additional information about the model.

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 group 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)

5-fold cross-validation illustration


















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	