# Example Analysis

*Irfan Kanat*

*11/04/2015*

## Statistical Models

In this section, I will try to provide an introduction to using two simple statistical models in R: regression and logistic regression.

### Regression

If your dependent variable is continuous you can simply use regression.

For this demonstration, I will use the same Motor Trends dataset I used in Visualization section.

```
data(mtcars) # Get the data
?mtcars # Help on dataset
```

We will use lm() function to fit regular regression.

```
?lm
```

Below I declare a model where I use horse power, cylinders, and transmission type to estimate gas milage. Pay attention to model specification:

```
mpg ~ hp + cyl + am
```

Here the left hand side of the tilde is the dependent variable. and the right hand side has all the predictors we use separated by plus signs.

```
# Fit
reg_0 <- lm( mpg ~ hp + cyl + am, data = mtcars)
summary(reg_0)
```

```
##
## Call:
## lm(formula = mpg ~ hp + cyl + am, data = mtcars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.864 -1.811 -0.158  1.492  6.013
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.88834    2.78422  11.094 9.27e-12 ***
## hp          -0.03688    0.01452  -2.540  0.01693 *
## cyl         -1.12721    0.63417  -1.777  0.08636 .
## am           3.90428    1.29659   3.011  0.00546 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.807 on 28 degrees of freedom
## Multiple R-squared:  0.8041, Adjusted R-squared:  0.7831
## F-statistic: 38.32 on 3 and 28 DF,  p-value: 4.791e-10
```

Look at the R-squared value to see how much variance is explained by the model, the more the better.

You can access estimated values as follows. I used a head function to limit the output.

```
head(reg_0$fitted.values)
```

```
##         Mazda RX4      Mazda RX4 Wag         Datsun 710     Hornet 4 Drive
##          23.97302           23.97302           26.85433           20.06874
## Hornet Sportabout            Valiant
##          15.41740           20.25312
```

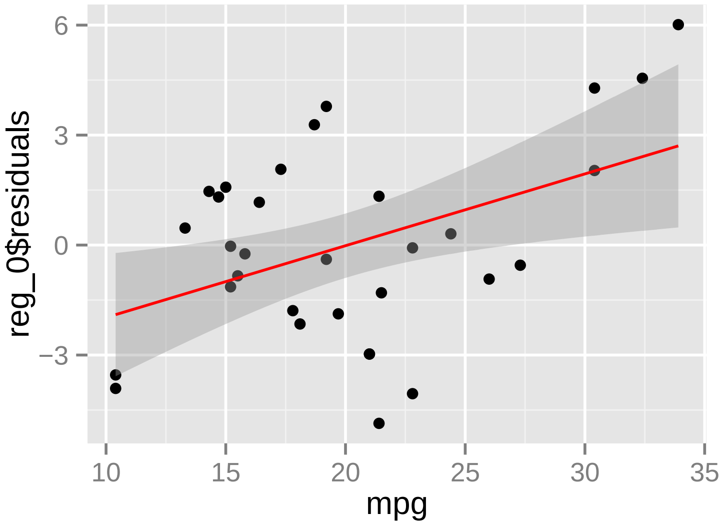
You can use the fitted model to predict new datasets. Here I am modifying Datsun710 to see how the gas milage may have been influenced if the car was automatic instead of manual transmission.

```
newCar <- mtcars[3,] # 3rd observation is Datsun 710
newCar$am <- 0 # What if it was automatic?
predict(reg_0, newdata = newCar) # Estimate went down by 4 miles
```

```
## Datsun 710
##   22.95005
```

One way to see how your model did is to plot residuals. Ideally the residuals should be close to 0 and randomly distributed. If you see a pattern, it indicates misspecification.

```
library(ggplot2)
# Plot the fitted values against real values
qplot(data=mtcars, x = mpg, y = reg_0$residuals) +
  stat_smooth(method = "lm", col = "red")
```



```r
# Are the residuals normally distributed?
shapiro.test(reg_0$residuals) # yes
```

```
##
##  Shapiro-Wilk normality test
##
## data:  reg_0$residuals
## W = 0.98366, p-value = 0.8961
```

Comparing models. If you are using the same dataset, and just adding or removing variables to a model. You can compare models with a likelihood ratio test or an F test. Anova facilitates comparison of simple regression models.

```r
# Add variable wt
reg_1 <- lm( mpg ~ hp + cyl + am + wt, mtcars)

# Aikikae Information Criteria
# AIC lower the better
AIC(reg_0)
```

```
## [1] 162.5849
```

```r
AIC(reg_1)
```

```
## [1] 156.2536
```

```r
# Compare
anova(reg_0, reg_1) # models are significantly different
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ hp + cyl + am
## Model 2: mpg ~ hp + cyl + am + wt
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1     28 220.55
## 2     27 170.00  1    50.555 8.0295 0.008603 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### Logistic Regression

Let us change gears and try to predict a binary variable. For this purpose we will use the logistic regression with a binomial link function. The model estimates the probability of Y=1.

Let us stick to the mtcars dataset and try to figure out if a car is automatic or manual based on predictors.

We will use glm function.

```r
?glm
```

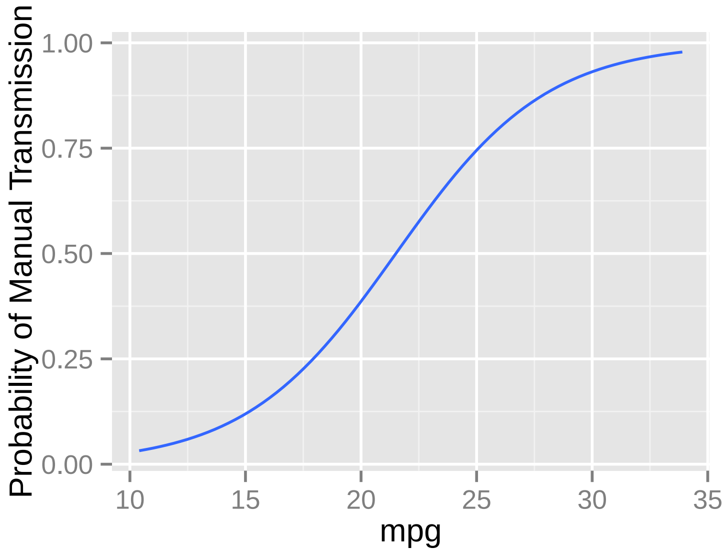
Let us fit the model

```
logit_2 <- glm(am ~ mpg + drat + cyl, data = mtcars, family='binomial')
summary(logit_2)
```

```
##
## Call:
## glm(formula = am ~ mpg + drat + cyl, family = "binomial", data = mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.58367  -0.31020  -0.03757   0.17972   1.75395
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -49.4548    24.1280  -2.050   0.0404 *
## mpg           0.6378     0.4266   1.495   0.1349
## drat          7.2595     3.2702   2.220   0.0264 *
## cyl           1.6115     1.0801   1.492   0.1357
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.23  on 31  degrees of freedom
## Residual deviance: 17.03  on 28  degrees of freedom
## AIC: 25.03
##
## Number of Fisher Scoring iterations: 7
```

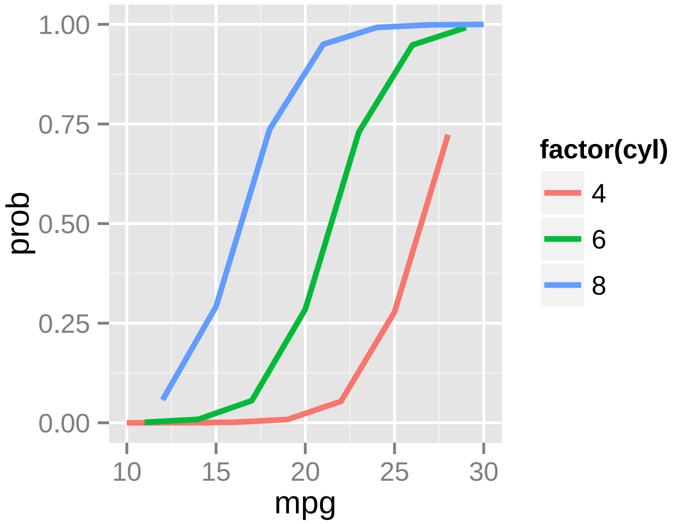
Visualize the results.

```
ggplot(mtcars, aes(x = mpg, y = am)) +
    stat_smooth(method="glm", family="binomial", se=FALSE)+
# Bonus: rename the y axis label
        ylab('Probability of Manual Transmission')
```



How about plotting results for number of cylinders? We will need to process the data a little bit.

```
# Create a new dataset with varying number of cylinders and other variables fixed at mean levels.
mtcars2<-data.frame(mpg = rep(10:30, 3),drat = mean(mtcars$drat), disp = mean(mtcars$disp), cyl = rep(c
# Predict probability of new data
mtcars2$prob<-predict(logit_2, newdata=mtcars2, type = "response")

# Plot the results
ggplot(mtcars2, aes(x=mpg, y=prob)) +
  geom_line(aes(colour = factor(cyl)), size = 1)
```



Diagnostics with logistic regression.

```
library(caret)
```

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

```
# Let us compare predicted values to real values
mtcars$prob <- predict(logit_2, type="response")
# Prevalence of Manual Transmission
mean(mtcars$am)
```

```
## [1] 0.40625
```

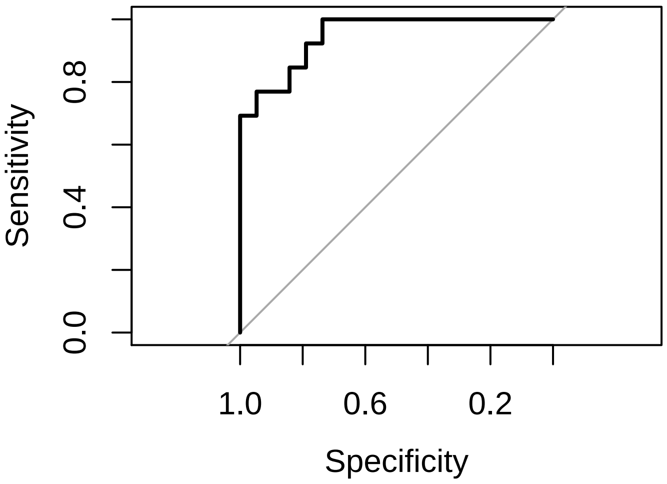
```
# Create predict variable
mtcars$pred <- 0
# If probability is greater than .6 (1-prevalence), set prediction to 1
mtcars[mtcars$prob>.6, 'pred'] <- 1

# Confusion Matrix
confusionMatrix(table(mtcars[,c("am", "pred")]))
```

```
## Confusion Matrix and Statistics
##
##    pred
## am   0  1
##   0 18  1
##   1  3 10
##
##                Accuracy : 0.875
##                  95% CI : (0.7101, 0.9649)
##     No Information Rate : 0.6562
##     P-Value [Acc > NIR] : 0.005004
##
##                   Kappa : 0.7344
##  Mcnemar's Test P-Value : 0.617075
##
##             Sensitivity : 0.8571
##             Specificity : 0.9091
##          Pos Pred Value : 0.9474
##          Neg Pred Value : 0.7692
##              Prevalence : 0.6562
##          Detection Rate : 0.5625
##    Detection Prevalence : 0.5938
##       Balanced Accuracy : 0.8831
##
##        'Positive' Class : 0
##
```

```r
## ROC CURVE
# Load the necessary library
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
# Calculate the ROC curve using the predicted probability vs actual values
logit_2_roc <- roc(am~prob, mtcars)
# Plot ROC curve
plot(logit_2_roc)
```



```
##
## Call:
## roc.formula(formula = am ~ prob, data = mtcars)
##
## Data: prob in 19 controls (am 0) < 13 cases (am 1).
## Area under the curve: 0.9474
```

## Caret Package

Caret package is a wrapper that brings together functionality from 27 packages. Caret supports estimating over 150 models including bayesian, SVM, discriminant analysis, regressions, neural networks, and more.

```
# Run below commands to get a list of related packages
available.packages()["caret","Depends"]
available.packages()["caret","Suggests"]
```

Caret package aims to be the go to package for your predictive analytics needs. Thus it not only covers model training, but also data manipulation, visualization, and parallelization capabilities.

Since so many packages involved, the installation takes a while.

```
install.packages("caret", dependencies = c("Depends", "Suggests"))
```

The main idea is comparing performance of alternate models and finding the best fit. Clarification of ambiguities: 1. Alternate models: test various model parameters 2. Best fit: according to various metrics 3. Compare performance: through resampling.

## Model Training - Classification Trees with caret Package

In this section, I will be presenting a classification tree with caret package. The example presented here follows closely Kuhn's UseR 2013 presentation. I had to leave out significant portions of Kuhn's work to fit it into the scope of our workshop. Please refer to the original material for details.

### Data

We will use segmentationData from the caret package.

```
library(caret) # Load necessary library
data(segmentationData) # Obtain Dataset
```

We do not need the Cell identifiers.

```
# Drop one column
segmentationData <- segmentationData[,!(colnames(segmentationData)=='Cell')]
```

The data has a variable “Case” to indicate Training, vs Testing observations. We will obtain subsets based on this variable.

```
# Data set has a variable that separates training vs testing
Training <- segmentationData[segmentationData$Case == 'Train', ] #$ One Way
Testing <- subset(segmentationData, Case == 'Test') # Another way
# Drop the now defunct Case variable
Training <- Training[,!(colnames(Training)=='Case')] # Drop Case column
Testing <- subset(Testing, select=-c(Case)) # Another way
```

### Classification Trees without Caret

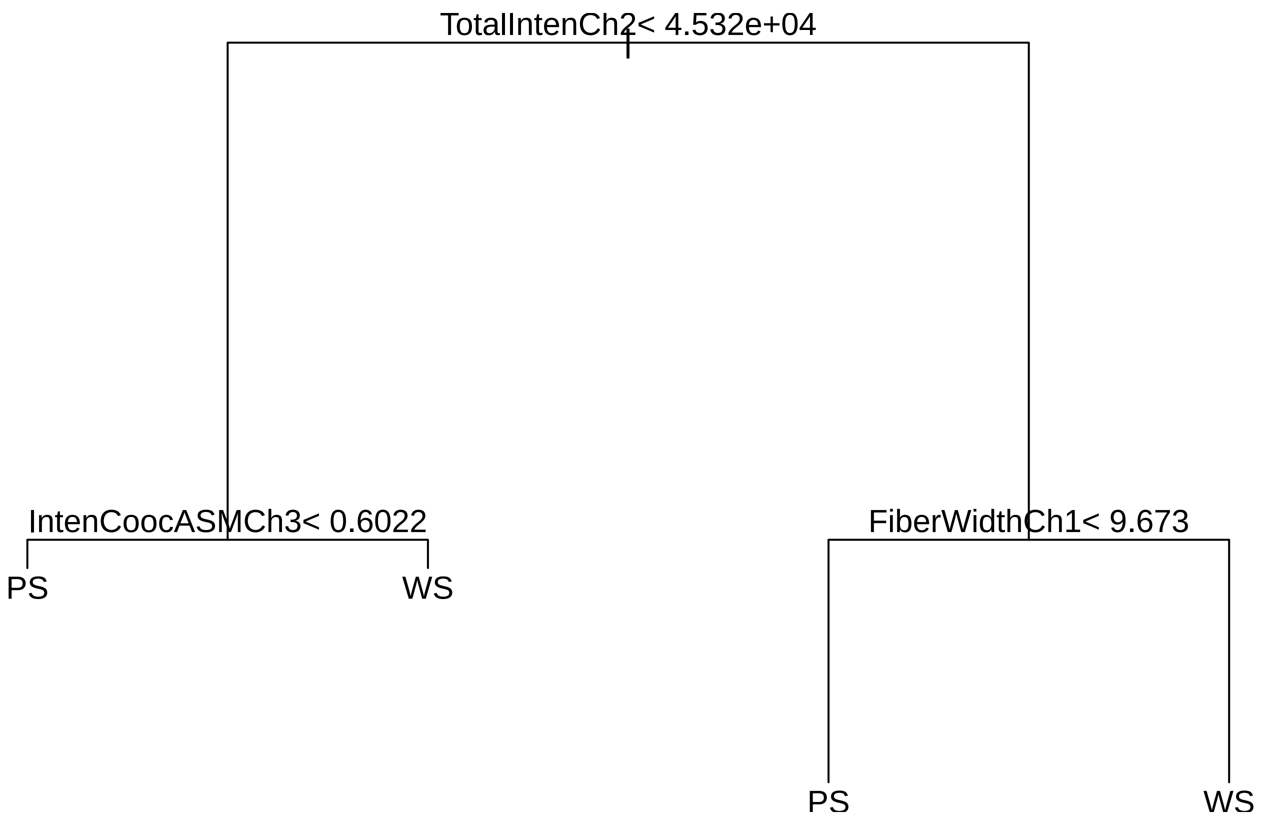
I want to demonstrate how Classification Trees can be fit without caret first, so that the contribution of caret’s train function becomes clearer.

Let us first fit a tree of a limited depth.

```
library(rpart) # Load necessary library
rpart_1 <- rpart(Class ~ ., data = Training,
            control = rpart.control(maxdepth=2)) # Fit a shallow tree
rpart_1 # View results
```

```
## n= 1009
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 1009 373 PS (0.63032706 0.36967294)
##   2) TotalIntenCh2< 45324.5 454  34 PS (0.92511013 0.07488987)
##     4) IntenCoocASMCh3< 0.6021832 447  27 PS (0.93959732 0.06040268) *
##     5) IntenCoocASMCh3>=0.6021832 7   0 WS (0.00000000 1.00000000) *
##   3) TotalIntenCh2>=45324.5 555 216 WS (0.38918919 0.61081081)
##     6) FiberWidthCh1< 9.673245 154  47 PS (0.69480519 0.30519481) *
##     7) FiberWidthCh1>=9.673245 401 109 WS (0.27182045 0.72817955) *
```

```
plot(rpart_1) # Visualization
text(rpart_1) # Text
```



When you fit a tree with rpart, it conducts as many spits as possible, then use 10 fold cross validation to prune the tree.

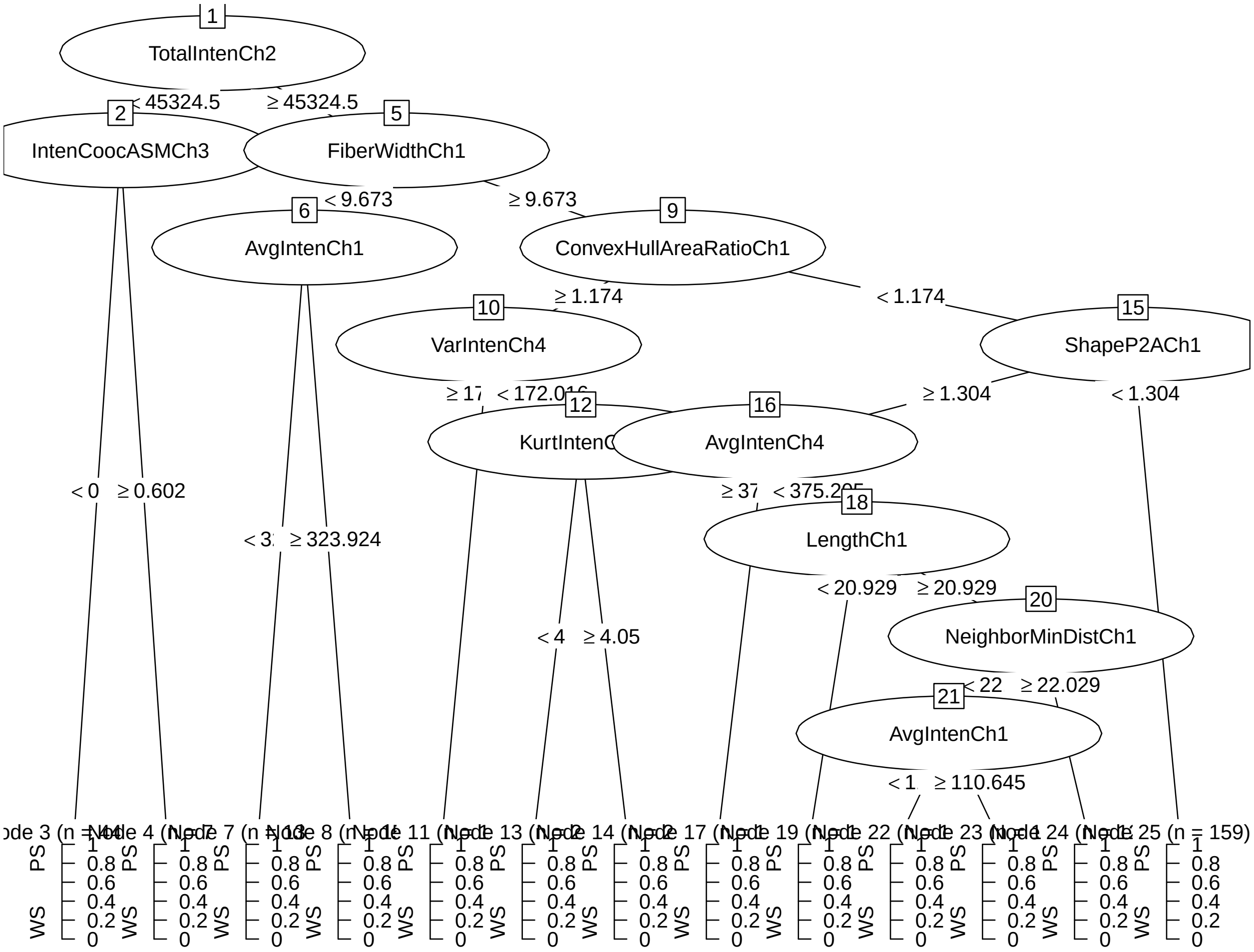
```r
# Fit a larger tree and prune it
# Rpart does 10 fold cross validation
rpart_2 <- rpart(Class ~ ., data = Training)
```

partykit package provides beter plots for classification trees.

```r
library(partykit)
```

```
## Loading required package: grid
```

```r
plot(as.party(rpart_2))
```



```r
# Validate the model
rpart_2Test <- predict(rpart_2, Testing, type='class')
confusionMatrix(rpart_2Test, Testing$Class) # Get the benchmark$
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  PS  WS
##         PS 561 108
##         WS 103 238
##
##                Accuracy : 0.7911
##                  95% CI : (0.7647, 0.8158)
##     No Information Rate : 0.6574
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5346
##  Mcnemar's Test P-Value : 0.783
##
##             Sensitivity : 0.8449
##             Specificity : 0.6879
##          Pos Pred Value : 0.8386
##          Neg Pred Value : 0.6979
##              Prevalence : 0.6574
##          Detection Rate : 0.5554
##    Detection Prevalence : 0.6624
##       Balanced Accuracy : 0.7664
##
##        'Positive' Class : PS
##
```

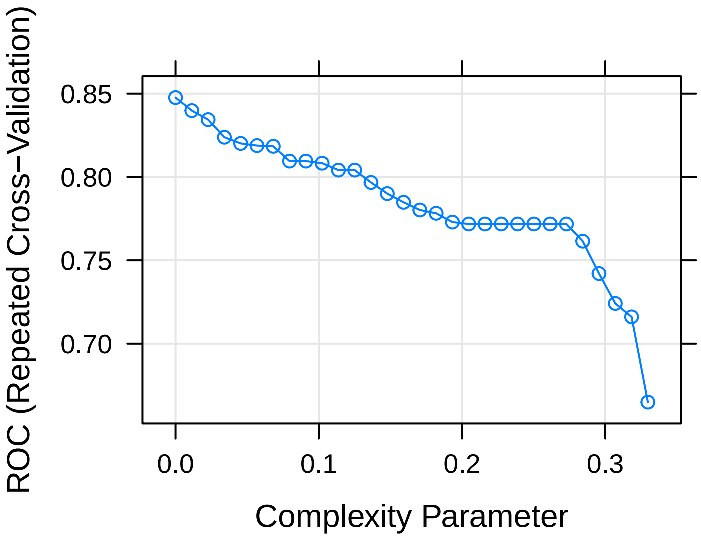
### Model Tuning - Classification Trees with Caret

caret package allows you to change tuning parameters and resampling strategies for the models. Below we instruct caret to use ROC curve with k-fold cross validation repeated 3 times to pick the best model.

```r
library(caret)
##  Set Training Parameters
# Triple Cross Validation
cvCtrl <- trainControl(method = "repeatedcv", repeats = 3, # K-fold cross validation repeated thrice
                summaryFunction = twoClassSummary, classProbs = TRUE) # Class probabilities for ROC

# Use caret to fit a model and fine tune the fit
CarrotTree <- train(Class ~., data=Training, method='rpart',
                    trControl=cvCtrl, metric = "ROC", tuneLength=30) # Change metric to ROC
```

Let us see the performance of models over various levels of tuning parameter.

```r
plot(CarrotTree)
```



Let us see the final model

```r
# Output omited to conserve space
plot(as.party(CarrotTree$finalModel))
```

#### Validate Using Testing Data

Predict the class membership with test data and then plot out the confusion matrix for performance metrics of this model over test data.

```
# Testing
CarrotTree_Test <- predict(CarrotTree, Testing)
head(CarrotTree_Test) # Compare
```

```
## [1] PS PS WS WS PS PS
## Levels: PS WS
```

```
head(Testing$Class) # and Contrast
```

```
## [1] PS PS WS WS PS WS
## Levels: PS WS
```

```
# Evaluate Model Performance
confusionMatrix(CarrotTree_Test, Testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  PS  WS
##         PS 554 104
##         WS 110 242
##
##                Accuracy : 0.7881
##                  95% CI : (0.7616, 0.8129)
##     No Information Rate : 0.6574
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5316
##  Mcnemar's Test P-Value : 0.7325
##
##             Sensitivity : 0.8343
##             Specificity : 0.6994
##          Pos Pred Value : 0.8419
##          Neg Pred Value : 0.6875
##              Prevalence : 0.6574
##          Detection Rate : 0.5485
##    Detection Prevalence : 0.6515
##       Balanced Accuracy : 0.7669
##
##        'Positive' Class : PS
##
```

## Model Training - Discriminant Analysis with caret Package

In this section I will present a classification example using caret package. The example presented here follows closely the caret package vignette. I only made a few changes to fit it into the workshop's time frame. So if you need any clarification, you can read up more in the vignette.

### Data

We will use the sonar dataset from mlbench package. It has 208 observations of 60 predictor variables and a binary class variable as dependent. We do not know what these 60 variables are.

```
# Dataset comes with mlbench package
library(mlbench)
# Load dataset into the current workspace
data(Sonar)
```

### SPLIT THE DATA

This dataset is not conveniently split as the previous set was, we need to create our own subsets. Caret provides functionality to split the data into a training and a testing set while at the same time preserving the distribution of dependent variable.

createDataPartition function receives the dependent variable, proportion of training set in the whole of dataset as parameters.

```
library(caret) # Load Library
set.seed(107)  # Set random number seed for reproducibility
# Create an index of observations to be included in Training
indexTrain <- createDataPartition(y = Sonar$Class, p = .75, list = FALSE)

# Split the data using the index
Train <- Sonar[indexTrain,]
Test  <- Sonar[-indexTrain,]
```

### Training a Discriminant Model

We will be using a PLS DA model to train.

First step is to set resampling and validation strategy. Here we are using 3 resamplings of 10-fold cross validation. We also configure the trainer to produce predicted probabilities to be used in ROC calculation.

```
## Declare Tuning Control Parameters
ctrl <- trainControl(method = "repeatedcv", # K-fold cross validation
                     repeats = 3, # Repeat resampling 3 times
                     classProbs = TRUE, # Calculate predicted prob for ROC
                     summaryFunction = twoClassSummary) # Set performance metrics for binary
```

Below we train the model with varying parameters. The tune length in this case specifies maximum the number of components to be extracted.

```
plsFit <- train(Class ~ ., data = Train, method = "pls",
        tuneLength = 10,    # Number of component sets to be evaluated (more is better)
        trControl = ctrl, # Use control parameters from above
        metric = "ROC" ,  # Criteria ROC
        preProc = c("center", "scale")) # Center and scale the predictors
```

```
## Loading required package: pls
##
## Attaching package: 'pls'
##
## The following object is masked from 'package:caret':
##
##     R2
```

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

Let us review the models that were fit.

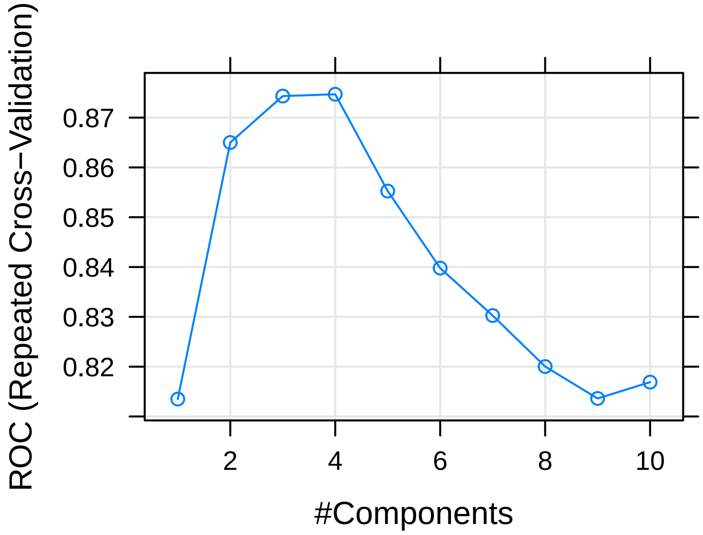
```
plsFit
```

```
## Partial Least Squares
##
## 157 samples
##  60 predictors
##   2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 142, 141, 142, 142, 142, 142, ...
## Resampling results across tuning parameters:
##
##   ncomp  ROC        Sens       Spec       ROC SD      Sens SD    Spec SD
##    1     0.8134921  0.7291667  0.7291667  0.11844879  0.1387811  0.1935289
##    2     0.8649967  0.7694444  0.8041667  0.08381907  0.1416676  0.1521373
##    3     0.8743221  0.7476852  0.8363095  0.08548836  0.1726683  0.1375303
##    4     0.8746858  0.7578704  0.7642857  0.08443793  0.1512983  0.1539276
##    5     0.8552497  0.7152778  0.7767857  0.09587112  0.1771056  0.1584577
##    6     0.8397817  0.7337963  0.7732143  0.09814150  0.1726297  0.1749730
##    7     0.8302579  0.7101852  0.7916667  0.10762747  0.1992674  0.1655561
##    8     0.8200231  0.7157407  0.7607143  0.12724476  0.1805642  0.1621515
##    9     0.8136161  0.7245370  0.7696429  0.12961432  0.1847286  0.1593987
##   10     0.8168981  0.7203704  0.7559524  0.11678705  0.1912620  0.1436503
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was ncomp = 4.
```

As you can see the area under the ROC curve increases up till 4 components and starts declining afterwards. By default the model with 4 components is used.

Plotting this fit would demonstrate the change in area under ROC for different components.

```
# evaluate the performance of different number of components extracted
plot(plsFit)
```



### Validate with Test Data

Let us see how our fitted model performs with Test data.

```
plsPredict <- predict(plsFit, newdata = Test) # Predict results
head(plsPredict) # View predictions
```

```
## [1] R M M R M R
## Levels: M R
```

```
head(Test$Class) # View actual$
```

```
## [1] R R R R R R
## Levels: M R
```

Get confusion matrix (predicted vs actual) for performance reports.

```
confusionMatrix(data = plsPredict, Test$Class)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  M  R
##          M 17  6
##          R 10 18
##
##                Accuracy : 0.6863
##                  95% CI : (0.5411, 0.8089)
##     No Information Rate : 0.5294
##     P-Value [Acc > NIR] : 0.01674
##
##                   Kappa : 0.3761
##  Mcnemar's Test P-Value : 0.45325
```

```
##
##             Sensitivity : 0.6296
##             Specificity : 0.7500
##          Pos Pred Value : 0.7391
##          Neg Pred Value : 0.6429
##              Prevalence : 0.5294
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.4510
##       Balanced Accuracy : 0.6898
##
##        'Positive' Class : M
##
```