Chapter 4: Classification

a.h.thiery@nus.edu.sg

Version: 0.1

Contents

1 Naive Bayes classifier

```
set.seed(4240)  #for reproducibility
sex = as.factor(c("M","M","F","F","F","M","M","M","M","F","F","F"))
height = c(1.70, 1.67, 1.60, 1.62, 1.54, 1.82, 1.75,
           1.7, 1.69, 1.6, 1.70, 1.70)
weight = c(70, 71, 60, 50, 55, 80, 68, 62, 69, 55, 66, 70)
shoes = as.factor(c("Few","Avg","Lot","Lot","Avg",
          "Avg","Few","Lot","Lot","Avg","Avg","Lot"))
data.training = data.frame(sex, height, weight, shoes)


#Library for Naive Bayes Classifier
library(e1071)

#laplace = regularization parameter
classifier.NB = naiveBayes(sex ~ .,data.training, laplace = 0)
print(classifier.NB$apriori)

## Y
## F M
## 6 6

print(classifier.NB$tables)

## $height
##    height
## Y       [,1]       [,2]
```

```
##   F 1.626667 0.06282250
##   M 1.721667 0.05492419
##
## $weight
##    weight
## Y       [,1]     [,2]
##   F 59.33333 7.527727
##   M 70.00000 5.830952
##
## $shoes
##    shoes
## Y         Avg       Few       Lot
##   F 0.5000000 0.0000000 0.5000000
##   M 0.3333333 0.3333333 0.3333333
```

```
#laplace = regularization parameter
classifier.NB.regularized = naiveBayes(sex ~ .,data.training, laplace = 2)
print(classifier.NB.regularized$apriori)
```

```
## Y
## F M
## 6 6
```

```
print(classifier.NB.regularized$tables)
```

```
## $height
##    height
## Y       [,1]       [,2]
##   F 1.626667 0.06282250
##   M 1.721667 0.05492419
##
## $weight
##    weight
## Y       [,1]     [,2]
##   F 59.33333 7.527727
##   M 70.00000 5.830952
##
## $shoes
##    shoes
## Y         Avg       Few       Lot
##   F 0.4166667 0.1666667 0.4166667
##   M 0.3333333 0.3333333 0.3333333
```

How would you classify an individual who is $1.70m$ tall, weights $65kg$ and has an average number of shoes?

```
nb.prediction = predict(classifier.NB,
                      data.frame(height=1.7, weight=65, shoes="Avg"),
                      type="raw")

print(nb.prediction)
```

```
##              F         M
```

```
## [1,] 0.3767206 0.6232794
```

2 Titanic dataset: Naive Bayes

Fir, let us load the Titanic dataset and create a training and testing set.

```
#<<>>=
#load covariates
#setwd("/home/alex/Dropbox/teaching/2015_ST4240/chap5_slides/code")
filename = "Titanic.csv"
titanic = read.csv(filename, header = TRUE, sep = ",")

#for simplicity, drop the following covariates
drops <- c("PassengerId","Name","Ticket", "Cabin")
titanic = titanic[,!(names(titanic) %in% drops)]
print(names(titanic))

## [1] "Survived" "Pclass"   "Sex"      "Age"      "SibSp"    "Parch"
## [7] "Fare"     "Embarked"

head(titanic)

##   Survived Pclass    Sex Age SibSp Parch    Fare Embarked
## 1        0      3   male  22     1     0  7.2500        S
## 2        1      1 female  38     1     0 71.2833        C
## 3        1      3 female  26     0     0  7.9250        S
## 4        1      1 female  35     1     0 53.1000        S
## 5        0      3   male  35     0     0  8.0500        S
## 6        0      3   male  NA     0     0  8.4583        Q

#remove rows with NAs, missing information or fare paid less than 5
titanic = na.omit(titanic)
titanic <- titanic[titanic$Embarked != "",]
titanic <- titanic[titanic$Fare > 5,]


#"sex" and "Embarked" and "Pclass" are a categorical data
titanic$Sex = as.factor(titanic$Sex)
titanic$Embarked = as.factor(titanic$Embarked)
titanic$Pclass = as.factor(titanic$Pclass)

#create training and test set
n_total = length(titanic[,1])
train = sample(1:n_total, 500)
```

Let us first a Naives Bayes classifier (without any regularization), plot a ROC curve and compute the AUC. We will also superpose the results obtained by a naive algorithm that makes random guesses, just for sanity check.

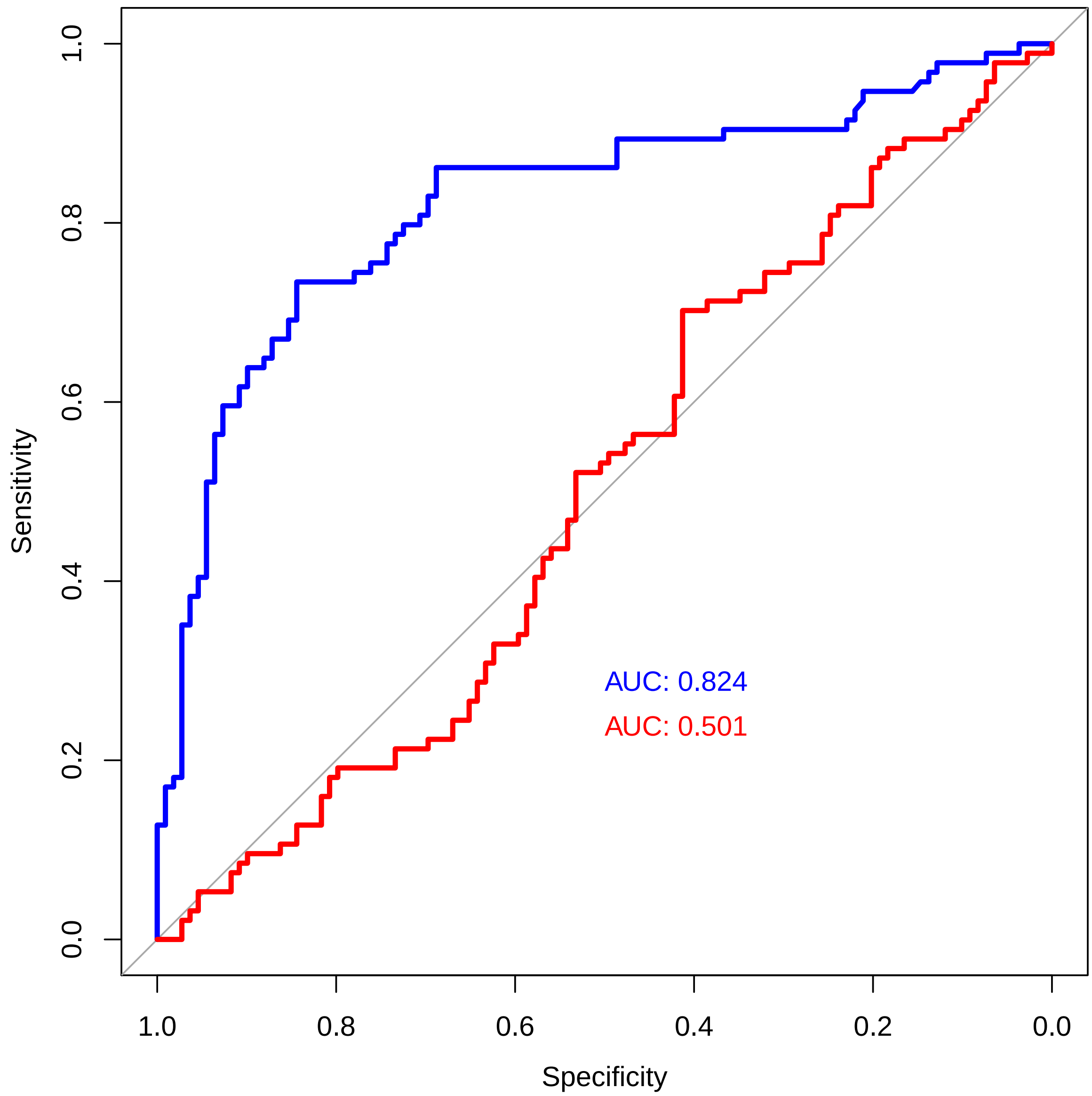
```r
#naive Bayes
naive = naiveBayes(Survived ~ ., titanic[train,], type="raw")
prediction.naive = predict(naive, titanic[-train,-1], type = c("raw"))

#ROC + AUC
library(pROC)
```

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

```r
plot.roc(titanic[-train, "Survived"], prediction.naive[,1], col="blue",
         lwd=3, print.auc=TRUE,print.auc.y = 0.3)
```

```
##
## Call:
## plot.roc.default(x = titanic[-train, "Survived"], predictor = prediction.naive[,     1], col = "blue
##
## Data: prediction.naive[, 1] in 109 controls (titanic[-train, "Survived"] 0) > 94 cases (titanic[-trai
## Area under the curve: 0.8235
```

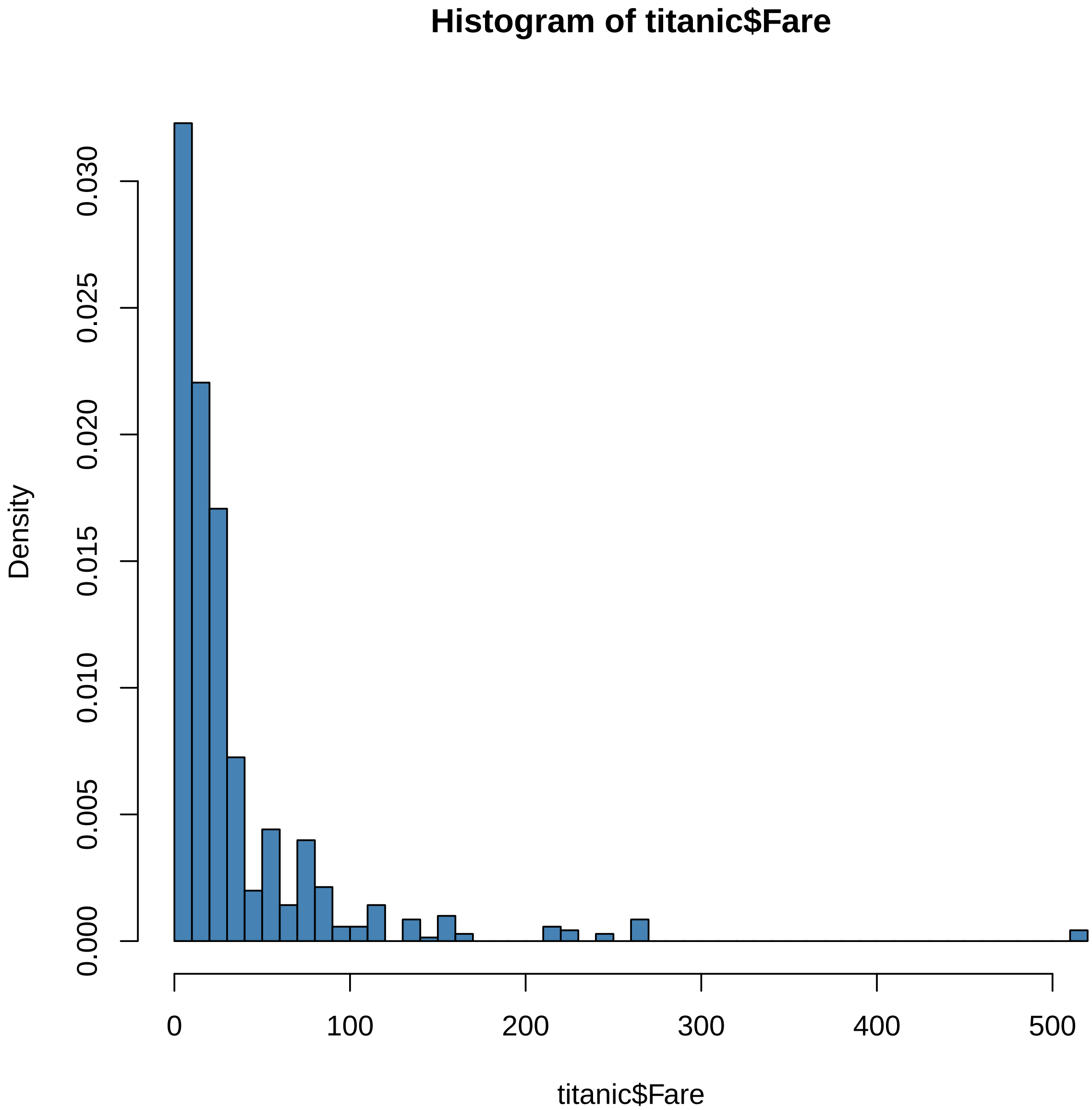
```r
#random guesses
prediction.random = runif(n_total - 500)
plot.roc(titanic[-train, "Survived"], prediction.random, col="red",
         lwd=3, print.auc=TRUE, print.auc.y = 0.25, add=TRUE)
```



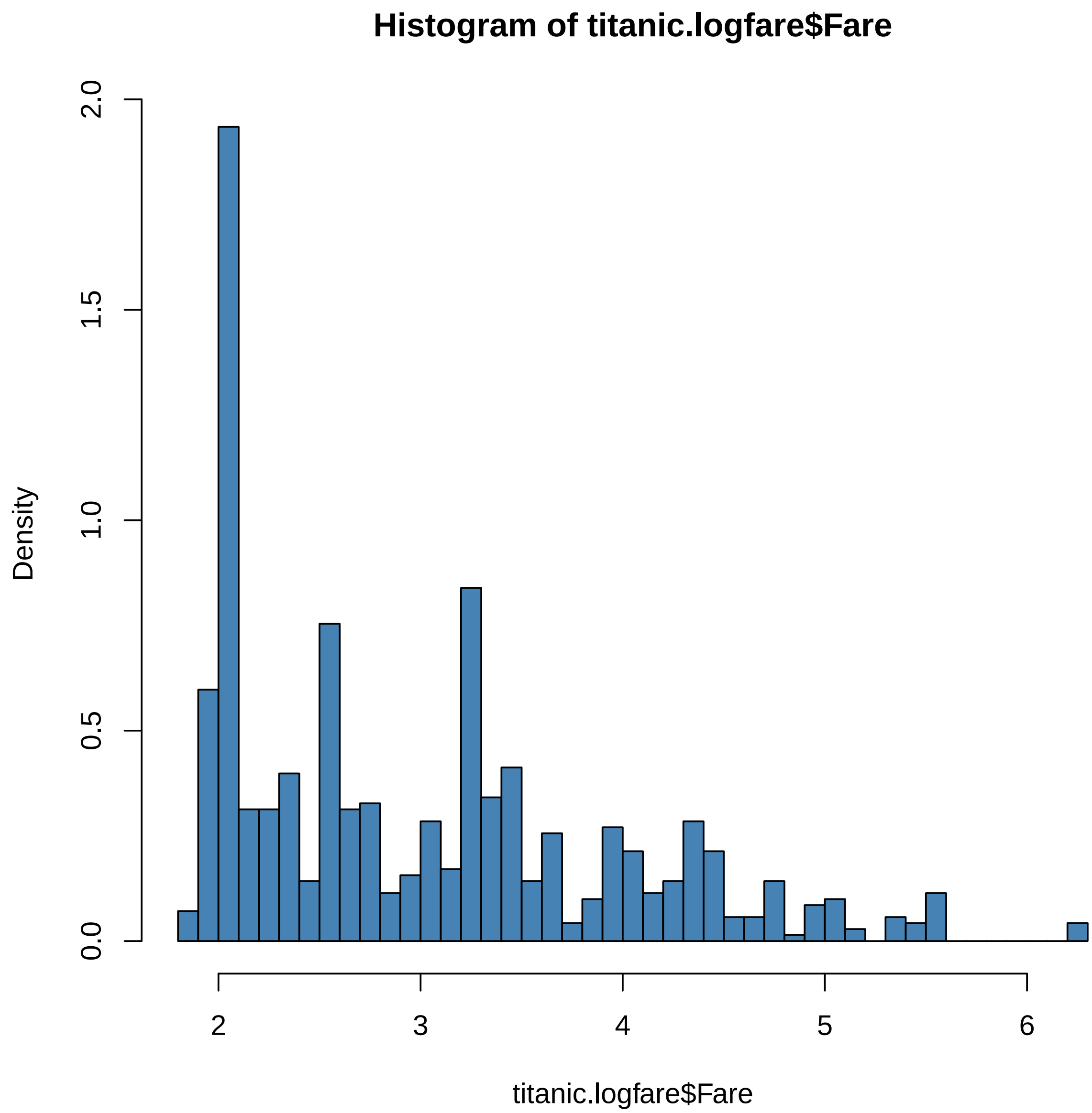

```
##
## Call:
## plot.roc.default(x = titanic[-train, "Survived"], predictor = prediction.random,     col = "red", lwd
##
## Data: prediction.random in 109 controls (titanic[-train, "Survived"] 0) < 94 cases (titanic[-train,
## Area under the curve: 0.5013
```

Sometimes it helps to transform some variables.

```
#let us look at the "Fare" variable
hist(titanic$Fare, nclass=50, col="steelblue", proba=TRUE)
```



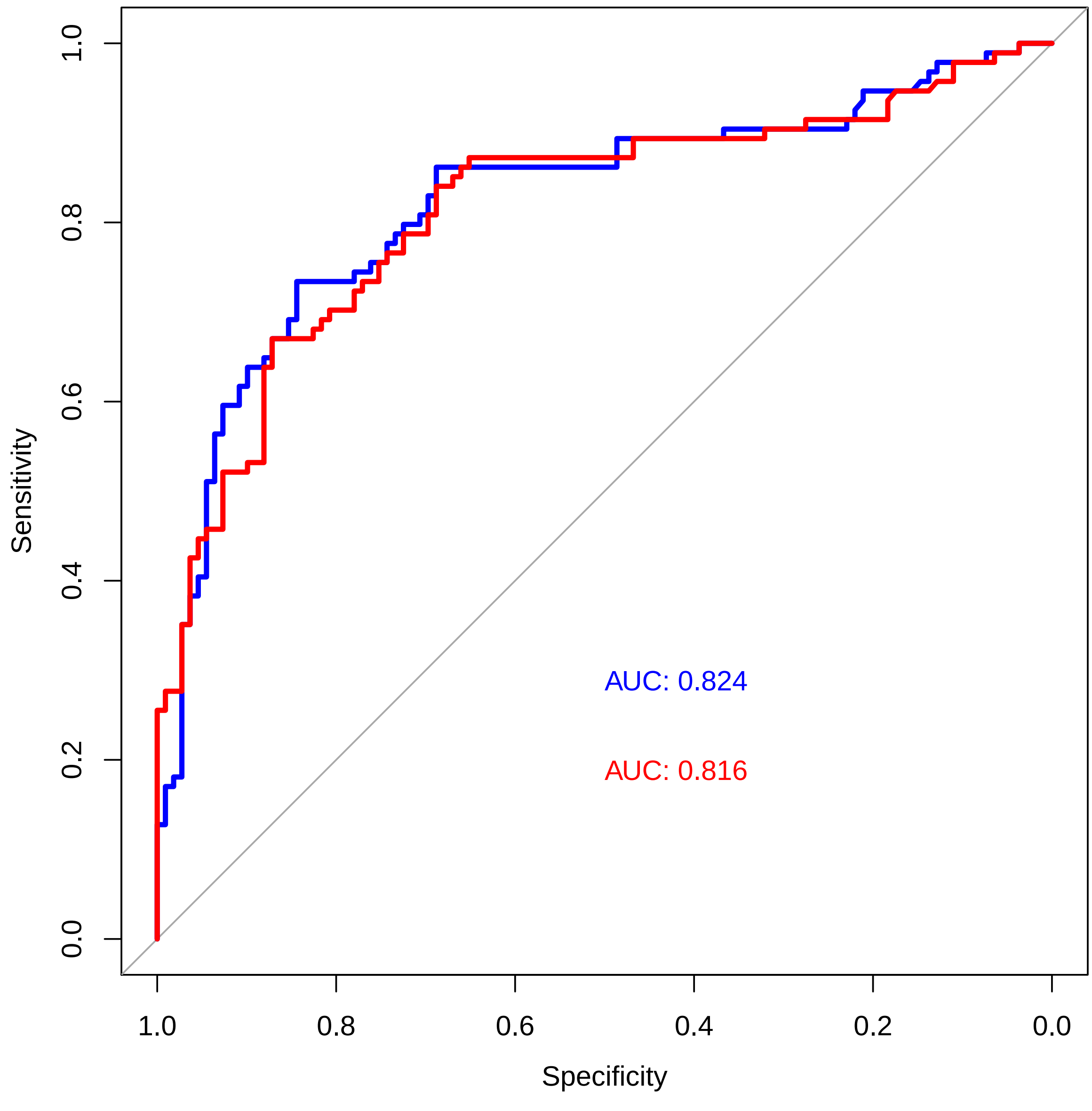

```r
#it is quite far from Gaussian, let us look at that on log-scale
titanic.logfare = titanic
titanic.logfare$Fare = log(titanic.logfare$Fare)
hist(titanic.logfare$Fare, nclass=50, col="steelblue", proba=TRUE)
```




```r
#fit a naiveBayes classifier
naive.logfare = naiveBayes(Survived ~ ., titanic.logfare[train,], type="raw")
prediction.naive.logfare = predict(naive.logfare, titanic.logfare[-train,-1],
                                   type = c("raw"))

#ROC + AUC
plot.roc(titanic[-train, "Survived"], prediction.naive[,1], col="blue",
         lwd=3, print.auc=TRUE,print.auc.y = 0.3)

##
## Call:
## plot.roc.default(x = titanic[-train, "Survived"], predictor = prediction.naive[,     1], col = "blue
```

```
##
## Data: prediction.naive[, 1] in 109 controls (titanic[-train, "Survived"] 0) > 94 cases (titanic[-tra
## Area under the curve: 0.8235
```

```
plot.roc(titanic[-train, "Survived"], prediction.naive.logfare[,1],
         col="red", add=TRUE,
         lwd=3, print.auc=TRUE,print.auc.y = 0.2)
```

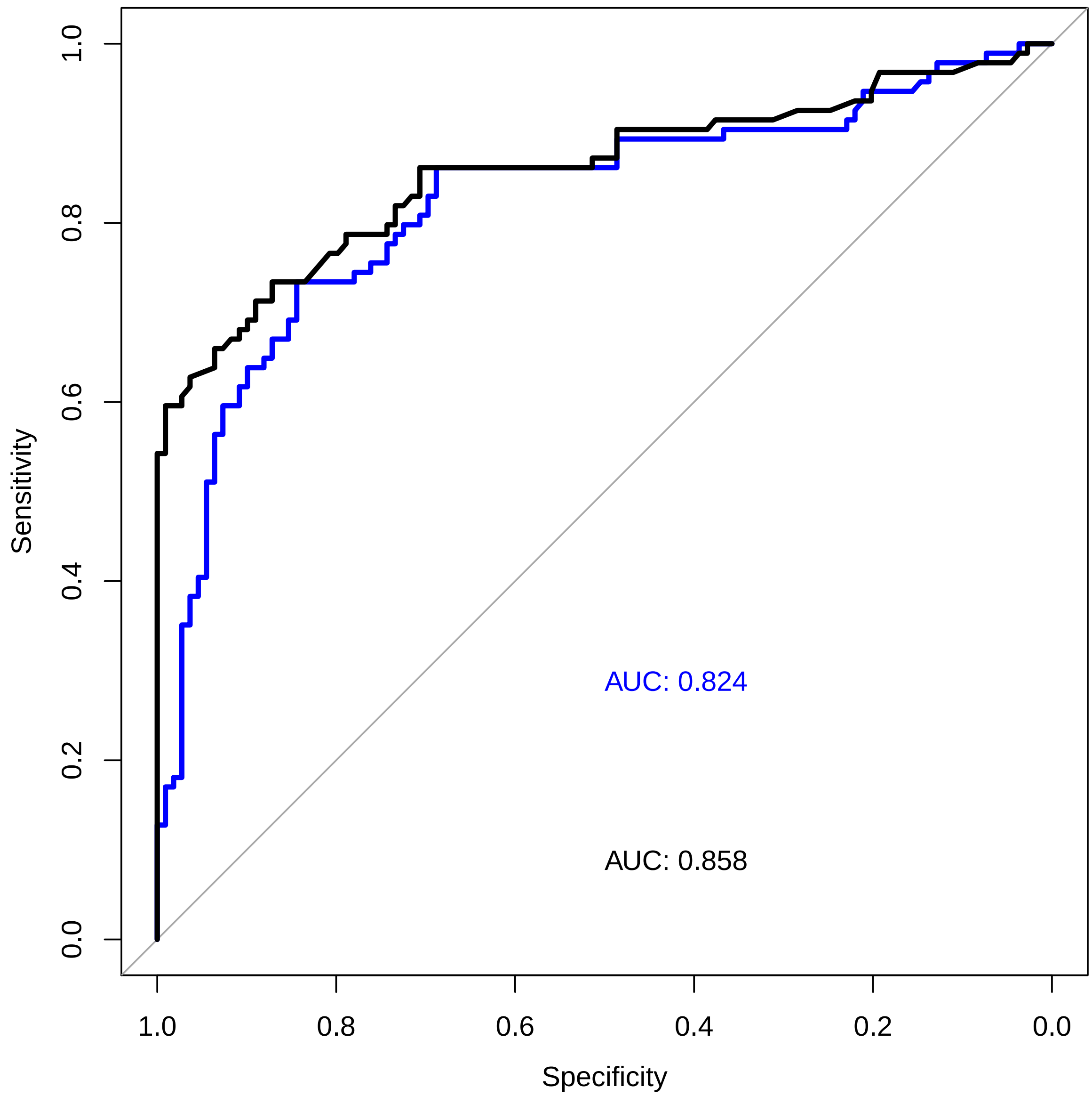



```
##
## Call:
## plot.roc.default(x = titanic[-train, "Survived"], predictor = prediction.naive.logfare[,     1], col
```

```
##
## Data: prediction.naive.logfare[, 1] in 109 controls (titanic[-train, "Survived"] 0) > 94 cases (titar
## Area under the curve: 0.8161
```

Sometimes it helps to **drop** some variables and/or add some new features. For example, in our case, it helps to keep track of the wealthy women (since they have a quite low chance of having died in the Titanic accident). In other words, it helps to create a new feature that equals one only if the passeneger is a wealthyrich woman.

```
Rich_woman = as.factor((titanic$Pclass == 1) & (titanic$Sex == "female") )
titanic.transformed = data.frame("Survived" = titanic$Survived,
                           "Sex" = titanic$Sex,
                           "Age" = titanic$Age,
                           "Pclass"= titanic$Pclass,
                           "RichWoman" = Rich_woman)

#fit a naiveBayes classifier
naive.transformed = naiveBayes(Survived ~ ., titanic.transformed[train,], type="raw")
prediction.naive.transformed = predict(naive.transformed, titanic.transformed[-train,-1],
                                   type = c("raw"))

#ROC + AUC
plot.roc(titanic[-train, "Survived"], prediction.naive[,1], col="blue",
         lwd=3, print.auc=TRUE,print.auc.y = 0.3)

##
## Call:
## plot.roc.default(x = titanic[-train, "Survived"], predictor = prediction.naive[,     1], col = "blue
##
## Data: prediction.naive[, 1] in 109 controls (titanic[-train, "Survived"] 0) > 94 cases (titanic[-tra
## Area under the curve: 0.8235

plot.roc(titanic[-train, "Survived"], prediction.naive.transformed[,1],
         col="black", add=TRUE,
         lwd=3, print.auc=TRUE,print.auc.y = 0.1)
```

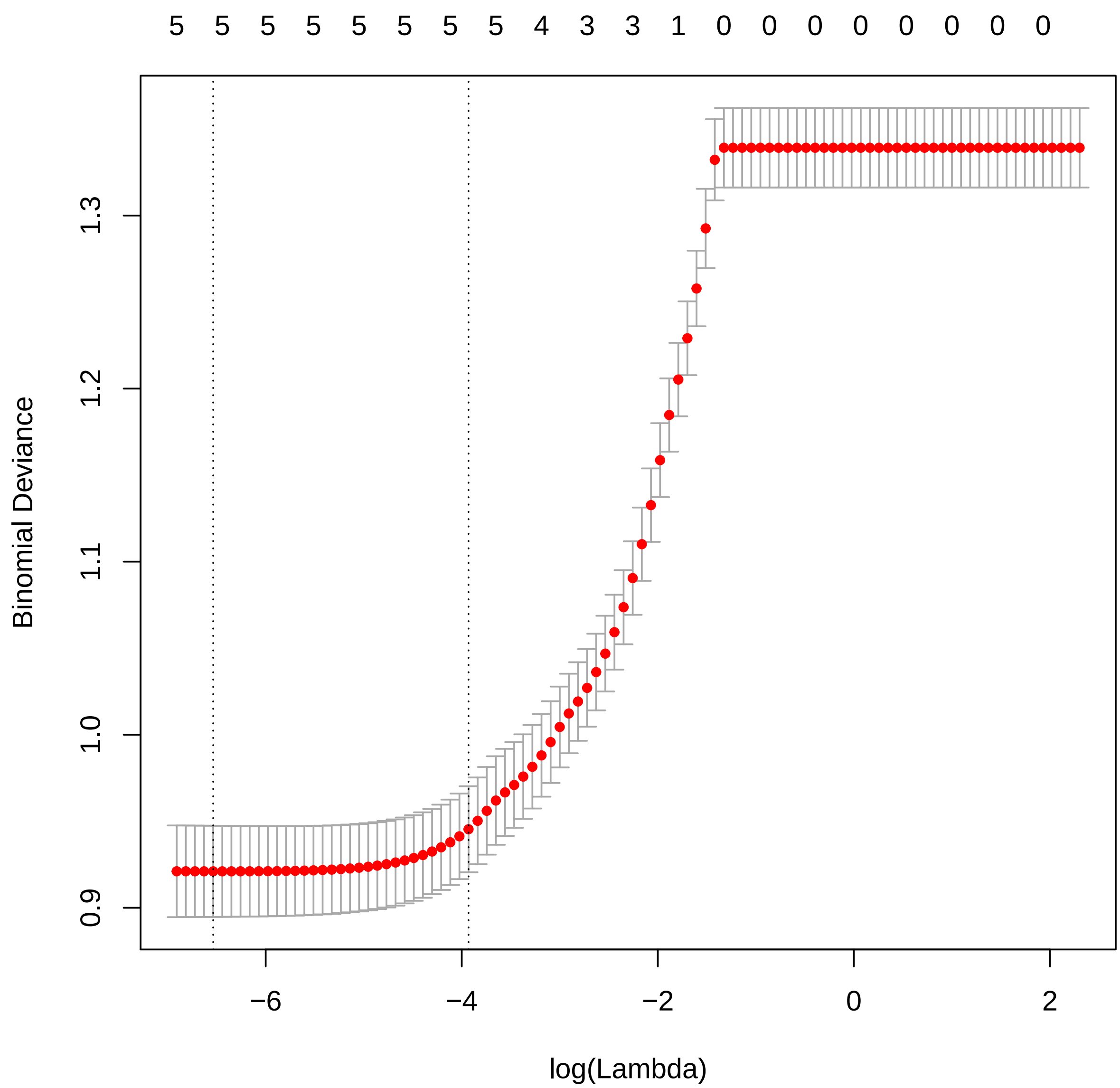



```
##
## Call:
## plot.roc.default(x = titanic[-train, "Survived"], predictor = prediction.naive.transformed[,     1],
##
## Data: prediction.naive.transformed[, 1] in 109 controls (titanic[-train, "Survived"] 0) > 94 cases (t
## Area under the curve: 0.8582
```

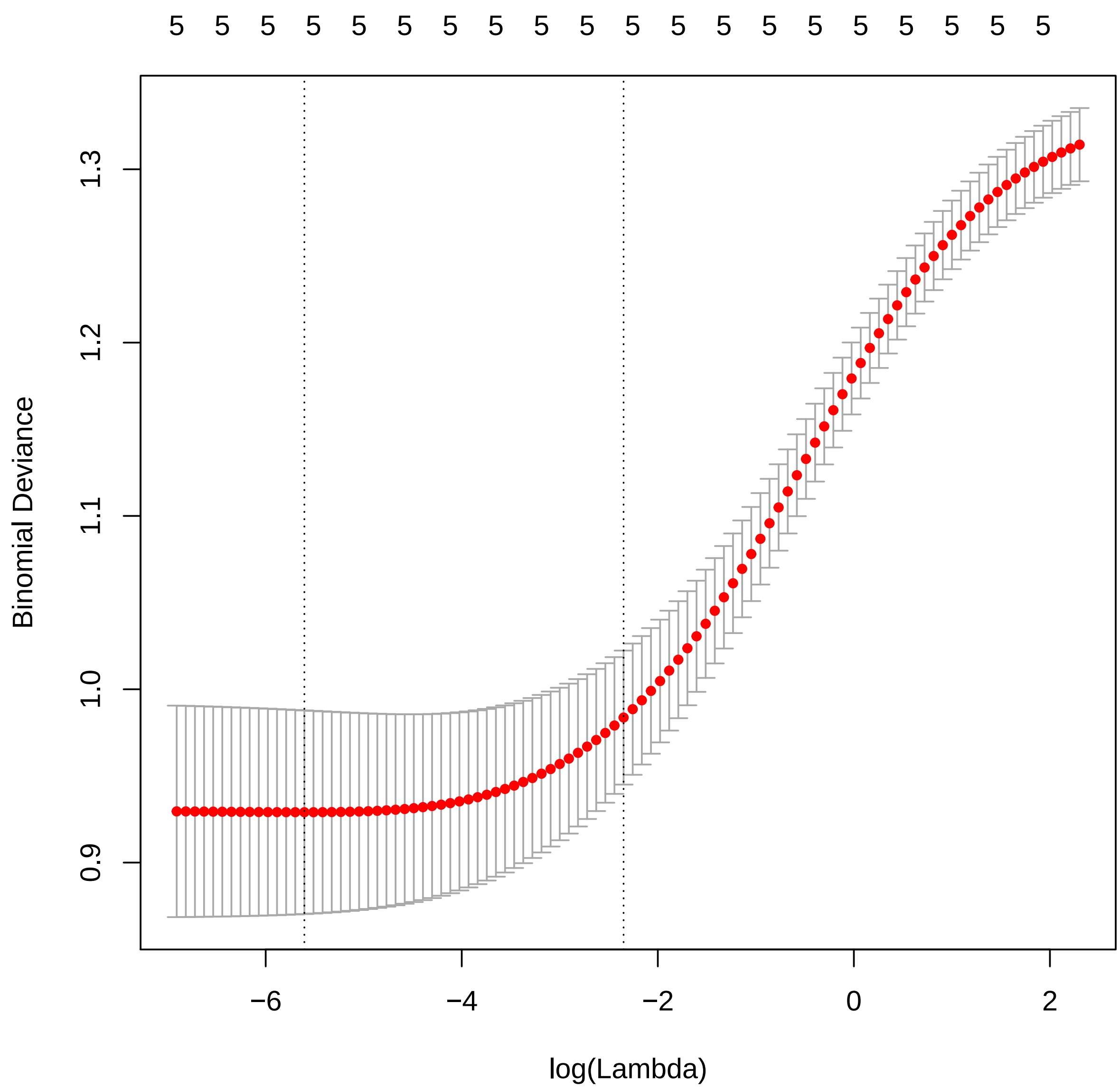
3 Titanic dataset: LASSO and Ridge Logistic Regression

Let us implement a logistic regression on the Titanic dataset.

```
# first, let us extract the X matrix.
# to do so, it is important to note that R needs to know which variable is categorical and
# which variable is real valued. We have already done so when we loaded the data.
X_train_logistic = model.matrix(Survived ~ .,titanic.transformed[train,])
X_test_logistic = model.matrix(Survived ~ .,titanic.transformed[-train,])
Y_train_logistic = as.factor( titanic.transformed[train,"Survived"] )
Y_test_logistic = as.factor( titanic.transformed[-train,"Survived"] )


library(glmnet)

## Warning:  package 'glmnet' was built under R version 3.2.4
## Loading required package:  Matrix
## Warning:  package 'Matrix' was built under R version 3.2.5
## Loading required package:  foreach
## Loaded glmnet 2.0-5
##
## Attaching package:  'glmnet'
## The following object is masked from 'package:pROC':
##
##     auc

lambda.grid = 10^seq(1,-3,length=100)
cvfit.lasso = cv.glmnet(X_train_logistic, Y_train_logistic,
  lambda = lambda.grid, alpha=1, family = "binomial")

plot(cvfit.lasso)
```




```
cvfit.ridge = cv.glmnet(X_train_logistic, Y_train_logistic,
  lambda = lambda.grid, alpha=0, family = "binomial")

plot(cvfit.ridge)
```

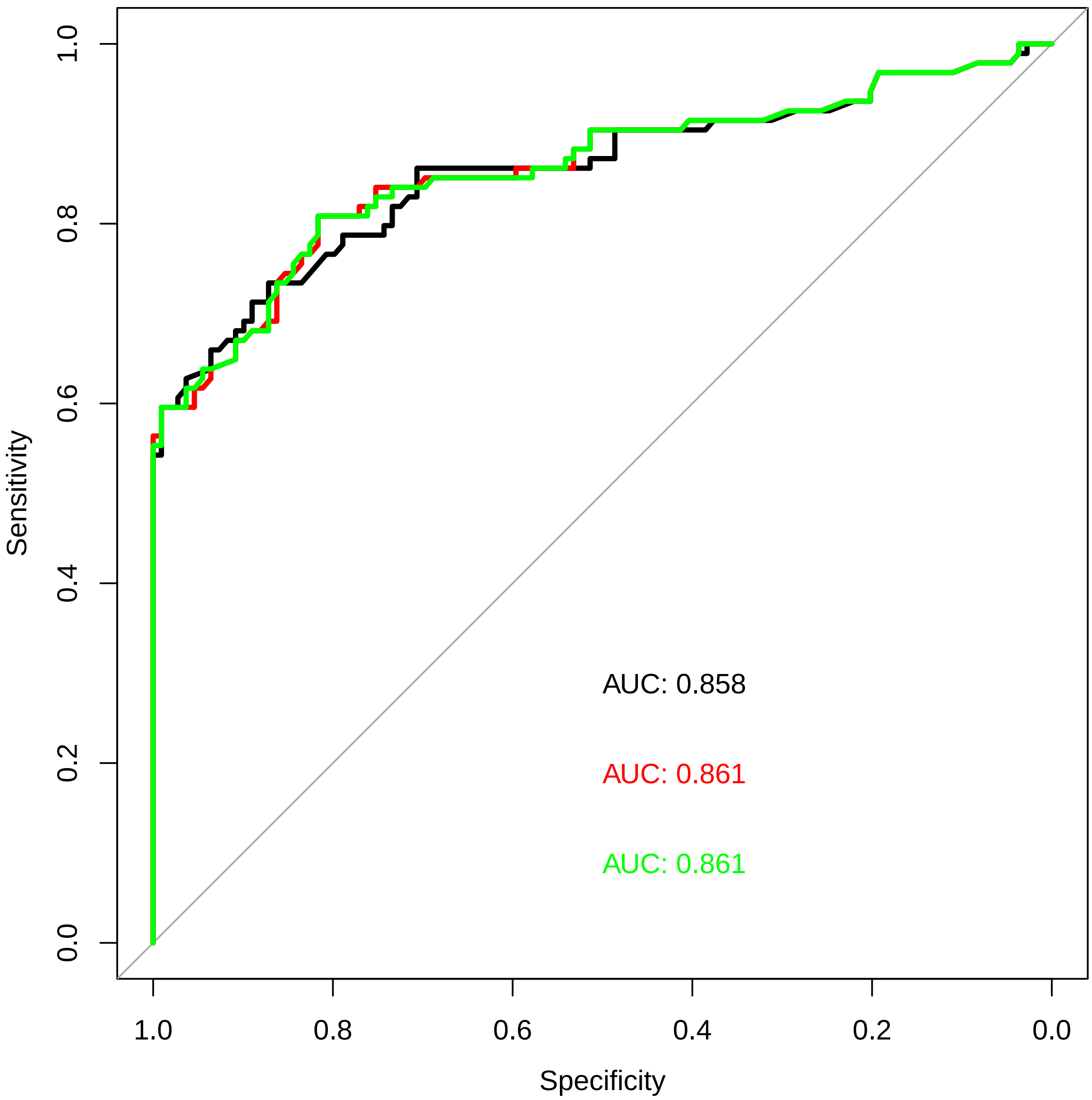



```
prediction.lasso = predict(cvfit.lasso, newx = X_test_logistic,
  s = "lambda.min", type="response")

prediction.ridge = predict(cvfit.ridge, newx = X_test_logistic,
  s = "lambda.min", type="response")
```

Let us display all the results.

```
plot.roc(titanic.transformed[-train, "Survived"], prediction.naive.transformed[,1],
         col="black", lwd=3, print.auc=TRUE, print.auc.y = 0.3)

##
```

```
## Call:
## plot.roc.default(x = titanic.transformed[-train, "Survived"],     predictor = prediction.naive.trans
##
## Data: prediction.naive.transformed[, 1] in 109 controls (titanic.transformed[-train, "Survived"] 0) 
## Area under the curve: 0.8582

plot.roc(titanic.transformed[-train, "Survived"], as.vector(prediction.lasso),
         col="red", lwd=3, print.auc=TRUE, print.auc.y = 0.2, add=TRUE)

##
## Call:
## plot.roc.default(x = titanic.transformed[-train, "Survived"],     predictor = as.vector(prediction.la
##
## Data: as.vector(prediction.lasso) in 109 controls (titanic.transformed[-train, "Survived"] 0) < 94 ca
## Area under the curve: 0.8608

plot.roc(titanic.transformed[-train, "Survived"], as.vector(prediction.ridge),
         col="green", lwd=3, print.auc=TRUE, print.auc.y = 0.1, add=TRUE)
```




```
##
## Call:
## plot.roc.default(x = titanic.transformed[-train, "Survived"],     predictor = as.vector(prediction.ri
##
## Data: as.vector(prediction.ridge) in 109 controls (titanic.transformed[-train, "Survived"] 0) < 94 ca
## Area under the curve: 0.8609
```

For this particular case, one can see that the Naive Bayes classifier is as good as LASSO/Ridge regression.

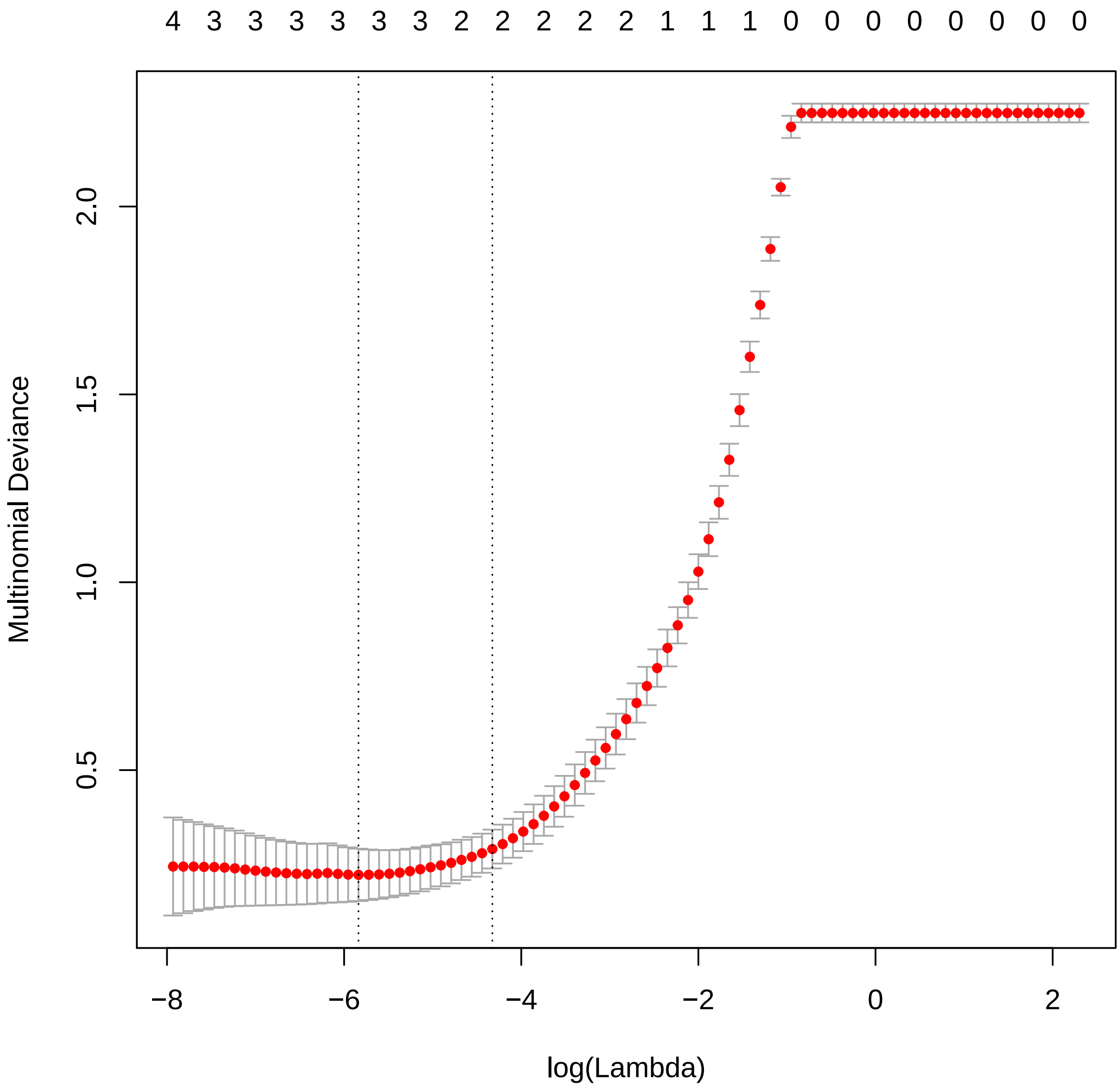
4 Multinomial Logistic Regression a.k.a Softmax Regression

Let us use a multinomial logistic model (also known as softmax regression) to predict the type of a seed based on a few covariates. Data can be downloaded from `https://archive.ics.uci.edu/ml/datasets/seeds`.

```
filename = "seeds.txt"
seeds = read.table(filename, header= TRUE)
seeds$type <- as.factor( seeds$type )

#shuffle and split train/test
#we will use 60% of the data for training
seeds <- seeds[sample(nrow(seeds)),]
train_index = sample(1:nrow(seeds), replace=FALSE, size = round(0.6 * nrow(seeds)))
test_index = (1:nrow(seeds))[-train_index]

X_train = model.matrix(type ~ .,seeds[train_index,])
X_test = model.matrix(type ~ .,seeds[test_index,])
Y_train = as.factor( seeds[train_index,"type"] )
Y_test = as.factor( seeds[test_index,"type"] )

#let us fit a multinomial logistic regression
#with LASSO regularization
lambda.grid = 10^seq(1,-4,length=100)
cvfit.lasso = cv.glmnet(X_train, Y_train,
  lambda = lambda.grid, alpha=1, family = "multinomial")
```

```
## Warning:  from glmnet Fortran code (error code -100); Convergence for 100th lambda value
not reached after maxit=100000 iterations; solutions for larger lambdas returned
## Warning:  from glmnet Fortran code (error code -95); Convergence for 95th lambda value not
reached after maxit=100000 iterations; solutions for larger lambdas returned
## Warning:  from glmnet Fortran code (error code -97); Convergence for 97th lambda value not
reached after maxit=100000 iterations; solutions for larger lambdas returned
## Warning:  from glmnet Fortran code (error code -90); Convergence for 90th lambda value not
reached after maxit=100000 iterations; solutions for larger lambdas returned
## Warning:  from glmnet Fortran code (error code -95); Convergence for 95th lambda value not
reached after maxit=100000 iterations; solutions for larger lambdas returned
## Warning:  from glmnet Fortran code (error code -96); Convergence for 96th lambda value not
reached after maxit=100000 iterations; solutions for larger lambdas returned
```

```
plot(cvfit.lasso)
```




```
#compute probabilistic output
prediction.lasso.probabilistic = predict(cvfit.lasso, newx = X_test,
  s = "lambda.min", type="class")

#compute class prediction
prediction.lasso.class = predict(cvfit.lasso, newx = X_test,
  s = "lambda.min", type="class")

#display a confusion matrix
library(caret)

## Warning:  package 'caret' was built under R version 3.2.5
```

```
## Loading required package:  lattice
## Warning:  package 'lattice' was built under R version 3.2.5
## Loading required package:  ggplot2
## Warning:  package 'ggplot2' was built under R version 3.2.5
```

```
CF = confusionMatrix(prediction.lasso.class, Y_test,
                dnn = c("Prediction", "Reference"))
print(CF$table)
```

```
##           Reference
## Prediction  1  2  3
##          1 25  1  4
##          2  1 33  0
##          3  0  0 20
```

Let us display all the results. @