TP3 MRR Zeyu CHEN & Clément VEYSSIÈRE

Zeyu Chen, Clément Veyssière

11 novembre 2018

Boston Housing dataset

```
set.seed(100)
#First, we get the data and replace medv by the binary variable medvBin
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 3.4.4
```

```
data(BostonHousing)
tab = BostonHousing
medvBin = as.factor(as.numeric(tab$medv > median(tab$medv)))
tab = cbind(tab[,1:13],medvBin)
```

```
res = glm(medvBin ~ ., family = binomial, data = tab)
summary(res)
```

```
##
## Call:
## glm(formula = medvBin ~ ., family = binomial, data = tab)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0237  -0.3635  -0.0039   0.3083   3.2149
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 12.369581   4.003846   3.089 0.002005 **
## crim        -0.061307   0.064570  -0.949 0.342381
## zn           0.017805   0.013617   1.308 0.191034
## indus        0.021489   0.043071   0.499 0.617833
## chas1        1.719603   0.661869   2.598 0.009374 **
## nox         -6.593055   2.715828  -2.428 0.015197 *
## rm           1.535316   0.431688   3.557 0.000376 ***
## age         -0.025074   0.010295  -2.436 0.014871 *
## dis         -0.711635   0.169498  -4.198 2.69e-05 ***
## rad          0.253286   0.061133   4.143 3.42e-05 ***
## tax         -0.009961   0.002961  -3.364 0.000767 ***
## ptratio     -0.539465   0.107420  -5.022 5.11e-07 ***
## b            0.004022   0.002840   1.416 0.156811
## lstat       -0.326600   0.054795  -5.960 2.52e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 701.39  on 505  degrees of freedom
## Residual deviance: 275.83  on 492  degrees of freedom
```

```
## AIC: 303.83
##
## Number of Fisher Scoring iterations: 7
```

```
attributes(res)
```

```
## $names
##  [1] "coefficients"      "residuals"         "fitted.values"
##  [4] "effects"           "R"                 "rank"
##  [7] "qr"                "family"            "linear.predictors"
## [10] "deviance"          "aic"               "null.deviance"
## [13] "iter"              "weights"           "prior.weights"
## [16] "df.residual"       "df.null"           "y"
## [19] "converged"         "boundary"          "model"
## [22] "call"              "formula"           "terms"
## [25] "data"              "offset"            "control"
## [28] "method"            "contrasts"         "xlevels"
##
## $class
## [1] "glm" "lm"
```

```
res$coefficients
```

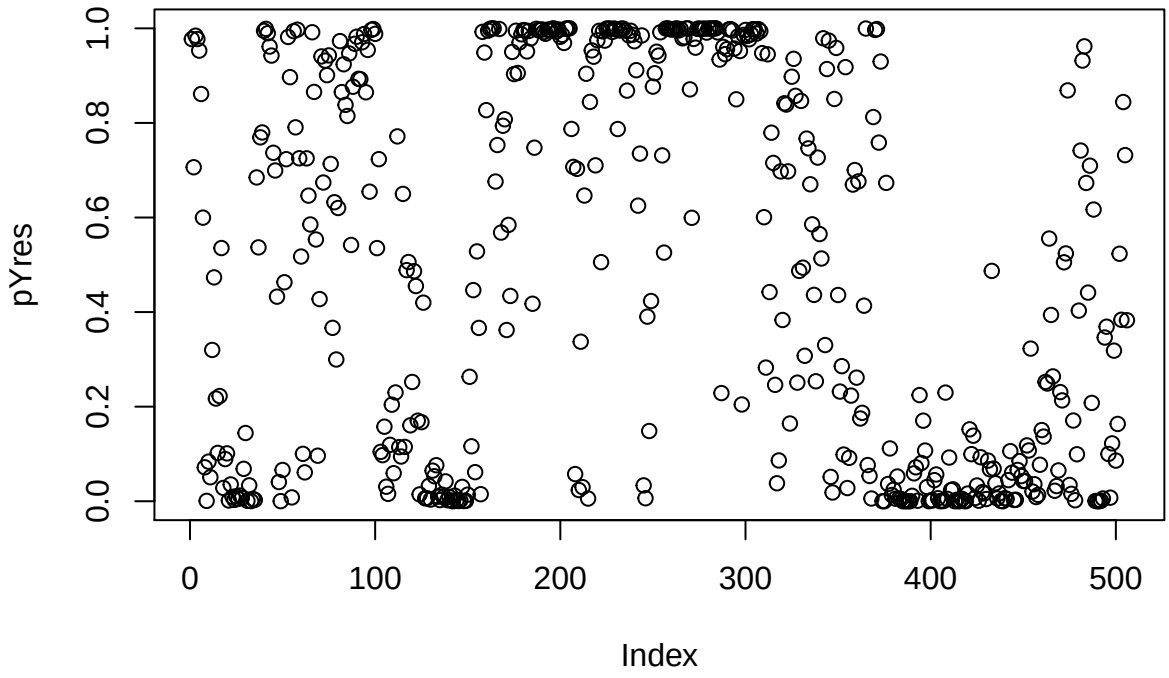
```
##  (Intercept)         crim           zn        indus        chas1
## 12.369581436 -0.061306983  0.017804993  0.021489073  1.719602805
##          nox           rm          age          dis          rad
## -6.593054848  1.535315825 -0.025074393 -0.711634920  0.253286472
##          tax      ptratio            b        lstat
## -0.009961497 -0.539464750  0.004021750 -0.326600236
```

```
#the most significant coefficient seems to be the one of the nox (highest absolute value)
#the less significant coefficient seems to be the one of the b (lowest absolute value)
```

```
pYres = predict.glm(res, type = "response")
pYlink = predict.glm(res, type= "response")
summary(pYres)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000005 0.0567997 0.4913208 0.4940711 0.9510381 0.9999934
```

```
plot(pYres)
```




```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.4.4
```

```
exp(coef(res)) #odds-ratio
```

```
##  (Intercept)         crim           zn        indus        chas1
## 2.355271e+05 9.405345e-01 1.017964e+00 1.021722e+00 5.582311e+00
##          nox           rm          age          dis          rad
## 1.369849e-03 4.642792e+00 9.752374e-01 4.908411e-01 1.288252e+00
##          tax      ptratio            b        lstat
## 9.900880e-01 5.830603e-01 1.004030e+00 7.213721e-01
```

```
coeff = tab[,1:13]
Prob = c(1:506)
coeff = cbind(Intercept = 1, coeff)
for (i in 1:506){
  Prob[i] = (exp(t(coef(res))%*%as.numeric(coeff[i,])))/(1+exp(t(coef(res))%*%as.numeric(coeff[i,])))
}
Prob = as.numeric(Prob >= 0.5)
```

```
#compute confusion matrix and False positive/negative rates
confusion = table(Prob,medvBin)
FP.Rate = confusion[2,1]/(confusion[2,1]+confusion[1,1])
FN.Rate = confusion[1,2]/(confusion[1,2]+confusion[2,2])
print(confusion)
```

```
##     medvBin
```

```
## Prob   0   1
##    0 171  10
##    1  85 240
```

```
cat("False positive rate =", FP.Rate, "\n" )
```

```
## False positive rate = 0.3320312
```

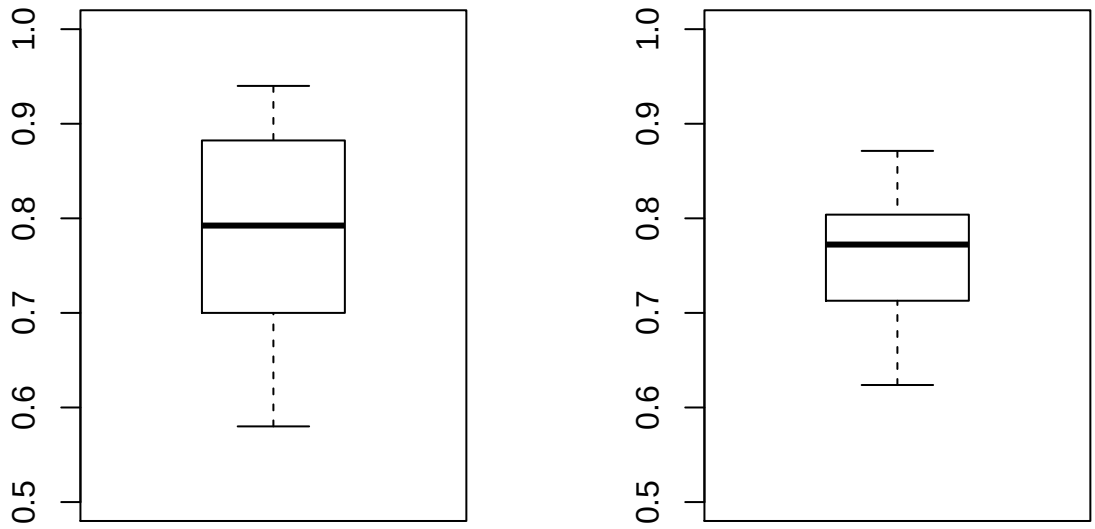
```
cat("False negative rate =", FN.Rate, "\n" )
```

```
## False negative rate = 0.04
```

K folds

```
kfold <- function(k)
{
  performance <- vector(length = k)
  #Create 10 equally size folds
  folds <- cut(seq(1,nrow(tab)),breaks=k,labels=FALSE)

  #Perform 10 fold cross validation
  for(i in 1:k){
    #Split data by fold using the which() function

    testIndexes <- which(folds==i,arr.ind=TRUE)
    testData <- tab[testIndexes,]
    trainData <- tab[-testIndexes,]

    #Use the test and train data partitions
    res <- glm(medvBin ~ .,family = binomial,data = trainData)
    testData$medvBin = NULL

    xNew <- cbind(1,testData)
    for (j in 1:14){
      xNew[,j] = as.numeric(xNew[,j])
    }

    nChap <- 1/(1+exp(- (as.matrix(xNew))%*%as.numeric(res$coefficients)))#erreur ici
    yChap<- as.numeric(nChap>0.5)
    t <- table(yChap,y=tab[testIndexes,]$medvBin)
    if(sum(yChap)/length(yChap) == 1){
      st = t[1,2]
    }
    else if (sum(yChap) == 0){
      st = t[1,1]
    }
    else{
      st = t[1,1] + t[2,2]
    }
    performance[i]<-(st)/nrow(testData)
  }
  boxplot(performance,ylim=c(0.5,1))
}
par(mfrow=c(1,2))
kfold(10)
```

```
kfold(5)
```



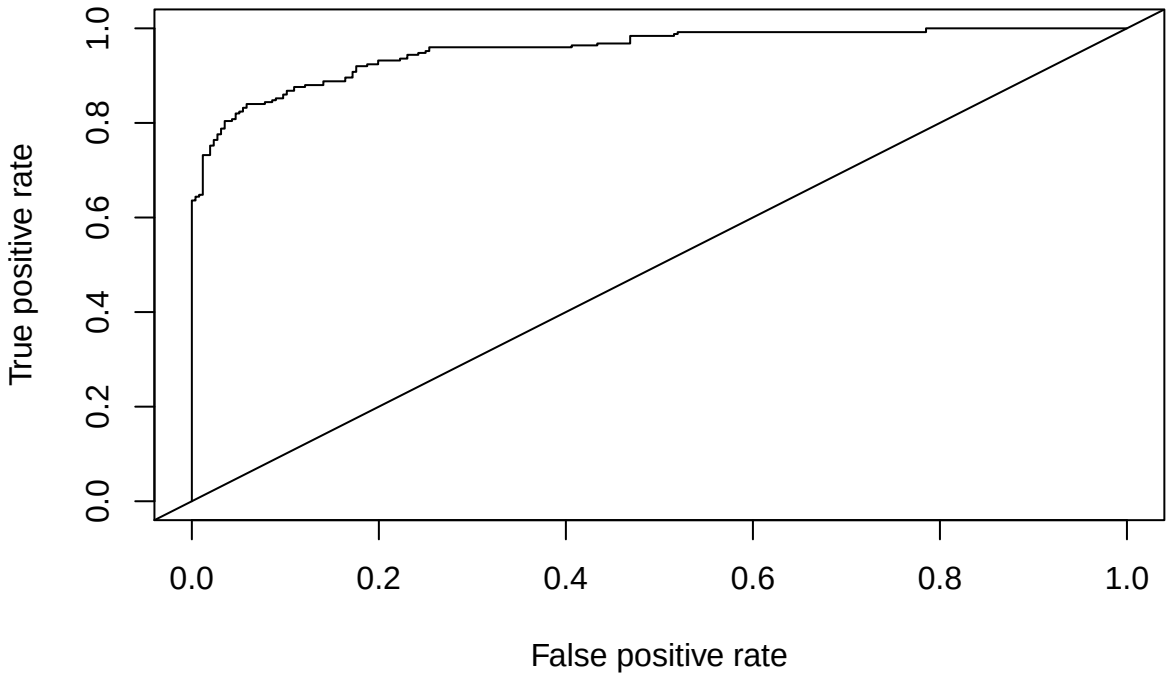
The plot on the left represents the result of the K-fold procedure for K=10 while the one on the right is for K=5

ROC curve

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.4.4
## Loading required package: gplots
## Warning: package 'gplots' was built under R version 3.4.4
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
##
##     lowess
```

```
prelink <- predict.glm(res,newdata = tab,type = "link")
pre <- prediction(prelink,tab$medvBin)

#threshold varie de 1 à 0
plot(performance(pre,"tpr","fpr"))
abline(0,1)
```



Model selection

1. Statistical approach: forward, backward, stepwise selection

a)

```
print(resfor)
```

```
##
## Call:  glm(formula = medvBin ~ lstat + ptratio + rm + chas + dis + age +
##     nox + rad + tax + b, family = binomial, data = tab)
##
## Coefficients:
## (Intercept)        lstat      ptratio           rm        chas1
##   12.455435    -0.334918    -0.567756     1.467629     1.796811
##         dis          age          nox          rad          tax
##   -0.586698    -0.023940    -6.074136     0.227716    -0.009131
##           b
##    0.004603
##
## Degrees of Freedom: 505 Total (i.e. Null);  495 Residual
## Null Deviance:       701.4
## Residual Deviance: 279.6     AIC: 301.6
```

```
print(resback)
```

```
##
## Call:  glm(formula = medvBin ~ chas + nox + rm + age + dis + rad + tax +
```

```
##     ptratio + b + lstat, family = binomial, data = tab)
##
## Coefficients:
## (Intercept)        chas1          nox           rm          age
##   12.455435     1.796811    -6.074136     1.467629    -0.023940
##         dis          rad          tax      ptratio            b
##   -0.586698     0.227716    -0.009131    -0.567756     0.004603
##       lstat
##   -0.334918
##
## Degrees of Freedom: 505 Total (i.e. Null);  495 Residual
## Null Deviance:       701.4
## Residual Deviance: 279.6     AIC: 301.6
```

```
print(resstep)
```

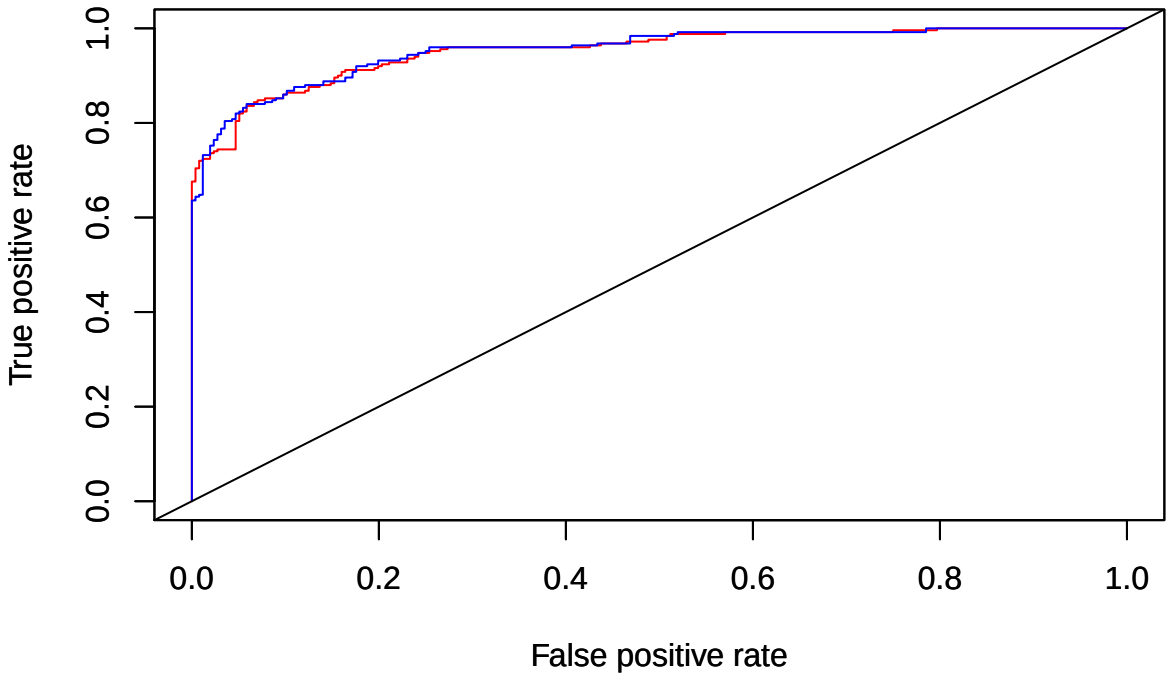
```
##
## Call:  glm(formula = medvBin ~ chas + nox + rm + age + dis + rad + tax +
##     ptratio + b + lstat, family = binomial, data = tab)
##
## Coefficients:
## (Intercept)        chas1          nox           rm          age
##   12.455435     1.796811    -6.074136     1.467629    -0.023940
##         dis          rad          tax      ptratio            b
##   -0.586698     0.227716    -0.009131    -0.567756     0.004603
##       lstat
##   -0.334918
##
## Degrees of Freedom: 505 Total (i.e. Null);  495 Residual
## Null Deviance:       701.4
## Residual Deviance: 279.6     AIC: 301.6
```

```
formula(resfor)
```

```
## medvBin ~ lstat + ptratio + rm + chas + dis + age + nox + rad +
##     tax + b
```

b)

```
prelink = predict.glm(resfor, newdata = tab, type = "link")
pre = prediction(prelink, tab$medvBin)
pre2 = prediction(pYlink, tab$medvBin)
plot(performance(pre,"tpr","fpr"), col = "red")
par(new = TRUE)
plot(performance(pre2,"tpr","fpr"), col = "blue")
abline(0,1)
```




```
#The two curves are very similar
```

2. Logistic regression with l1 or l2 penalizations

```
medvBin <- as.numeric(BostonHousing$medv>median(BostonHousing$medv))
BostonHousing$medv <- medvBin

# partitionning
sub <- sample(nrow(BostonHousing), 0.8 * nrow(BostonHousing))
tabTrain <- BostonHousing[sub,]
tabTest <- BostonHousing[-sub,]
xtrain = data.matrix(tabTrain[,1:13])
ytrain = as.factor(tabTrain$medv)
xtest = data.matrix(tabTest[,1:13])
ytest = as.factor(tabTest$medv)
```

a)Ridge Regression

```
library(glmnet)
```

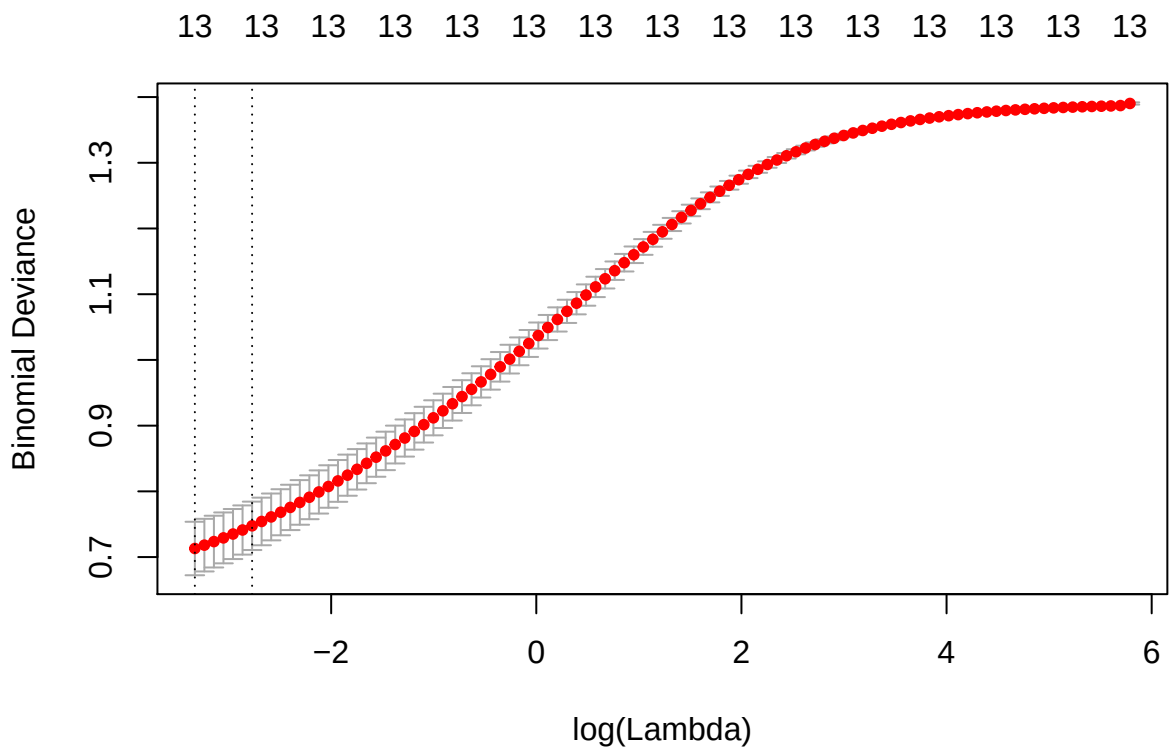
```
## Warning: package 'glmnet' was built under R version 3.4.4
## Loading required package: Matrix
## Loading required package: foreach
## Warning: package 'foreach' was built under R version 3.4.4
```

```
## Loaded glmnet 2.0-16
```

```
ridge = glmnet(xtrain,ytrain,alpha=0,family = "binomial")
summary(ridge)
```

```
##            Length Class     Mode
## a0          100   -none-    numeric
## beta       1300   dgCMatrix S4
## df          100   -none-    numeric
## dim           2   -none-    numeric
## lambda      100   -none-    numeric
## dev.ratio   100   -none-    numeric
## nulldev       1   -none-    numeric
## npasses       1   -none-    numeric
## jerr          1   -none-    numeric
## offset        1   -none-    logical
## classnames    2   -none-    character
## call          5   -none-    call
## nobs          1   -none-    numeric
```

```
#10-folds cross-validation with ridge model
cv_ridge = cv.glmnet(xtrain, ytrain, family = "binomial", nfolds = 10, alpha = 0)
plot(cv_ridge)
```




```
lambda.min = cv_ridge$lambda.min
lambda.1se = cv_ridge$lambda.1se
#we get the two models ridge.1se and ridge.min using respectively lambda.1se and lambda.min
ridge.1se = glmnet(xtrain, ytrain, alpha = 0, family = "binomial", lambda = lambda.1se)
```

```r
ridge.min = glmnet(xtrain, ytrain, alpha = 0, family = "binomial", lambda = lambda.min)
```

```r
#we predict the target value of the test dataset using both models
y_pred.1se = predict(ridge.1se, newx = xtest, type = "response")
y_pred.min = predict(ridge.min, newx = xtest, type = "response")
y_pred.1se[y_pred.1se >= 0.5] = 1
y_pred.1se[y_pred.1se != 1] = 0
y_pred.min[y_pred.min >= 0.5] = 1
y_pred.min[y_pred.min != 1] = 0
#we get the confusion matrix for each model
confusion.1se = table(y_pred.1se,ytest)
print(confusion.1se)
```

```
##           ytest
## y_pred.1se  0  1
##          0 47  5
##          1  7 43
```

```r
FP.Rate.1se = confusion.1se[2,1]/(confusion.1se[2,1]+confusion.1se[1,1])
FN.Rate.1se = confusion.1se[1,2]/(confusion.1se[1,2]+confusion.1se[2,2])
cat("False positive rate =", FP.Rate.1se, "\n" )
```

```
## False positive rate = 0.1296296
```

```r
cat("False negative rate =", FN.Rate.1se, "\n" )
```

```
## False negative rate = 0.1041667
```

```r
confusion.min = table(y_pred.min,ytest)
print(confusion.min)
```

```
##           ytest
## y_pred.min  0  1
##          0 48  5
##          1  6 43
```

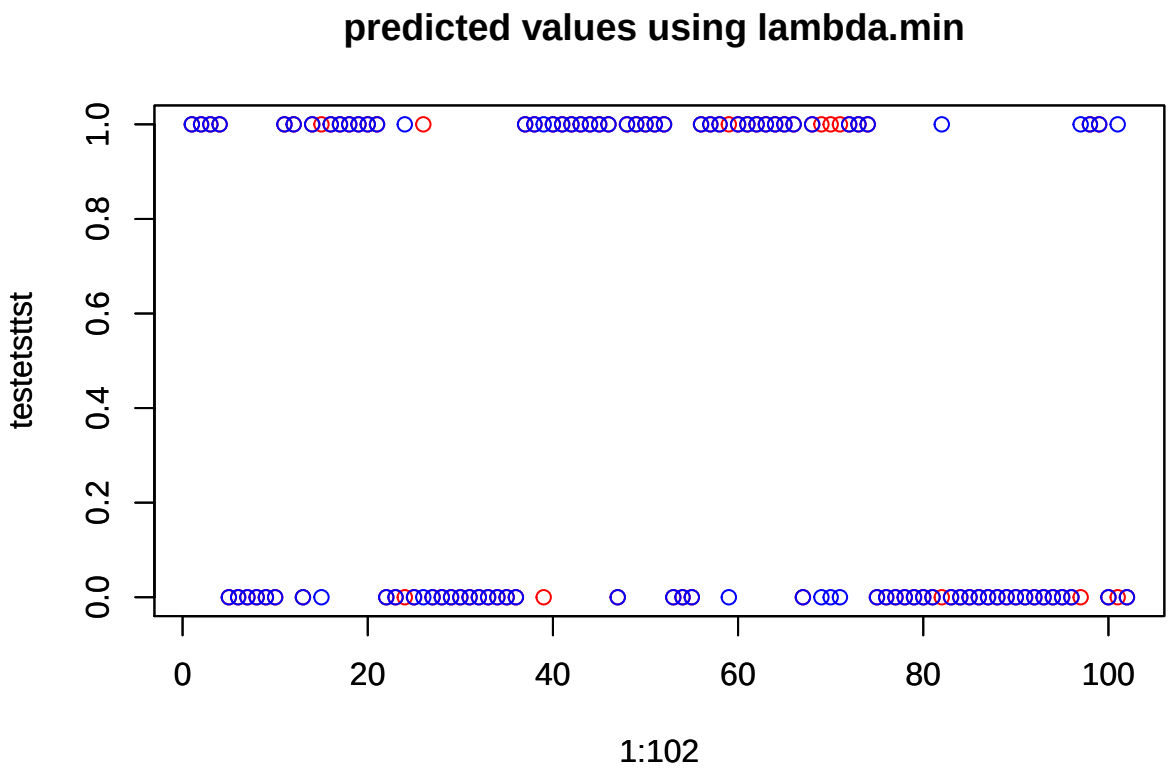
```r
FP.Rate.min = confusion.min[2,1]/(confusion.min[2,1]+confusion.min[1,1])
FN.Rate.min = confusion.min[1,2]/(confusion.min[1,2]+confusion.min[2,2])
cat("False positive rate =", FP.Rate.min, "\n" )
```

```
## False positive rate = 0.1111111
```

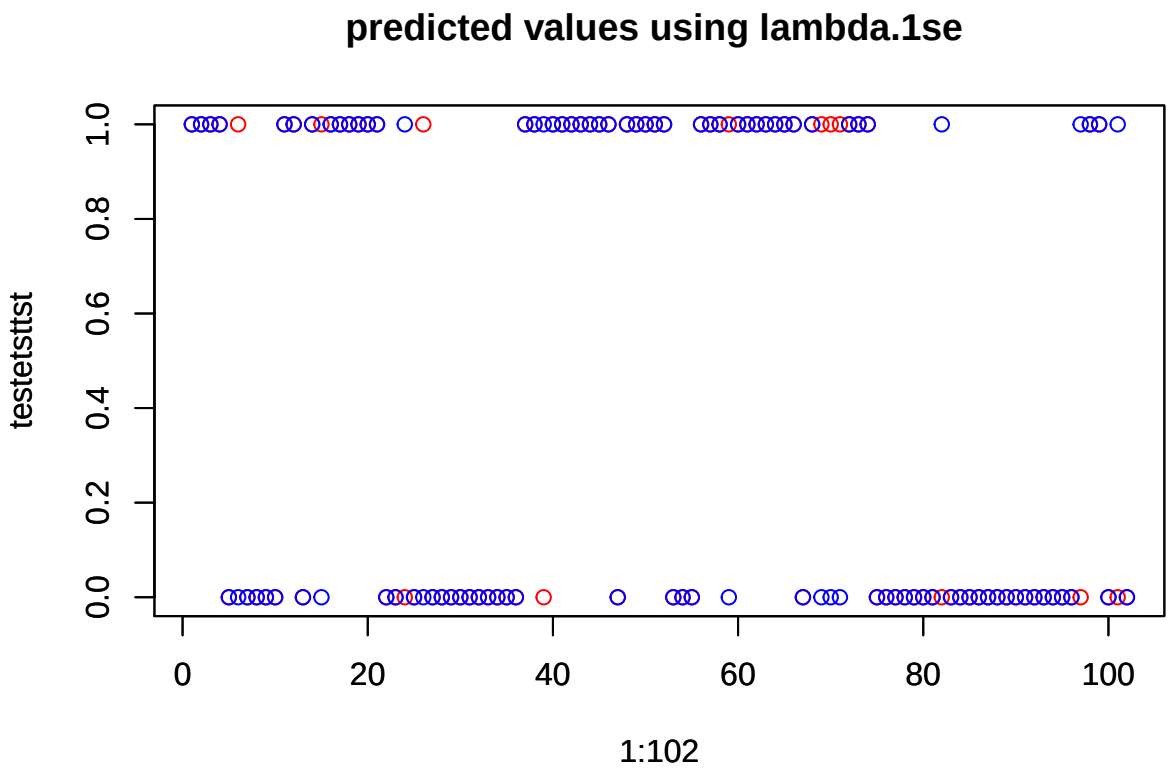
```r
cat("False negative rate =", FN.Rate.min, "\n" )
```

```
## False negative rate = 0.1041667
```

```r
ytest = as.numeric(ytest)-1
plot(1:102,y_pred.min, col = "red", ylim = c(0,1), ylab = "testetsttst", main = "predicted values using
par(new = TRUE)
plot(1:102,ytest, col = "blue", ylim = c(0,1), ylab = "testetsttst")
```



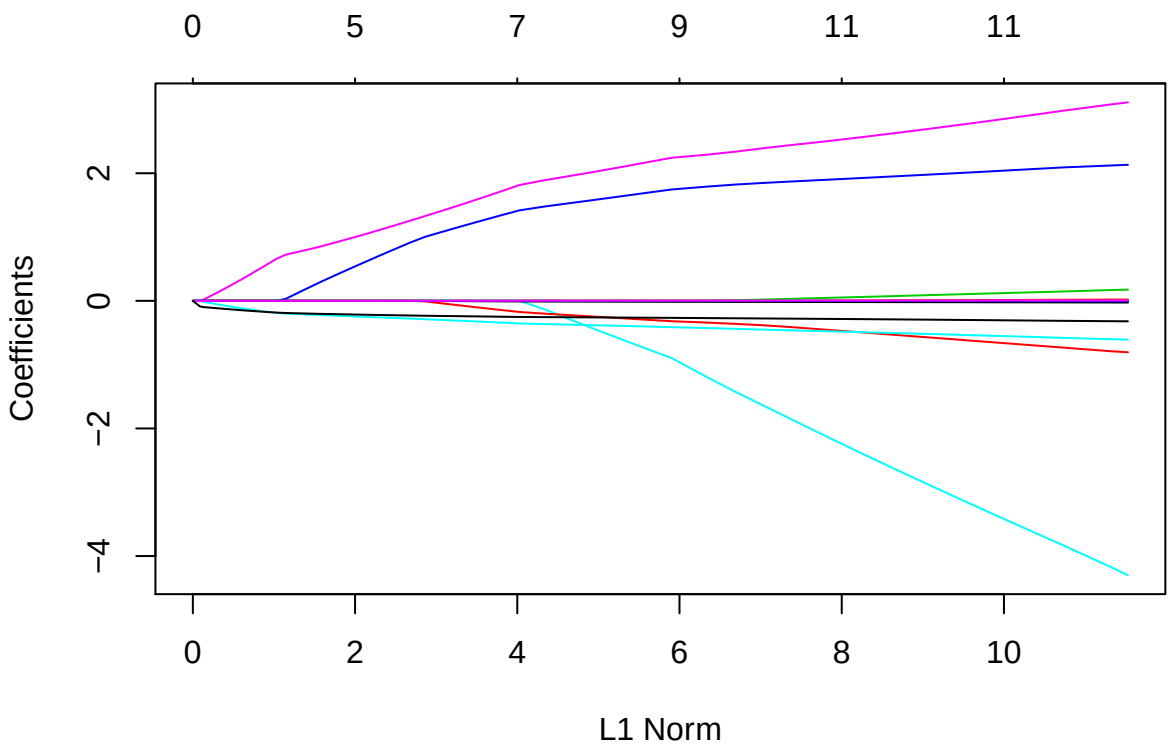

```
plot(1:102,y_pred.1se, col = "red", ylim = c(0,1), ylab = "testetsttst", main = "predicted values using
par(new=TRUE)
plot(1:102,ytest, col = "blue", ylim = c(0,1), ylab = "testetsttst")
```



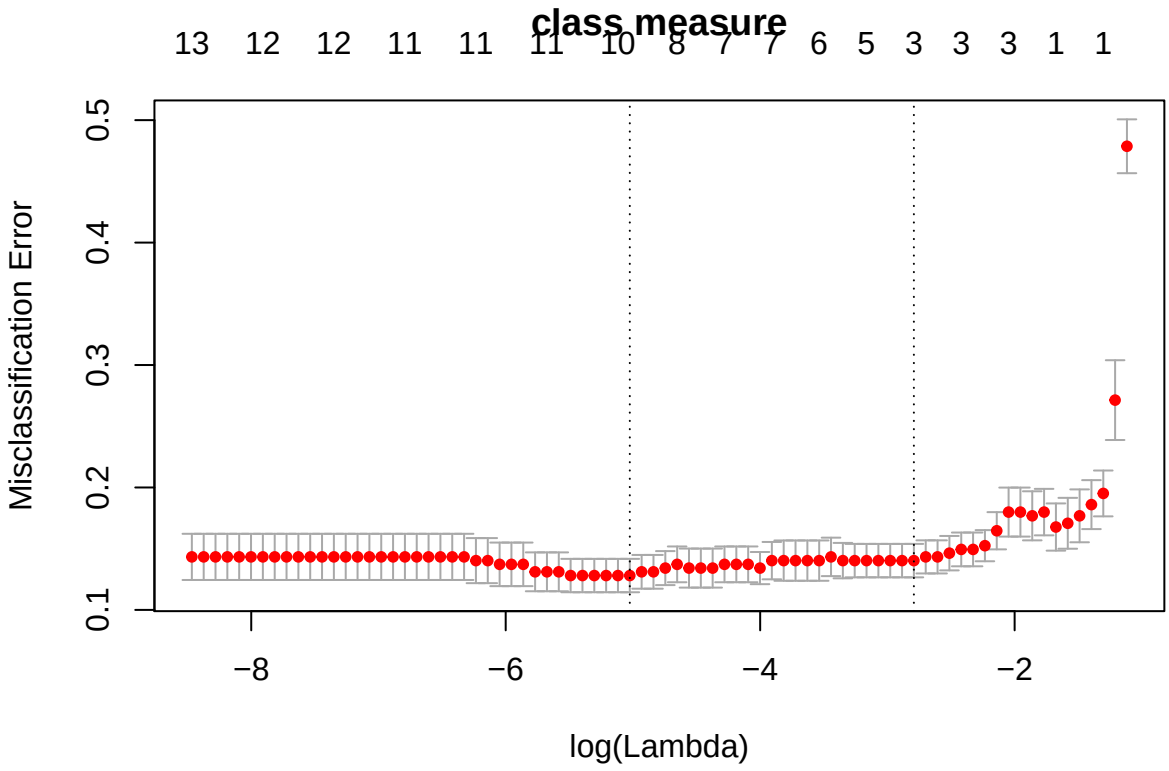
The red point correspond to the actual values that weren't correctly predicted. We can notice that both lambda gave quite similar predictions for this test with a slight advantage for the model using lambda.min. The false positive rate is about 3 times lower than with the first method. So, this is a much better approach.

b) Lasso regression

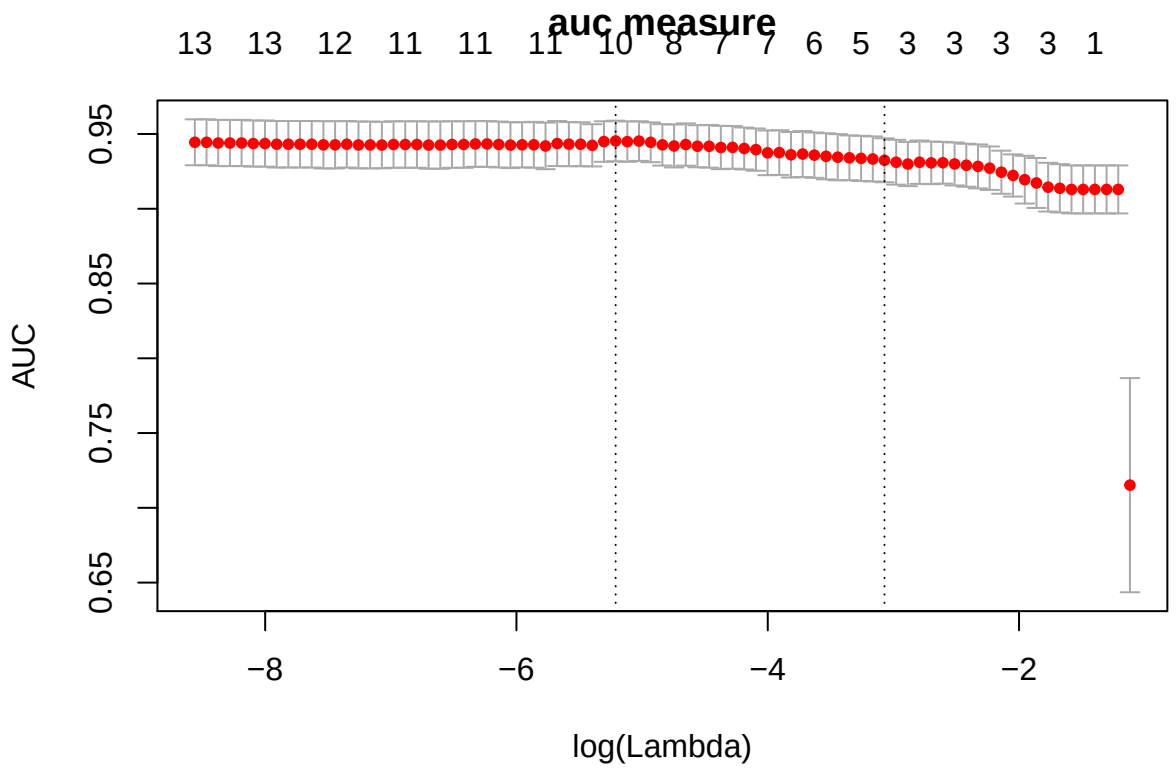
```
# partitionning
sub <- sample(nrow(BostonHousing), 0.65 * nrow(BostonHousing))
tabTrain <- BostonHousing[sub,]
tabTest <- BostonHousing[-sub,]

#drop the medv and transforme to matrix
X <- data.matrix(subset(tabTrain,select= -medv))


#Using lasso regression with alpha = 1
glmmod <- glmnet(X,as.factor(tabTrain$medv),alpha = 1,family="binomial")
plot(glmmod)
```



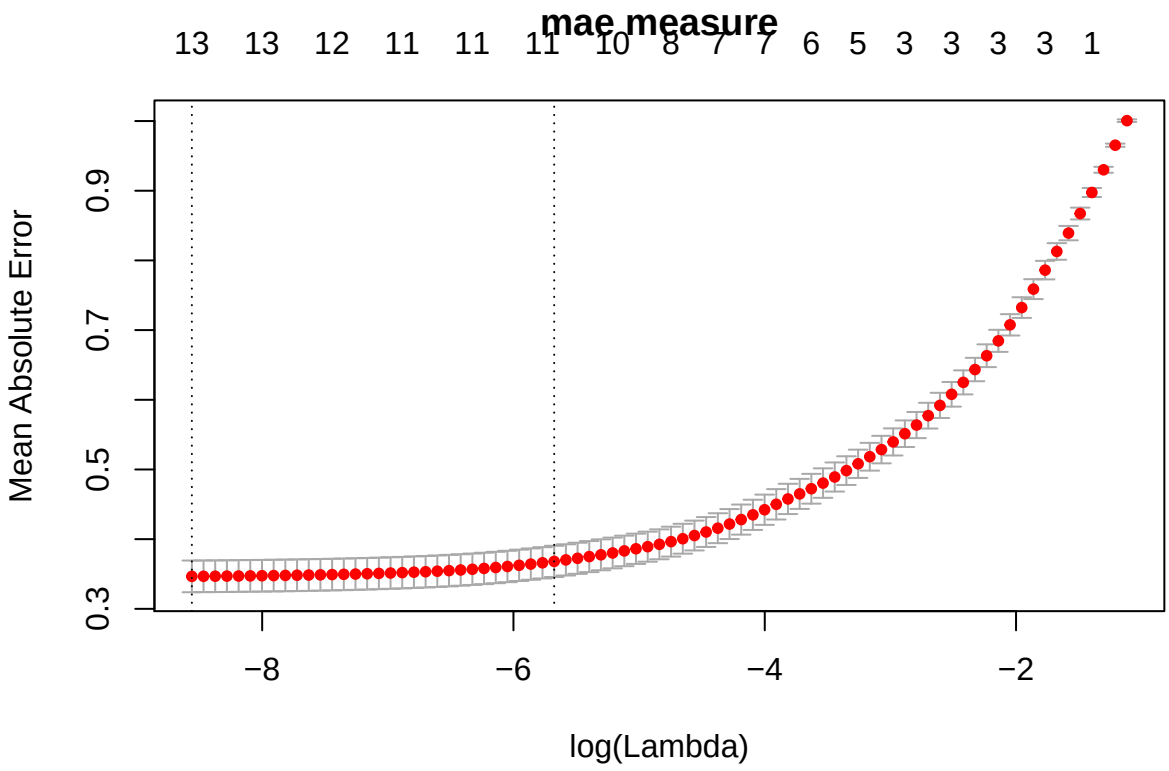

```
#We can find the samller L1 norm is , the more coef are equal to 0

#Using lasso regression through 10-fold with type.measure="class"
modLassoC<- cv.glmnet(X,tabTrain$medv,family="binomial",type.measure="class",alpha=1)
plot(modLassoC,main="class measure")
```




```
#Using lasso regression through 10-fold with type.measure="auc"
modLassoA<- cv.glmnet(X,tabTrain$medv,family="binomial",type.measure="auc",alpha=1)
plot(modLassoA,main="auc measure")
```




```
#Using lasso regression through 10-fold with type.measure="mae"
modLassoM<- cv.glmnet(X,tabTrain$medv,family="binomial",type.measure="mae",alpha=1)
plot(modLassoM,main="mae measure")
```




```
lambda.min<-vector(length = 3)
lambda.1se<-vector(length = 3)
newx <- data.matrix(subset(tabTest,select=-medv))
#Predicting with ="modLassoC" and with "s=modLassoC$lambda.min"
preMin<- predict(modLassoC,newx =newx,s=modLassoC$lambda.min,type = "response")
lambda.min[1] <- sum((as.numeric(preMin>0.5)-tabTest$medv)^2)

#Predicting with ="modLassoC" and with "s=modLassoC$lambda.1se"
pre1se<- predict(modLassoC,newx =newx,s=modLassoC$lambda.1se,type = "response")
lambda.1se[1] <- sum((as.numeric(pre1se>0.5)-tabTest$medv)^2)

#Predicting with ="modLassoA" and with "s=modLassoA$lambda.min"
preMin<- predict(modLassoA,newx =newx,s=modLassoA$lambda.min,type = "response")
lambda.min[2] <- sum((as.numeric(preMin>0.5)-tabTest$medv)^2)

#Predicting with ="modLassoA" and with "s=modLassoA$lambda.1se"
pre1se<- predict(modLassoA,newx =newx,s=modLassoA$lambda.1se,type = "response")
lambda.1se[2] <- sum((as.numeric(pre1se>0.5)-tabTest$medv)^2)

#Predicting with ="modLassoM" and with "s=modLassoM$lambda.min"
preMin<- predict(modLassoM,newx =newx,s=modLassoM$lambda.min,type = "response")
lambda.min[3] <- sum((as.numeric(preMin>0.5)-tabTest$medv)^2)

#Predicting with ="modLassoM" and with "s=modLassoM$lambda.1se"
pre1se<- predict(modLassoM,newx =newx,s=modLassoM$lambda.1se,type = "response")
lambda.1se[3] <- sum((as.numeric(pre1se>0.5)-tabTest$medv)^2)
```

```r
data.frame(lambda.min,lambda.1se,row.names = c("Class wrong times ","Auc wrong times ",
                                               "Mae wrong times "))
```

```
##                    lambda.min lambda.1se
## Class wrong times          20         26
## Auc wrong times            20         25
## Mae wrong times            20         21
```

```r
# We can find that the performance of Mae methods is quite good. And the performence using
# lambda.min is better than using lambda.1se.
```