Chapter 5: Tree methods

a.h.thiery@nus.edu.sg

Version: 0.1

Contents

1 Basic trees: Titanic dataset

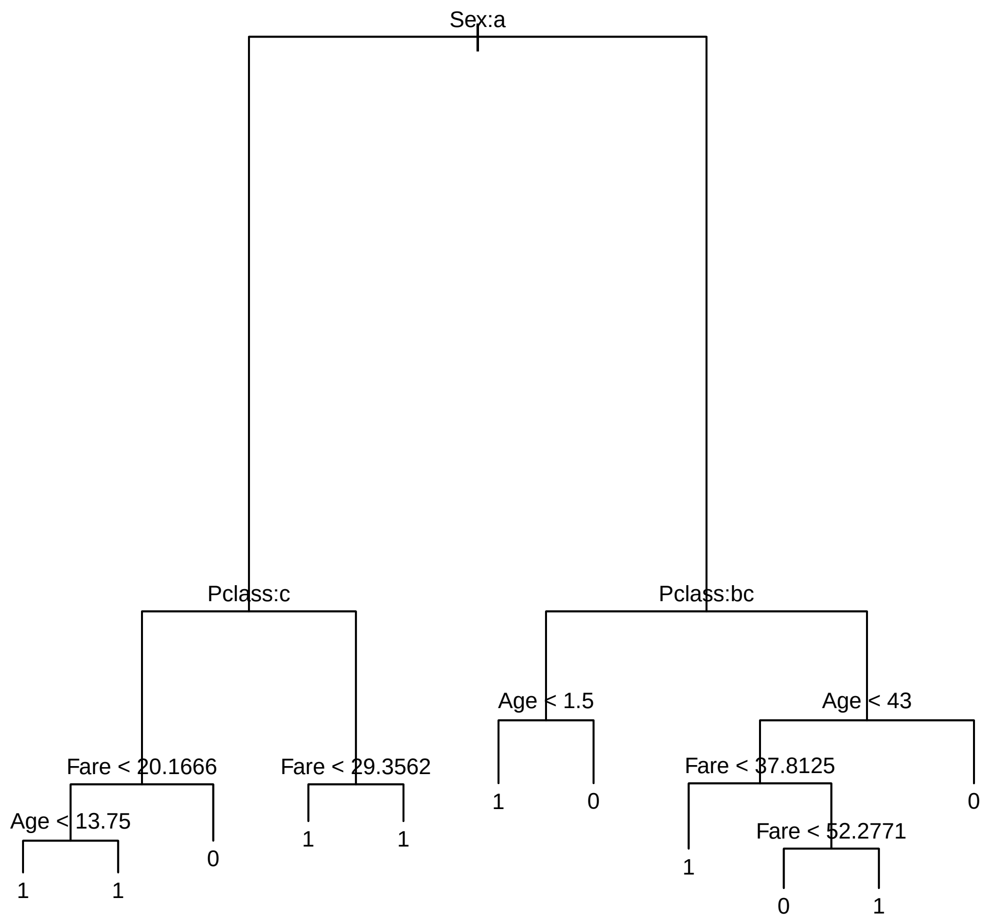
Let us play with the *Titanic* dataset and see the performances of basic trees. First, let us load the data, clean it up a little bit, and split it into a training and test set. Only the training set (even in the cross-validation part, if there is one) is used in the training phase; the test-set is only used to evaluate the performances of the algorithm. It is important not to use the test-set during the cross-validation part!

```
#<<>>=
set.seed(1)
#load covariates
#filename = "/home/alek_thiery/Dropbox/teaching/ST4240/chap6_slides/code/Titanic.csv"
#filename = "/Users/nus/Dropbox/teaching/ST4240/chap6_slides/code/Titanic.csv"
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
```

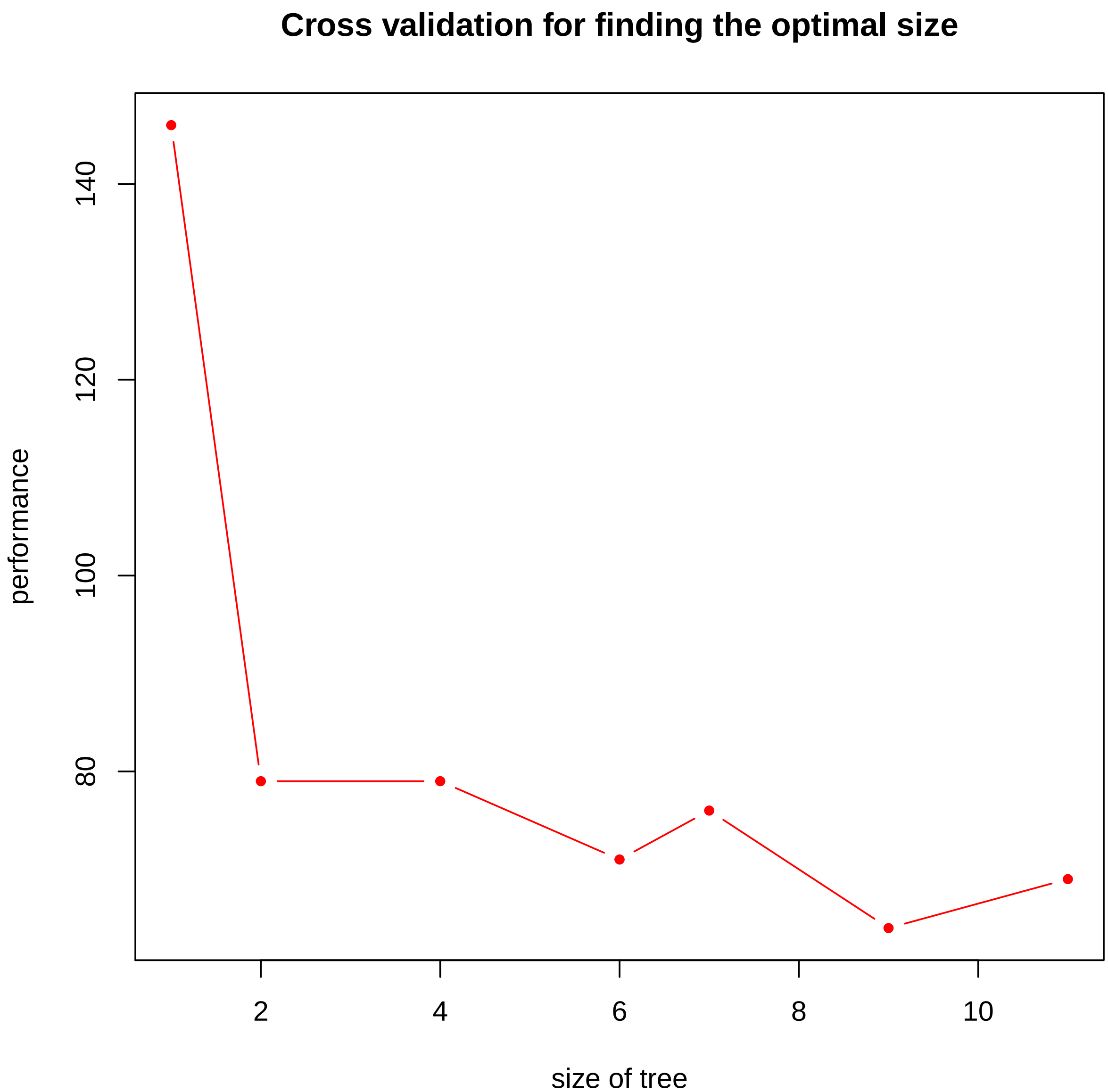
```
## 2        1      1 female  38     1     0 71.2833        C
## 3        1      3 female  26     0     0  7.9250        S
## 4        1      1 female  35     1     0 53.1000        S
## 5        0      3   male  35     0     0  8.0500        S
## 6        0      3   male  NA     0     0  8.4583        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,]


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

#split thedataset into a training and test set
n_total = nrow(titanic)
n_train = round(n_total / 2)
n_test = n_total - n_train
titanic.train = sample(1:n_total, n_train)
```

Now, let us grow a simple tree on the training set. The pruning part will be done by cross-validation.

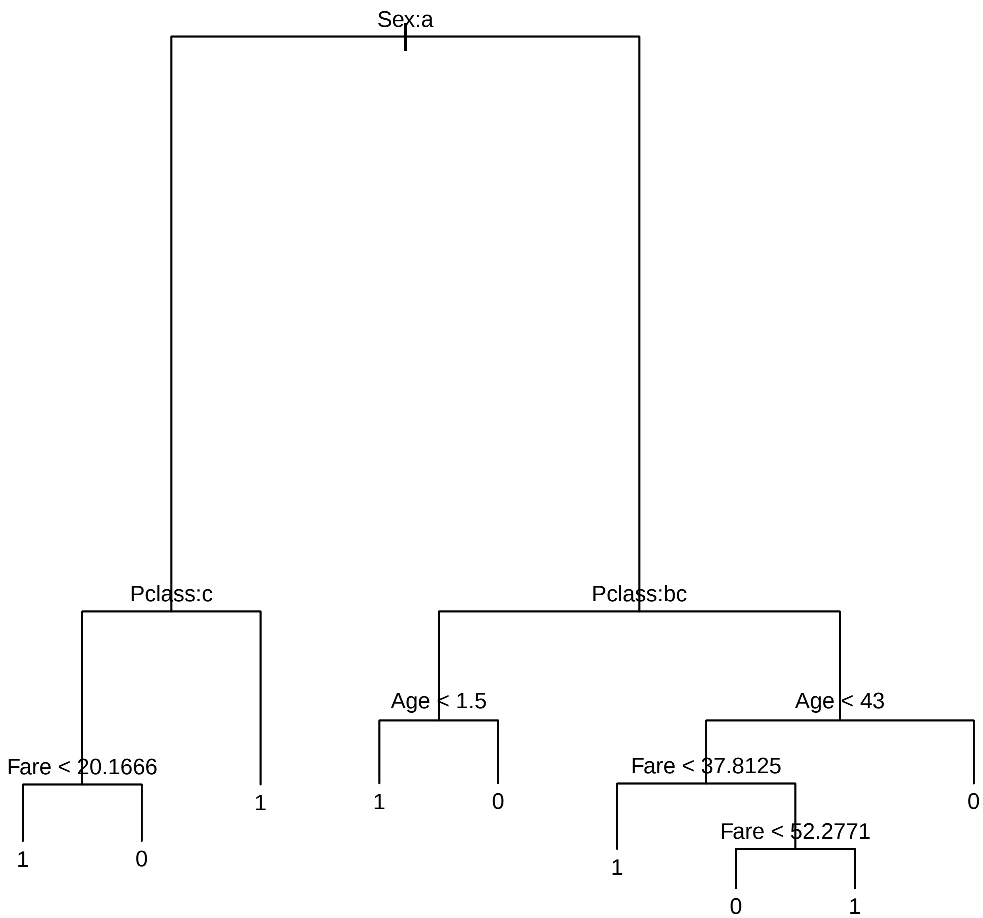
```
library(tree)
#fit a tree
tree.titanic = tree(Survived ~ ., data=titanic[titanic.train,])
plot(tree.titanic)
text(tree.titanic, cex=0.75)
```




```r
#let us see if the trees needs pruning
#we use the argument "FUN = prune.misclass" because we are doing classification
#this argument is not needed for classification
cv.pruning = cv.tree(tree.titanic, FUN = prune.misclass)
plot(cv.pruning$size, cv.pruning$dev, pch=20, col="red", type="b",
     main="Cross validation for finding the optimal size",
     xlab="size of tree", ylab="performance")
```




```
#let us extract the optimal size
best.size = cv.pruning$size[which(cv.pruning$dev == min(cv.pruning$dev))]
cat(" Optimal size =", best.size, "\n")

##  Optimal size = 9

#and let us prune the tree accordingly
tree.titanic.pruned = prune.misclass(tree.titanic, best=best.size[1])
plot(tree.titanic.pruned)
text(tree.titanic.pruned, cex=0.75)
```



Now, let us see the performance of this simple tree model on the test set.

```
#let us make sme prediction and look at the error rate
tree.titanic.pred = predict(tree.titanic.pruned , titanic[-titanic.train,-1], type="class")
titanic.tree.results = table(tree.titanic.pred, titanic[-titanic.train,1] )
print(titanic.tree.results)

##
## tree.titanic.pred   0   1
##                 0 169  36
##                 1  39 107

cat("Basic Tree :: error rate = ",
```

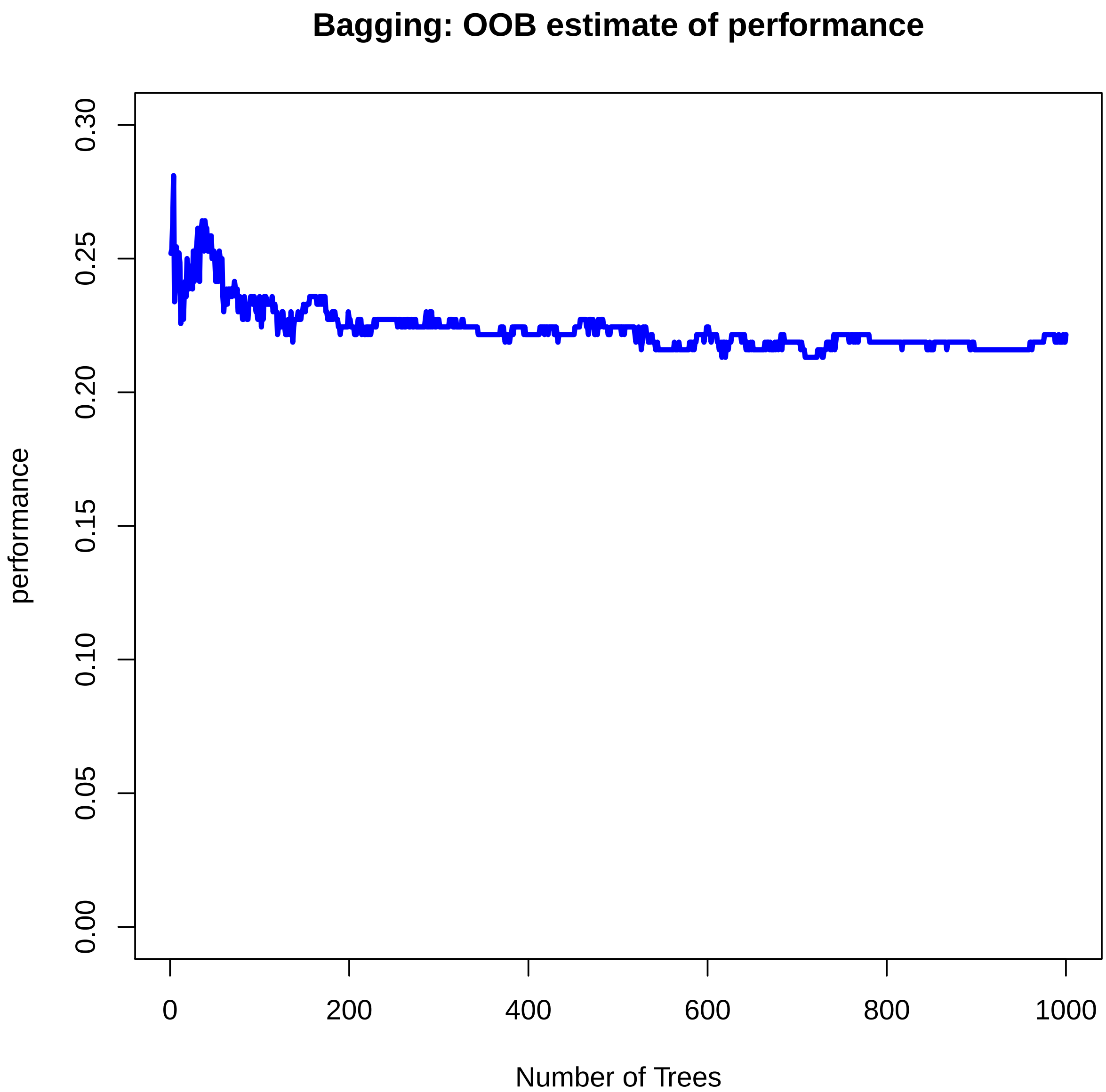
```
    (titanic.tree.results[1,2] + titanic.tree.results[2,1])/n_test,
    "\n" )

## Basic Tree :: error rate =  0.2136752

#let us compare that to naive Bayes
library(e1071)
titanic.naive = naiveBayes(Survived ~ ., titanic[titanic.train,], type="raw")
titanic.naive.pred = predict(titanic.naive, titanic[-titanic.train,-1])
titanic.naive.results = table(titanic.naive.pred, titanic[-titanic.train,1] )
print(titanic.naive.results)

##
## titanic.naive.pred   0   1
##                  0 189  58
##                  1  19  85

cat("Naive Bayes :: error rate = ",
    (titanic.naive.results[1,2] + titanic.naive.results[2,1])/(n_total - length(titanic.train)),
    "\n" )

## Naive Bayes :: error rate =  0.2193732
```

One can see that maybe surprisingly, a simple basic tree is doing already more or less as well as the naive Bayes classifier.

2 Bagging: Titanic dataset

Let us try the bagging approah on the Titanic dataset. Note that Bagging is the same as random forest if at each splitting all the set of covariates are used.

```
library(randomForest)

## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.

titanic.bagging= randomForest( Survived~., data=titanic,
  subset = titanic.train, mtry=7, ntree=1000, importance=TRUE)
#plot performances (OOB estimate)
plot(titanic.bagging$err.rate[,1], type="l", lwd=3, col="blue",
     main="Bagging: OOB estimate of performance",
     xlab="Number of Trees", ylab="performance",
     ylim=c(0,0.3))
```



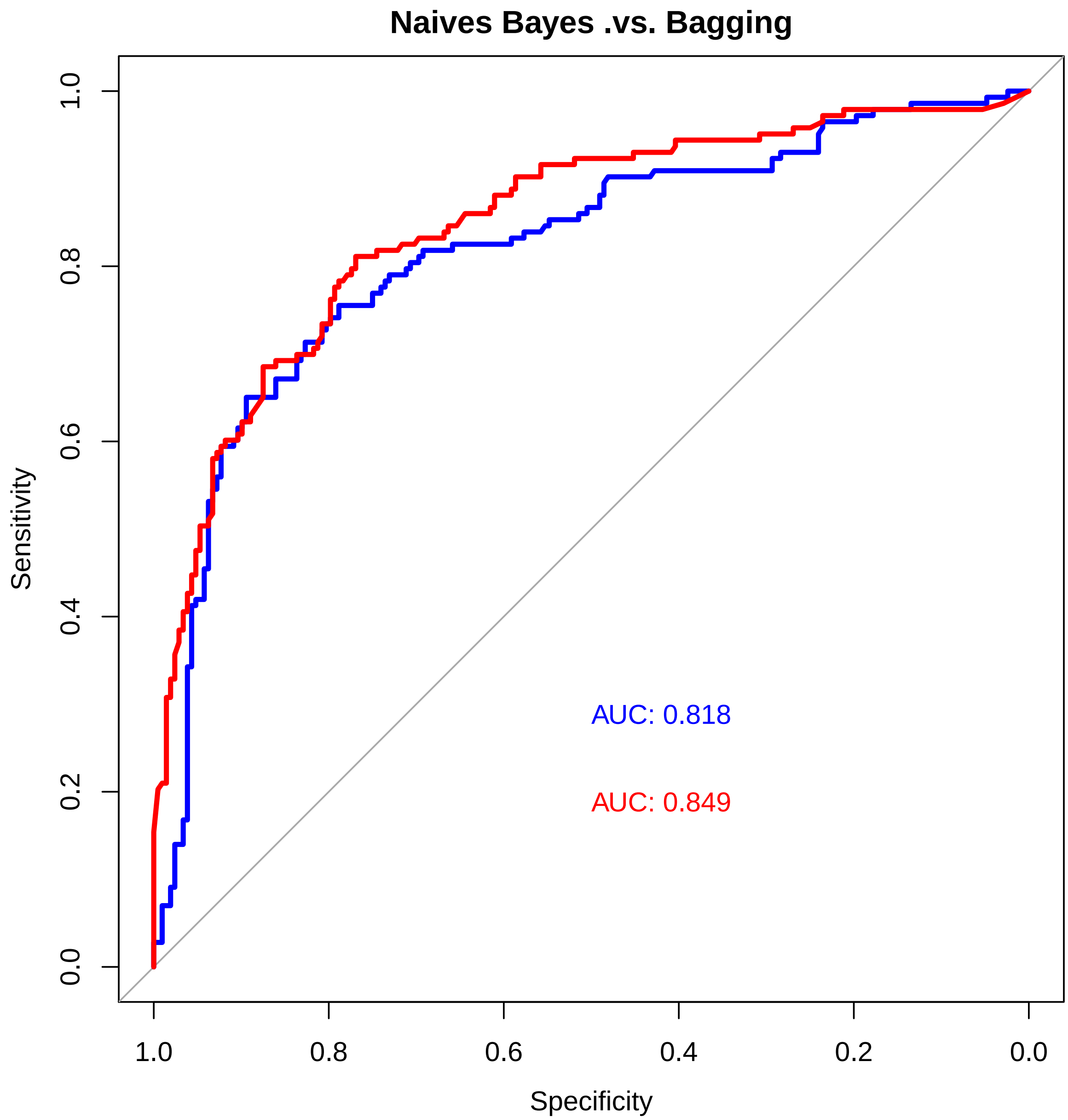

```
#let us compare bagging v.s. naive Bayes
#in terms of AUC
titanic.bagging.pred = predict( titanic.bagging ,
  newdata = titanic[ -titanic.train, -1], type="prob")
titanic.naive.pred.prob = predict(titanic.naive,
  titanic[-titanic.train,-1], type="raw")
#plot ROC curve + AUC
library(pROC)

## Type 'citation("pROC")' for a citation.
##
## Attaching package:  'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
plot.roc(titanic[-titanic.train, "Survived"], titanic.naive.pred.prob[,1], col="blue",
         lwd=3, print.auc=TRUE,print.auc.y = 0.3,
         main="Naives Bayes .vs. Bagging")
```

```
##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.naive.pred.prob[,     1
##
## Data: titanic.naive.pred.prob[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 case
## Area under the curve: 0.8185
```

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

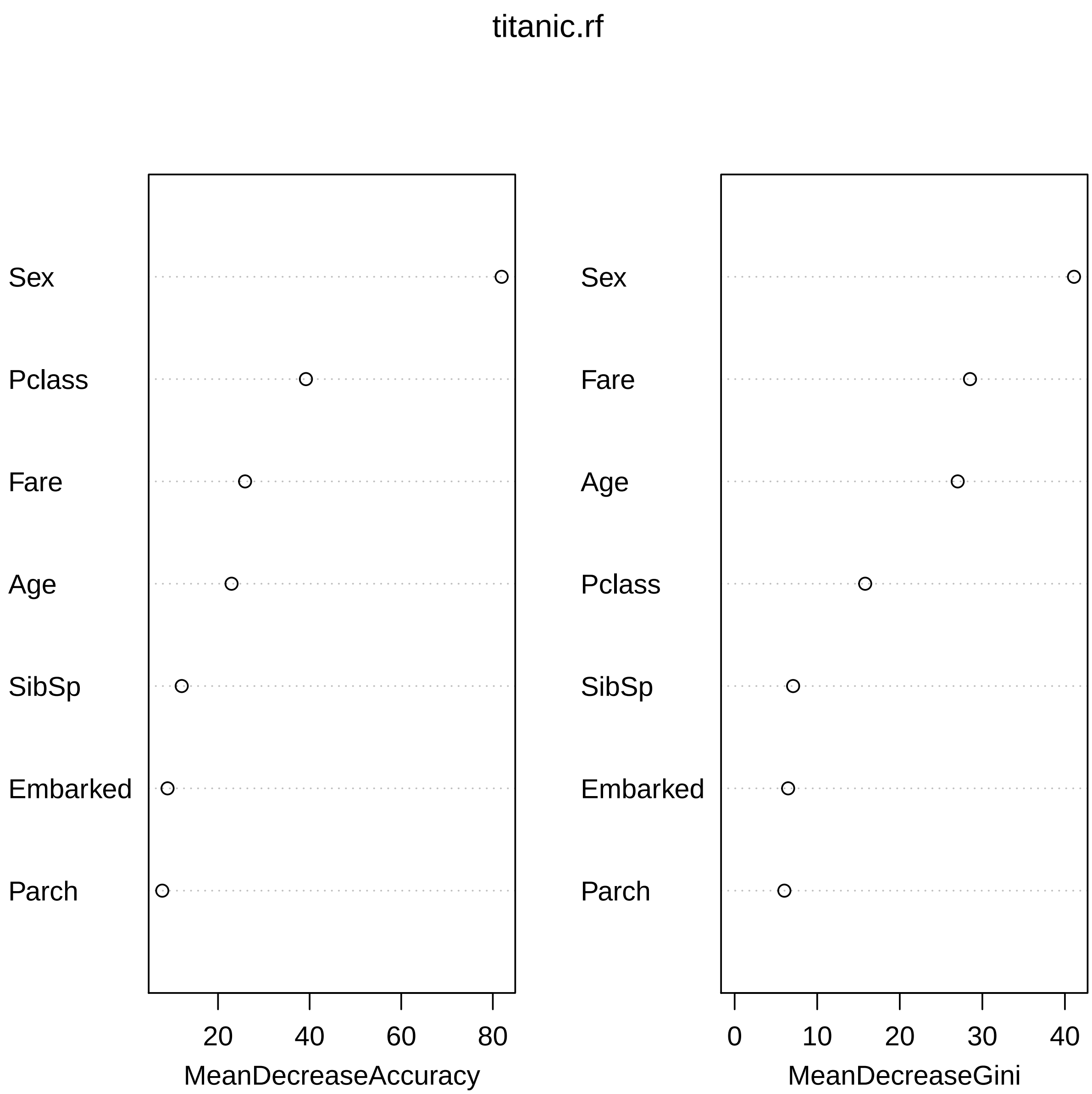



```
##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.bagging.pred[,     1],
##
## Data: titanic.bagging.pred[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 cases
## Area under the curve: 0.8493
```

3 Random Forest: Titanic dataset

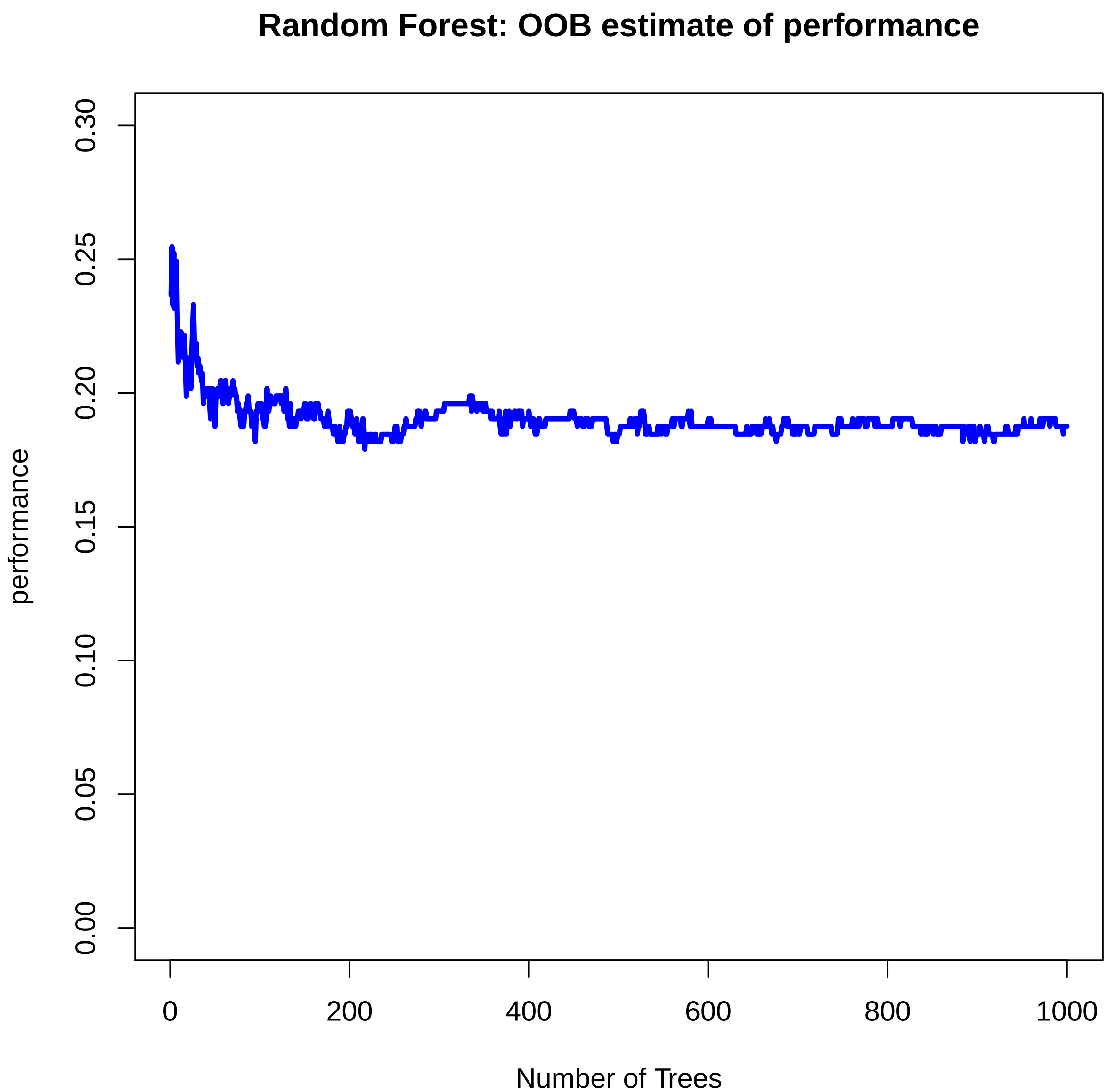
Let us try the random forest approach.

```r
# we will be suing ntry=2
titanic.rf= randomForest( Survived~., data=titanic,
  subset = titanic.train, mtry=2, ntree=1000, importance=TRUE)

#variable importance
varImpPlot(titanic.rf)
```



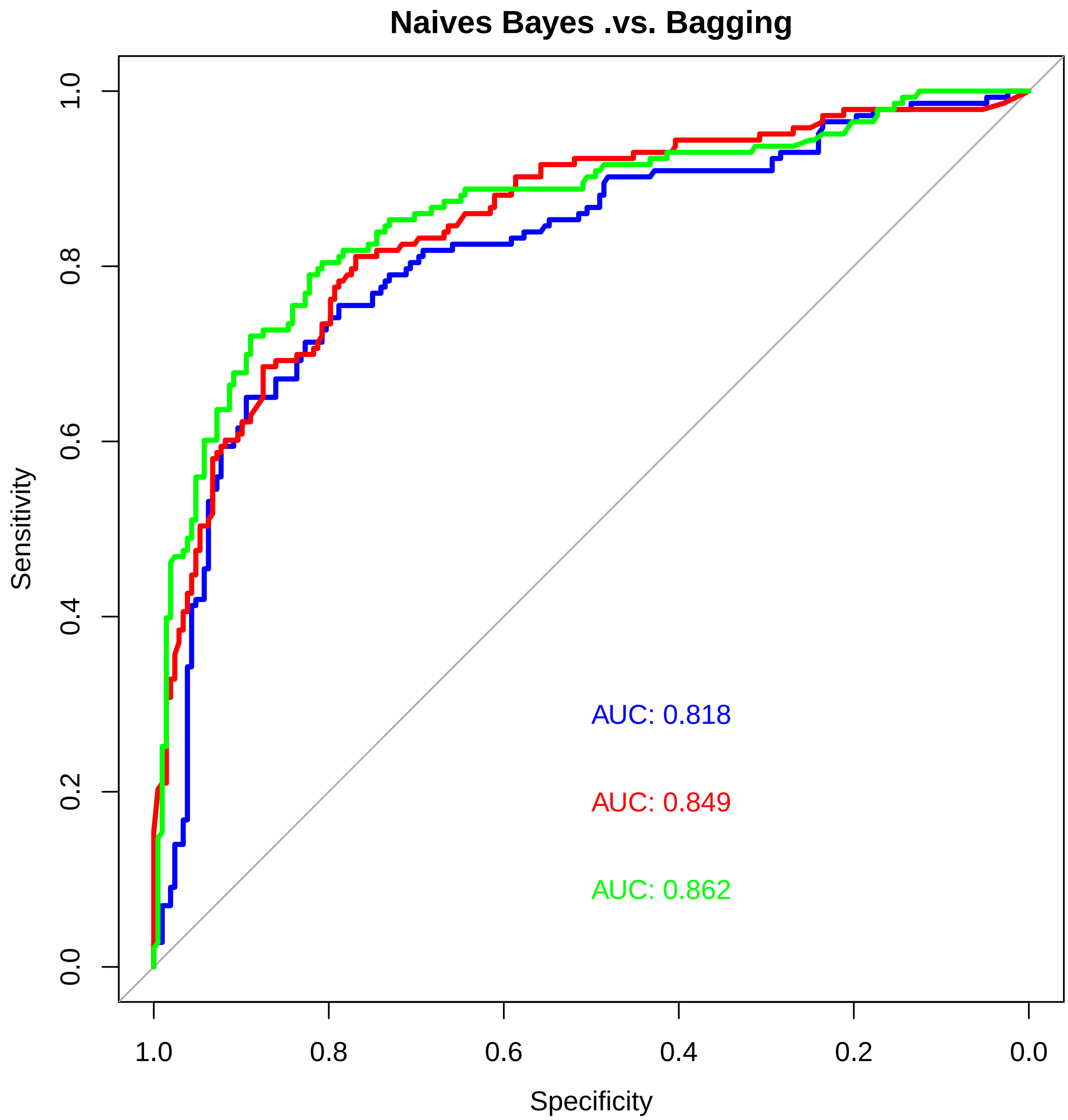

```r
#plot performances (OOB estimate)
plot(titanic.rf$err.rate[,1], type="l", lwd=3, col="blue",
     main="Random Forest: OOB estimate of performance",
     xlab="Number of Trees", ylab="performance",
```

```
    ylim = c(0, 0.3))
```




```
#let us compare random-forest v.s. naive Bayes
#in terms of AUC
titanic.rf.pred = predict( titanic.rf ,
  newdata = titanic[ -titanic.train, -1], type="prob")
titanic.naive.pred.prob = predict(titanic.naive,
  titanic[-titanic.train,-1], type="raw")
#plot ROC curve + AUC
library(pROC)
plot.roc(titanic[-titanic.train, "Survived"], titanic.naive.pred.prob[,1], col="blue",
```

```
        lwd=3, print.auc=TRUE,print.auc.y = 0.3,
        main="Naives Bayes .vs. Bagging")

##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.naive.pred.prob[,     1
##
## Data: titanic.naive.pred.prob[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 case
## Area under the curve: 0.8185

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

##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.bagging.pred[,     1],
##
## Data: titanic.bagging.pred[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 cases
## Area under the curve: 0.8493

plot.roc(titanic[-titanic.train, "Survived"], titanic.rf.pred[,1], col="green",
        lwd=3, print.auc=TRUE,print.auc.y = 0.1, add=TRUE)
```

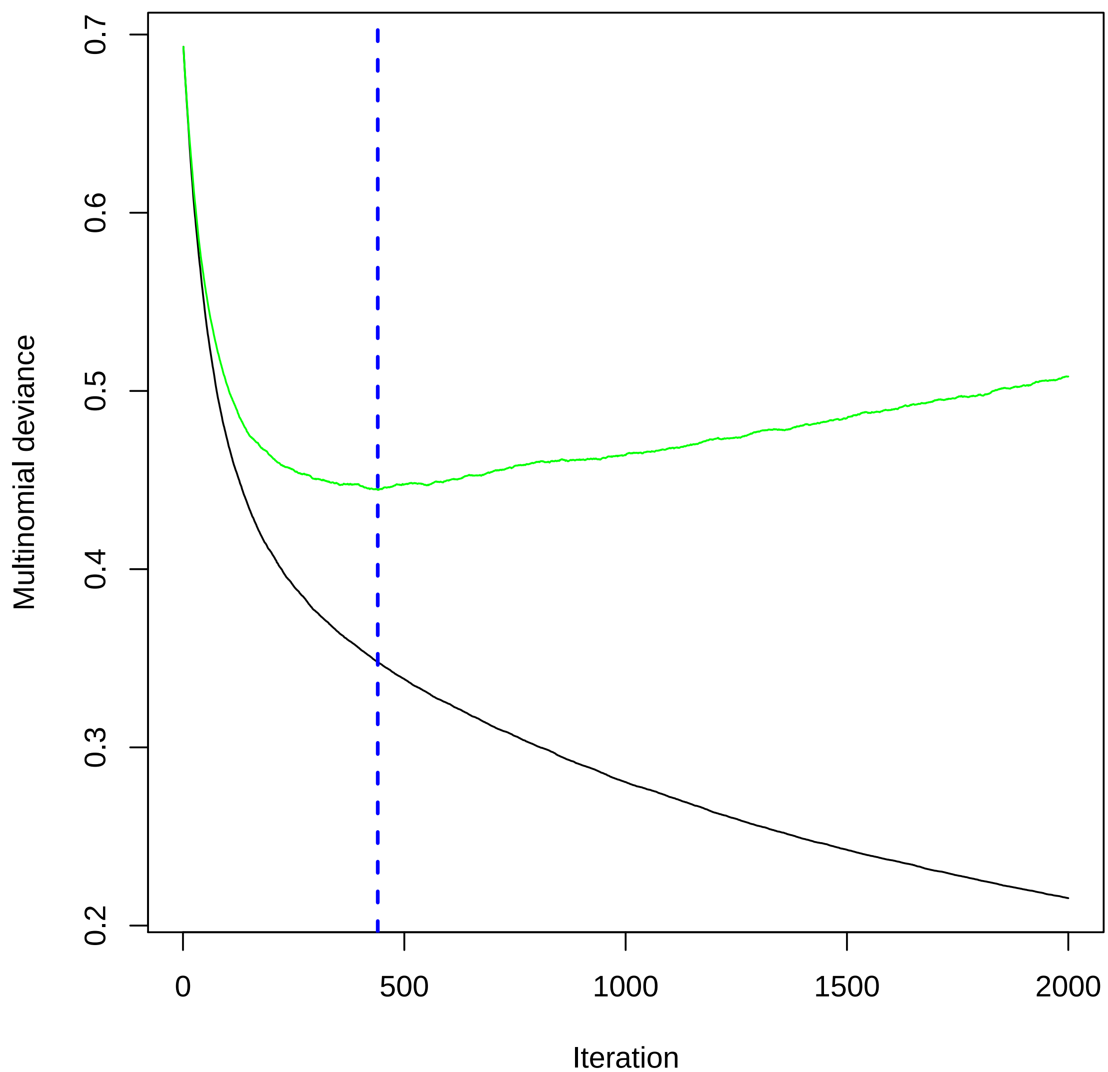



```
##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.rf.pred[,     1], col =
##
## Data: titanic.rf.pred[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 cases (titan
## Area under the curve: 0.8621
```

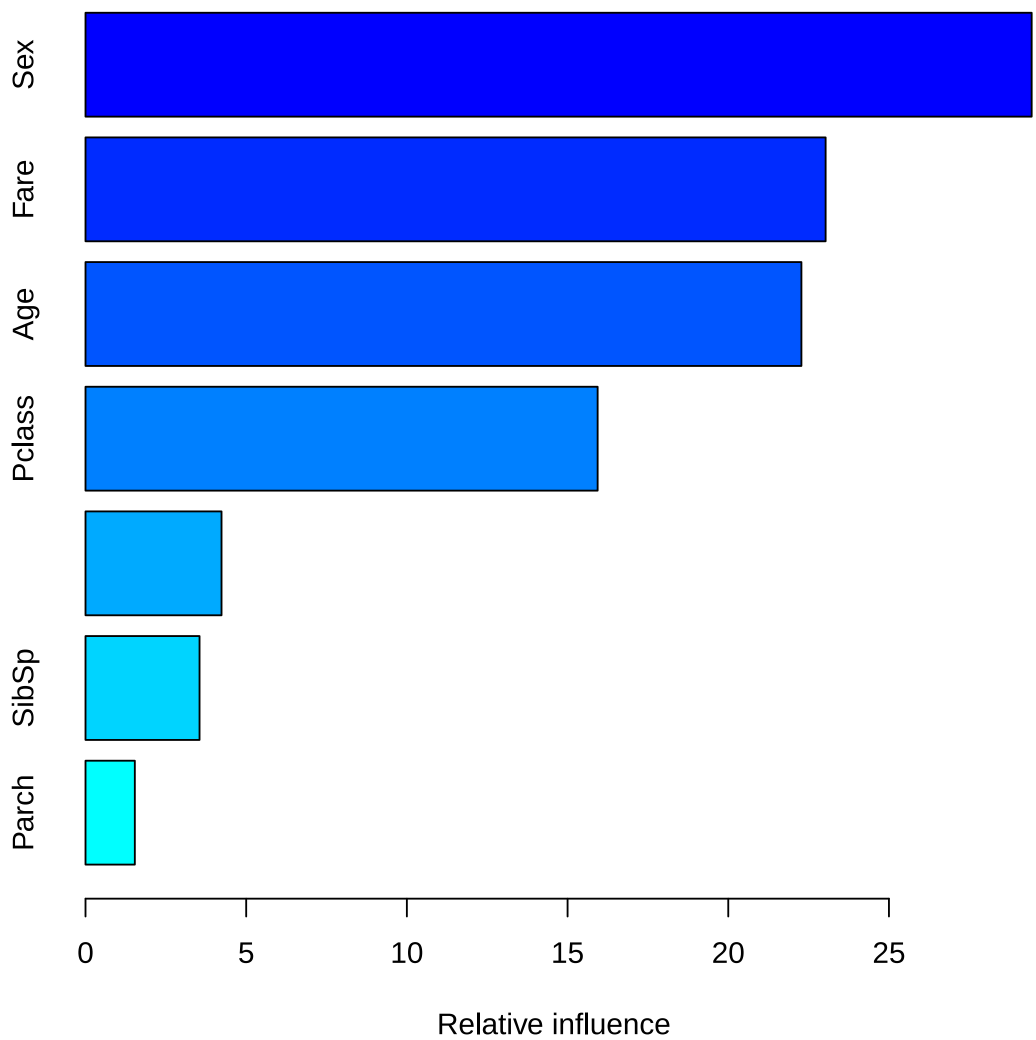
On this dataset, the performances of bagging and random-forest a very similar. This mainly because the Titanic dataset is a very easy dataset to analyse.

4 Boosting: Titanic

```
library(gbm)

## Loading required package:  survival
## Warning:  package 'survival' 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:  splines
## Loading required package:  parallel
## Loaded gbm 2.1.1

#try boosting with different size of learning rate, just for illustration
titanic.boosting = gbm(Survived~., data=titanic[titanic.train,],
  distribution="multinomial",
  n.trees=2000,
  interaction.depth=4, cv.folds=5,
  shrinkage=0.005)


#compute optimal (by corss-validation) number of tress
gbm.perf(titanic.boosting, plot.it = TRUE)

## Using cv method...
```



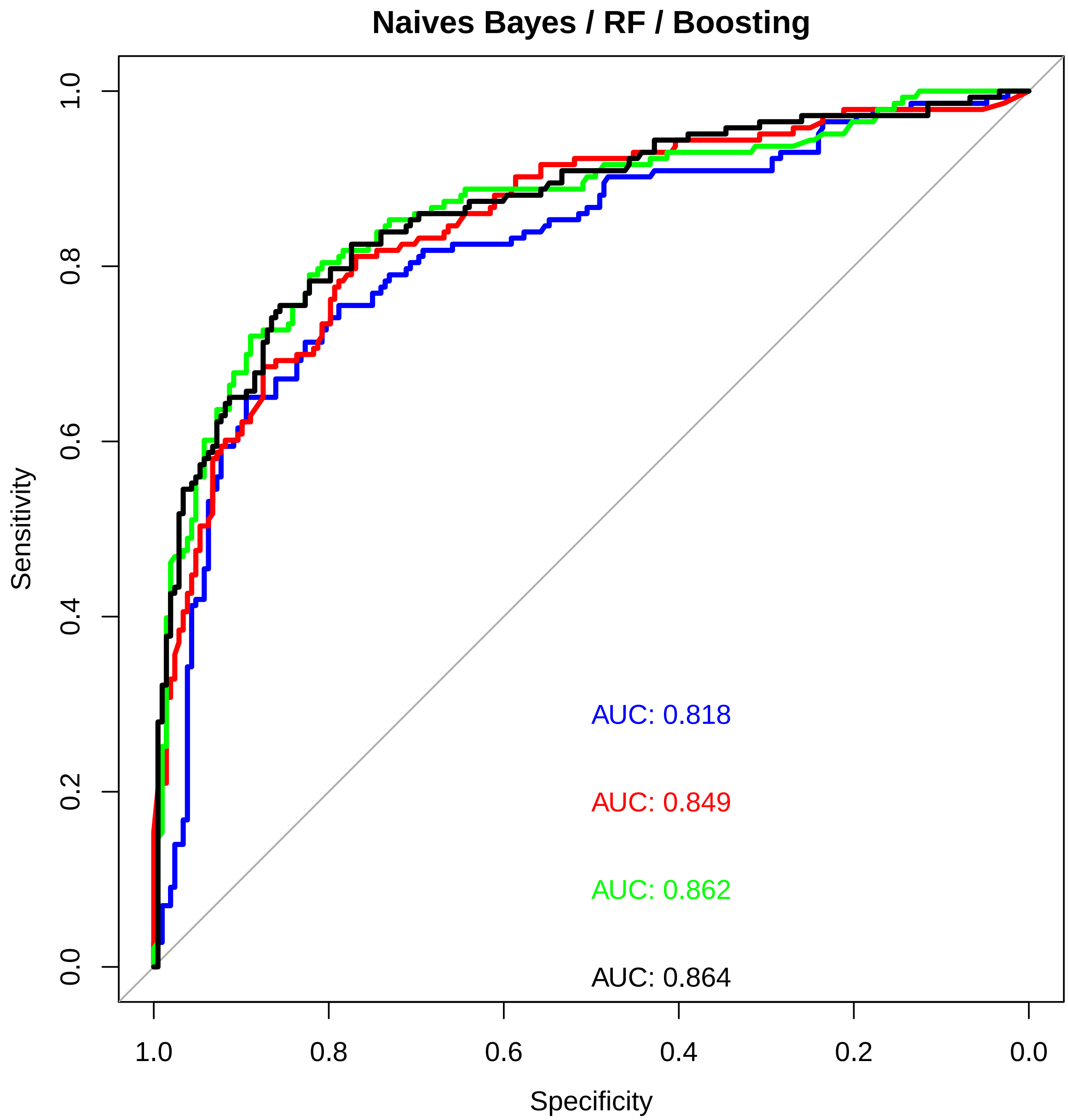

```
## [1] 440

boost.optimal = gbm.perf(titanic.boosting, plot.it = FALSE)

## Using cv method...

#moake some prediction
titanic.boosting.pred = predict(titanic.boosting,
  newdata=titanic[-titanic.train,], n.trees=boost.optimal, type="response")

#variable importance
summary(titanic.boosting)
```




```
##                var   rel.inf
## Sex            Sex 29.441446
## Fare          Fare 23.030067
## Age            Age 22.276758
## Pclass      Pclass 15.936242
## Embarked Embarked  4.232691
## SibSp        SibSp  3.548007
## Parch        Parch  1.534789
```

```
#compare with naive-bayes and random-forst
plot.roc(titanic[-titanic.train, "Survived"], titanic.naive.pred.prob[,1], col="blue",
         lwd=3, print.auc=TRUE,print.auc.y = 0.3,
```

```
        main="Naives Bayes / RF / Boosting")

##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.naive.pred.prob[,     1
##
## Data: titanic.naive.pred.prob[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 case
## Area under the curve: 0.8185

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

##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.bagging.pred[,     1],
##
## Data: titanic.bagging.pred[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 cases
## Area under the curve: 0.8493

plot.roc(titanic[-titanic.train, "Survived"], titanic.rf.pred[,1], col="green",
         lwd=3, print.auc=TRUE,print.auc.y = 0.1, add=TRUE)

##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.rf.pred[,     1], col =
##
## Data: titanic.rf.pred[, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 cases (tita
## Area under the curve: 0.8621

plot.roc(titanic[-titanic.train, "Survived"], titanic.boosting.pred[,1,1], col="black",
         lwd=3, print.auc=TRUE,print.auc.y = 0.0, add=TRUE)
```




```
##
## Call:
## plot.roc.default(x = titanic[-titanic.train, "Survived"], predictor = titanic.boosting.pred[,     1,
##
## Data: titanic.boosting.pred[, 1, 1] in 208 controls (titanic[-titanic.train, "Survived"] 0) > 143 ca
## Area under the curve: 0.8643
```

5 Hitters dataset

Let us analyse the *Hitters* dataset; we will try to build a regression model for prediction the *Salary* variable. First, let us prepare the data

```
library(ISLR)  #for "Hitters" dataset
attach(Hitters)
#for simplicity, delete rows with N.A. entries
Hitters = na.omit(Hitters)
names(Hitters)

##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"
##  [6] "Walks"     "Years"     "CAtBat"    "CHits"     "CHmRun"
## [11] "CRuns"     "CRBI"      "CWalks"    "League"    "Division"
## [16] "PutOuts"   "Assists"   "Errors"    "Salary"    "NewLeague"

#For simplicity, drop non numerical values
Hitters = Hitters[,!(names(Hitters)
                     %in% c("League", "Division", "NewLeague") ) ]

#create the covariate matrix (but delete the intercept)
x = model.matrix(Salary ~ ., Hitters)[,-1]
y = Hitters$Salary

#split training / test
n_data = length(y)
hitters.train = sample(1:n_data, 0.5*n_data, replace=FALSE)
x_train = x[hitters.train,]
y_train = y[hitters.train]
x_test = x[-hitters.train,]
y_test = y[-hitters.train]
```

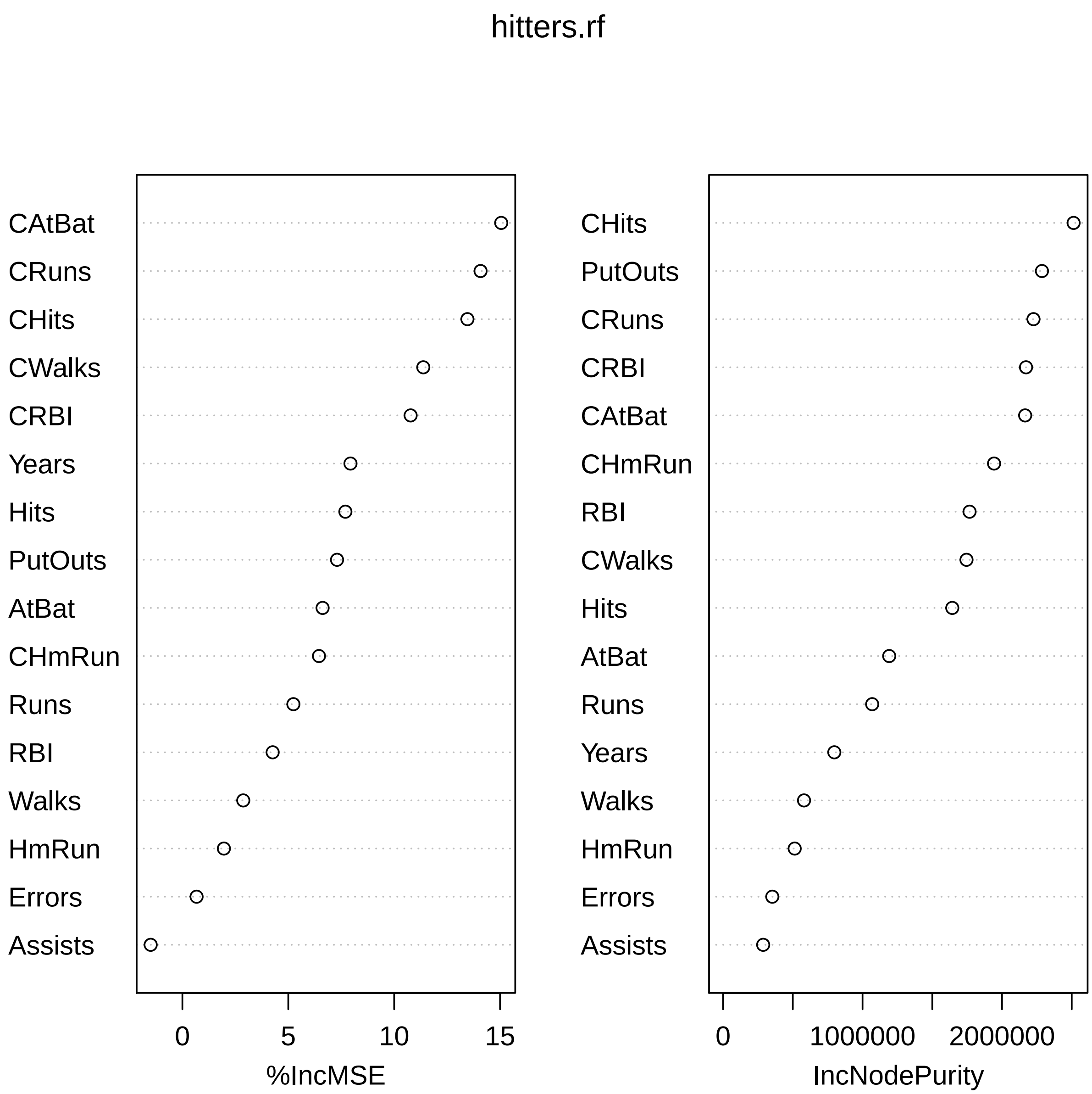
Let us compare several approaches.

```
#LASSO
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

lasso.cv = cv.glmnet(x_train, y_train, alpha=1,
      type.measure = "mse", nfolds = 10)
#relevant variables
names(Hitters)[which( coef(lasso.cv, s = "lambda.min") != 0 )]
```

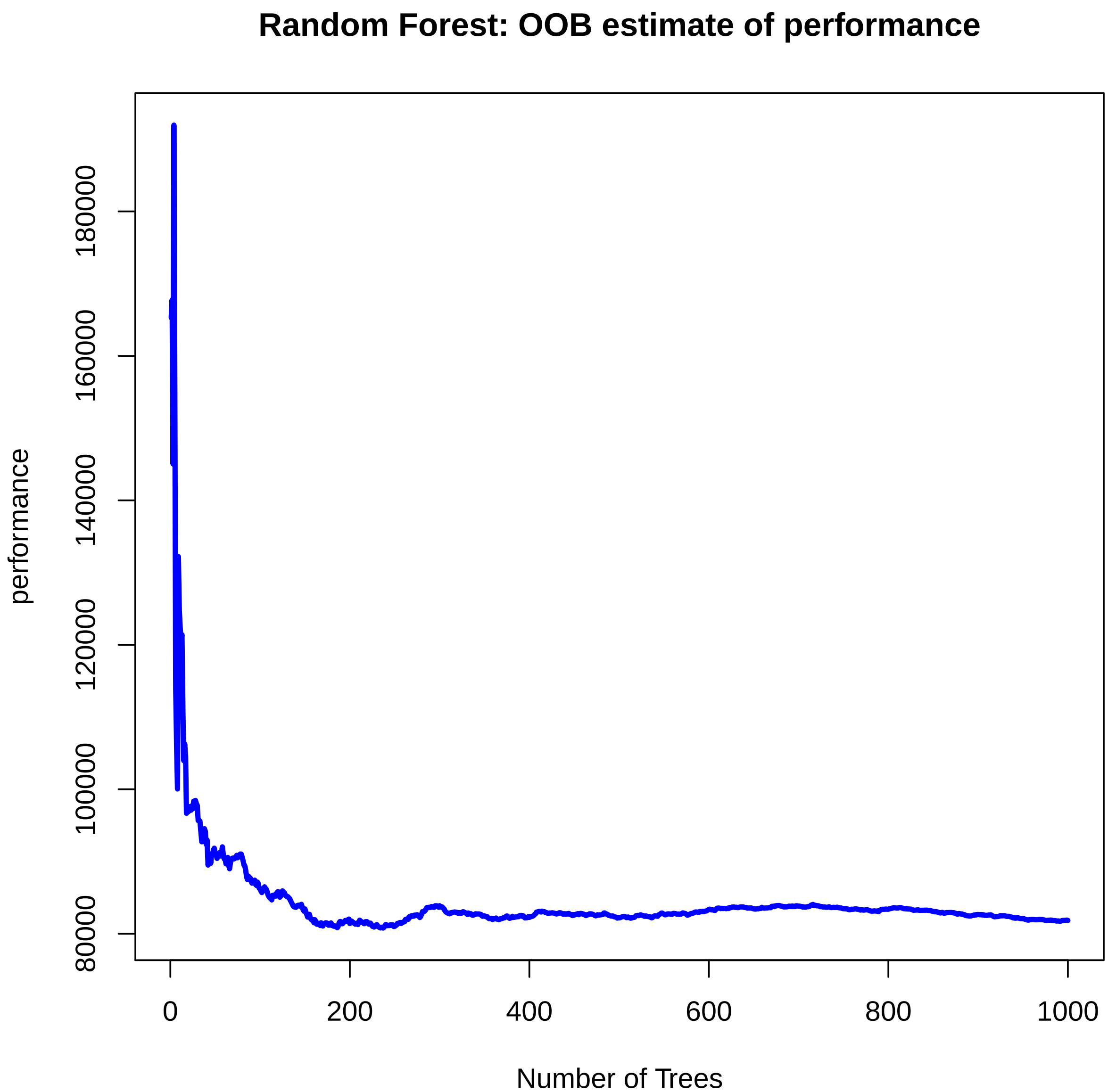
```
##  [1] "AtBat"   "Hits"    "HmRun"   "Runs"    "RBI"     "Walks"   "Years"
##  [8] "CAtBat"  "CHits"   "CHmRun"  "CRuns"   "CRBI"    "CWalks"  "PutOuts"
## [15] "Assists" "Errors"  "Salary"
```

```
lasso.predict = predict(lasso.cv, newx = x_test,
      s = "lambda.min", exact=TRUE)
hitters.lasso.MSE = mean( (y_test - lasso.predict)**2 )
```

```
#random forest
randomForest(x_train, y_train)

##
## Call:
##  randomForest(x = x_train, y = y_train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 78893.56
##                     % Var explained: 57.46

hitters.rf= randomForest( Salary~., data=Hitters, subset = hitters.train, mtry=3, ntree=1000, importanc

#variable importance
varImpPlot(hitters.rf)
```



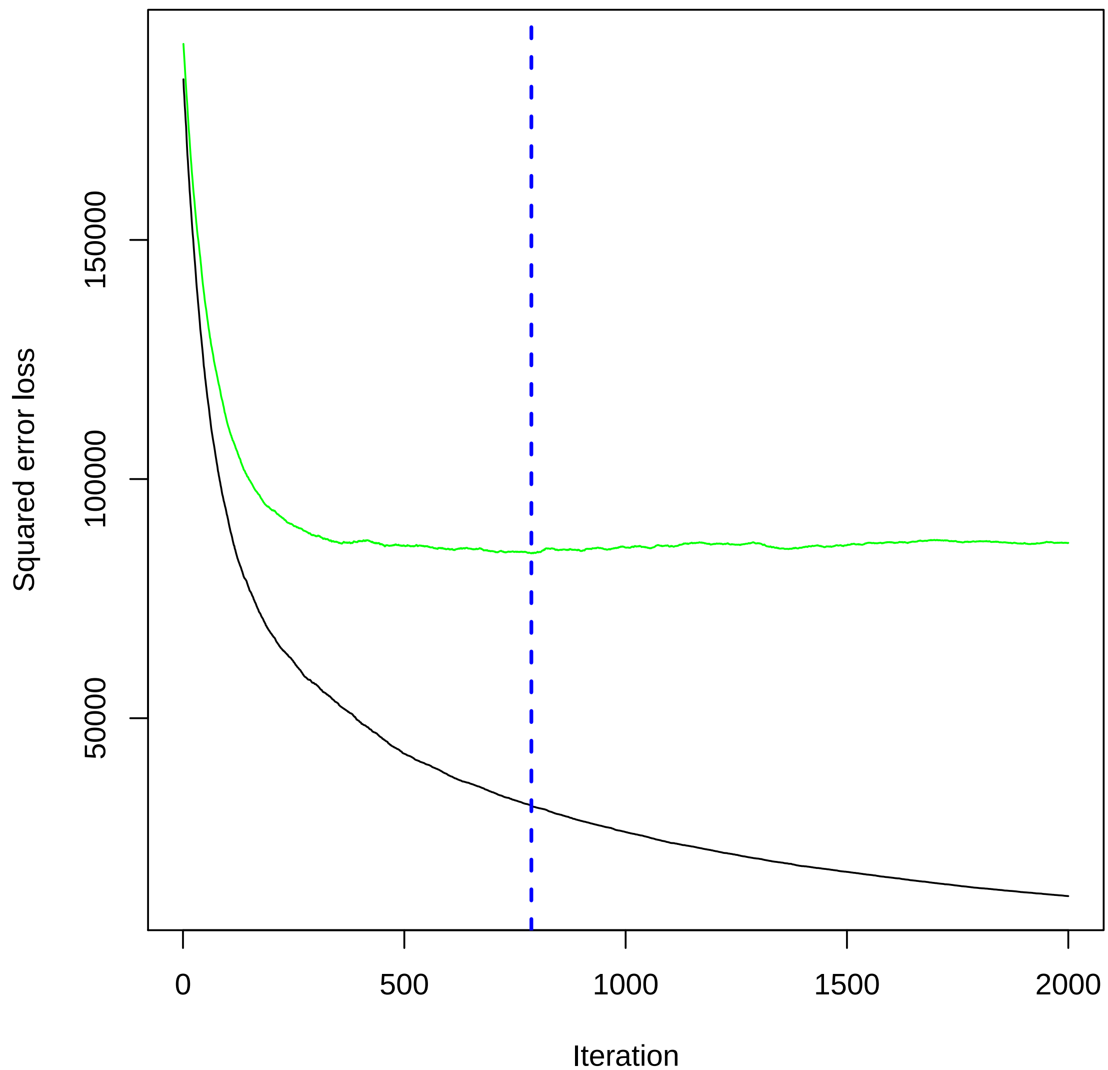

```r
#plot performances (OOB estimate)
plot(hitters.rf$mse, type="l", lwd=3, col="blue",
     main="Random Forest: OOB estimate of performance",
     xlab="Number of Trees", ylab="performance")
```




```
#make some predictions
hitters.rf.pred = predict( hitters.rf , newdata = x_test)
hitters.rf.MSE = mean( (y_test - hitters.rf.pred)**2 )
```

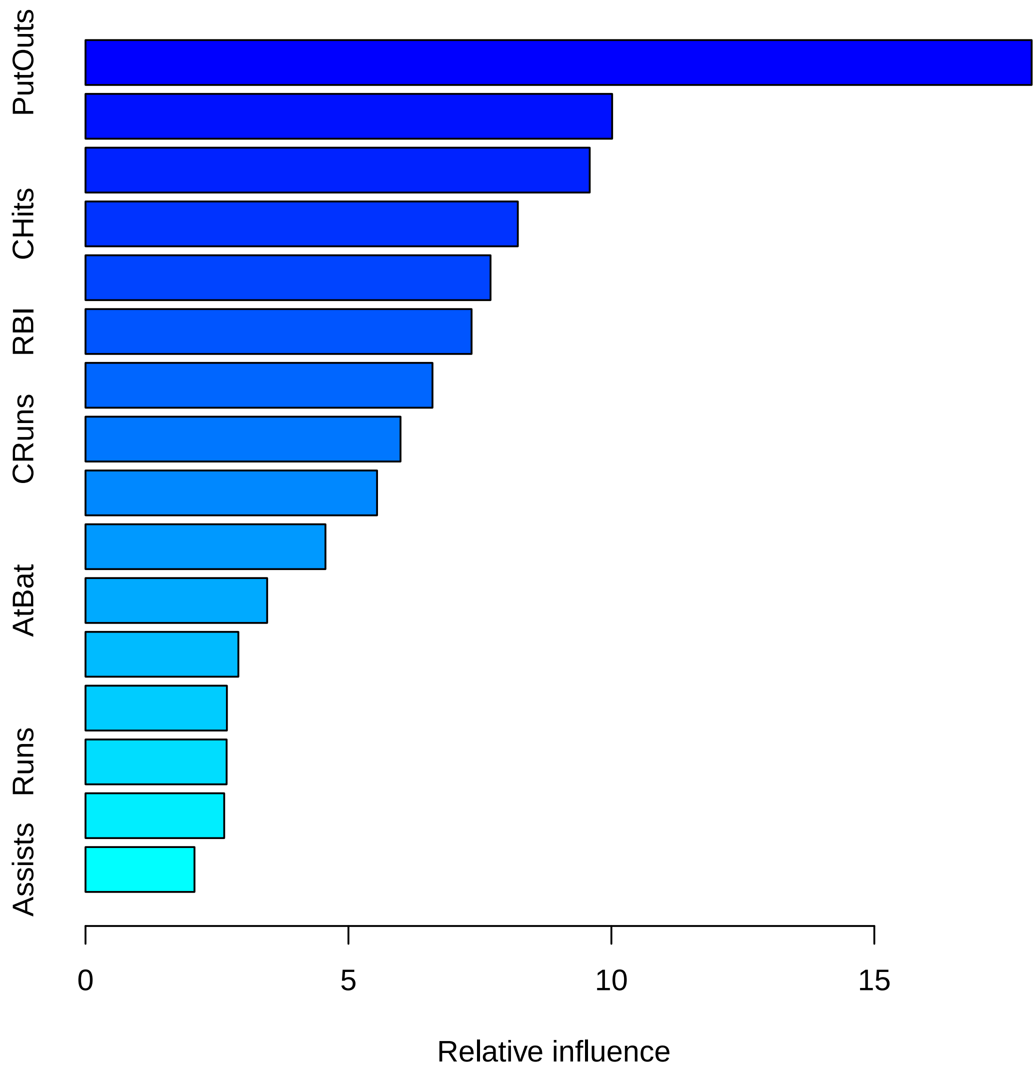
```
#Boosting
hitters.boosting = gbm(Salary~., data=Hitters[hitters.train,],
  distribution="gaussian",
  n.trees=2000,
  interaction.depth=2, cv.folds=5,
  shrinkage=0.01)
```

```r
#compute optimal (by corss-validation) number of tress
gbm.perf(hitters.boosting, plot.it = TRUE)

## Using cv method...
```




```r
## [1] 787

boost.optimal = gbm.perf(hitters.boosting, plot.it = FALSE)

## Using cv method...

#variable importance
```

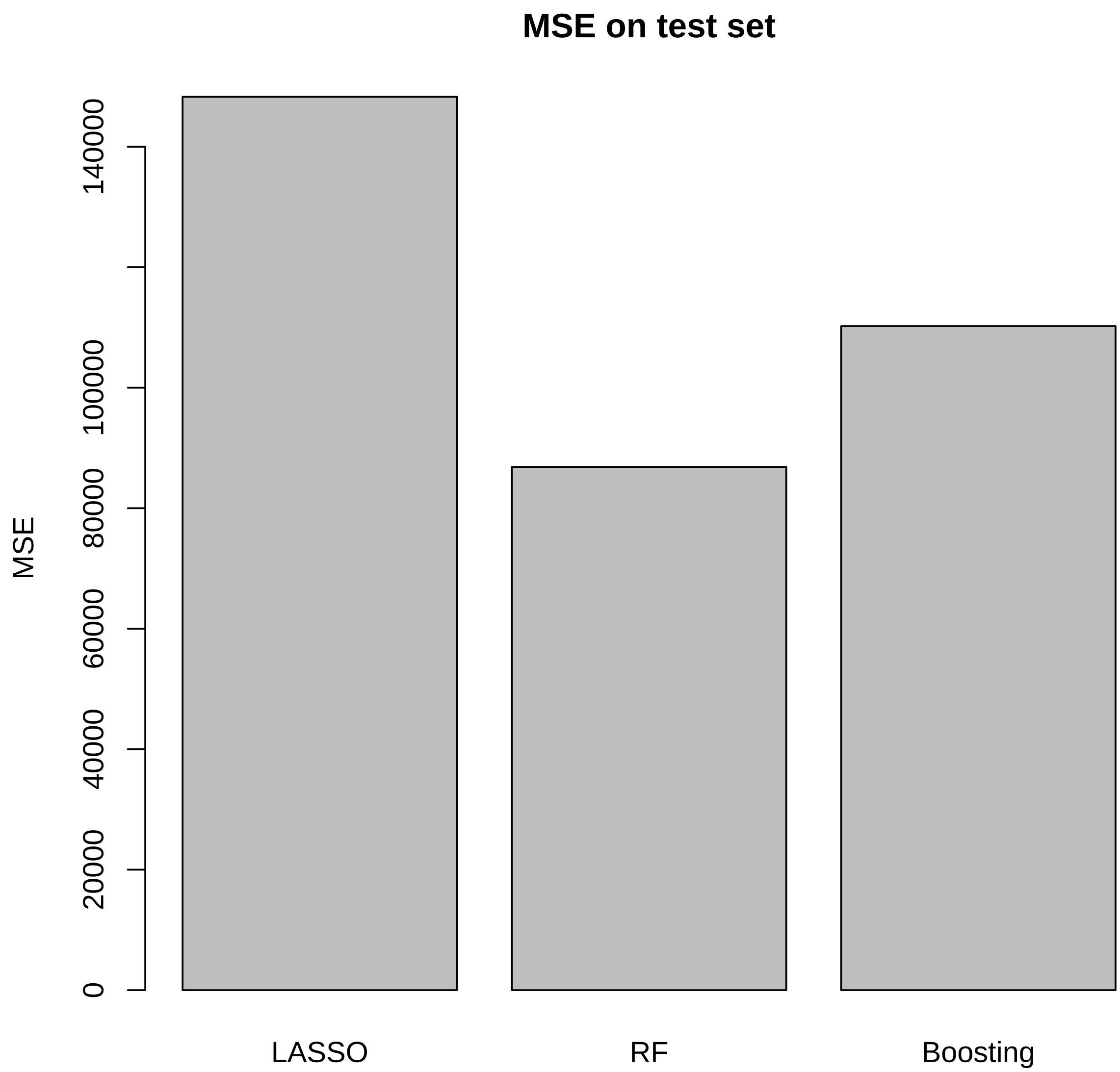
```r
summary(hitters.boosting)
```




```
##             var   rel.inf
## PutOuts PutOuts 17.992619
## CHmRun   CHmRun 10.014888
## CRBI       CRBI  9.588328
## CHits     CHits  8.221379
## Hits       Hits  7.702031
## RBI         RBI  7.341662
## CWalks   CWalks  6.598253
## CRuns     CRuns  5.990084
```

```
## CAtBat   CAtBat  5.544132
## Years     Years  4.563028
## AtBat     AtBat  3.454543
## HmRun     HmRun  2.906718
## Walks     Walks  2.688804
## Runs       Runs  2.683716
## Errors   Errors  2.637499
## Assists Assists  2.072314

#make some prediction
hitters.boosting.pred = predict(hitters.boosting,
  newdata=Hitters[-hitters.train,],
  n.trees=boost.optimal, type="response")

hitters.boosting.MSE = mean( (y_test - hitters.boosting.pred)**2 )
```

```
barplot(c(hitters.lasso.MSE,hitters.rf.MSE,hitters.boosting.MSE),
        main="MSE on test set", ylab="MSE",
        names.arg=c("LASSO", "RF", "Boosting"))
```


MSE on test set
MSE
0
20000
40000
60000
80000
100000
140000
LASSO
RF
Boosting