Homework 2

Anastasia Santo

2023-05-02

In this homework I want to to investigate the association between the level of prostate-specific antigen (`lpsa`, in ng/ml and log scaled) and a number of clinical measures, measured in 97 men who were about to receive a radical prostatectomy. In particular, the explanatory variables are:

- `lcavol`: log(cancer volume in cm3)
- `lweight`: log(prostate weight in g)
- `age` in years
- `lbph`: log(amount of benign prostatic hyperplasia in cm2)
- `svi`: seminal vesicle invasion (1 = yes, 0 = no)
- `lcp`: log(capsular penetration in cm)
- `gleason`: Gleason score for prostate cancer (6,7,8,9)
- `pgg45`: percentage of Gleason scores 4 or 5, recorded over their visit history before their final current Gleason score

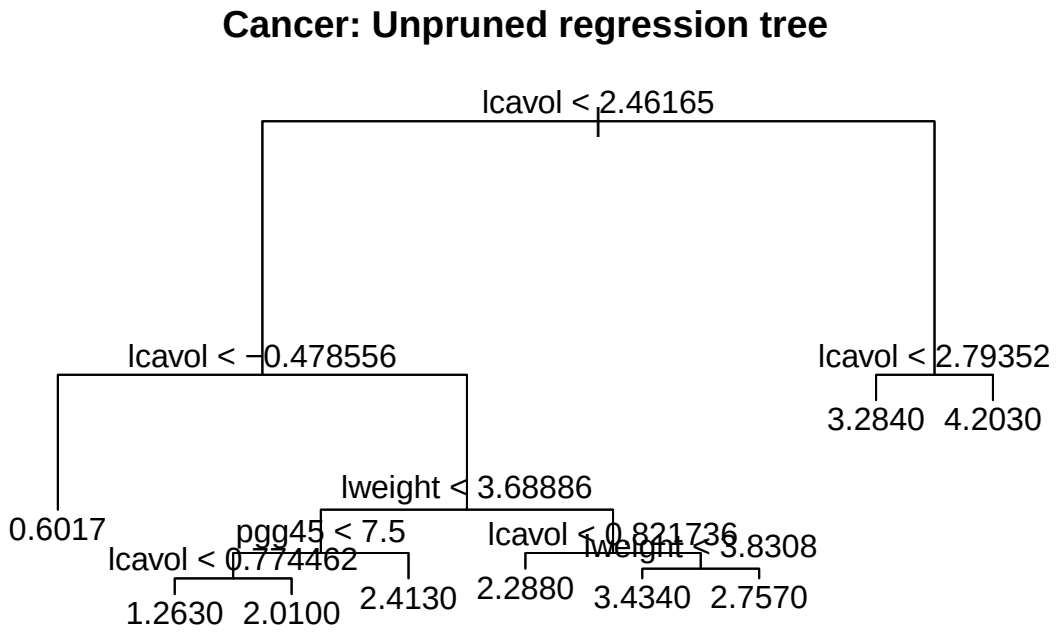
After a quick exploration of the data, I checked out for missing data and I explored the different type of variables. The data set is complete and all the variables are quantitative while just 'svi' is cathegorical and can take only 0-1 values. I started splitting the data in two halves for training and testing, but for the next points of the work I will use the entire dataset.

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
```

Since "lpsa" is a continuous variable I fit a regression tree on the entire data set in order to predict "lpsa" using all the variables of the Data, and I explore its structure.

```
##
## Regression tree:
## tree(formula = lpsa ~ ., data = prostate)
## Variables actually used in tree construction:
## [1] "lcavol"  "lweight" "pgg45"
## Number of terminal nodes:  9
## Residual mean deviance:  0.4119 = 36.24 / 88
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.499000 -0.488000  0.003621  0.000000  0.481200  1.380000
```

The summary reports the variables that are used to construct the tree: only 3 out of 8, which are "lcavol", "lweight", "pgg45". The "residual mean deviance" is the total residual deviance (36,24) divided by the number of observations minus number of terminal nodes,and in this case it is 0.4119. In the context of regression tree, the deviance is simply the sum of squared errors for the tree. The number of terminal nodes here is 9, so the structure of the tree is quite complex.



Plotting the decision tree I can see its structure : this tree predicts higher level of prostate-specific antigen(lpsa) for higher values of log(cancer volume in cm3), which is "lcavol" ($lcavol > 2,4616$); specifically, the highest value of lcavol is reached for "lcavol" values higher than 2,7935. For lower "lcavol" values ($lcavol < -0,4785$) the "lpsa" value is very low (0,6017); For values of "lcavol" higher than -0,4785 but lower than 2,4616, other variables are taken into account : For lower values of log(prostate weight in g) which is "lweight" ($lweight < 3,6888$) and lower values of percentage of Gleason scores("pgg4"), if the value of "lcavol is higher than 0,7744 the"lpsa value is 2,0100, while if it is lower the "lpsa" value is 1,2630. For higher values of "lweight" ($lweight > 3,6886$) : if the values of "lcavol" is higher than 0,8217, and the value of "lweight" is higher than 3,8308, the "lpsa" value is 2,7570, while if it is lower the "lpsa" value is 3,4340. if the values of "lcavol" is lower than 0,8217 the "lpsa" value is 2,2880. I compute the Train set MSE and the Test set MSE for the unpruned tree :

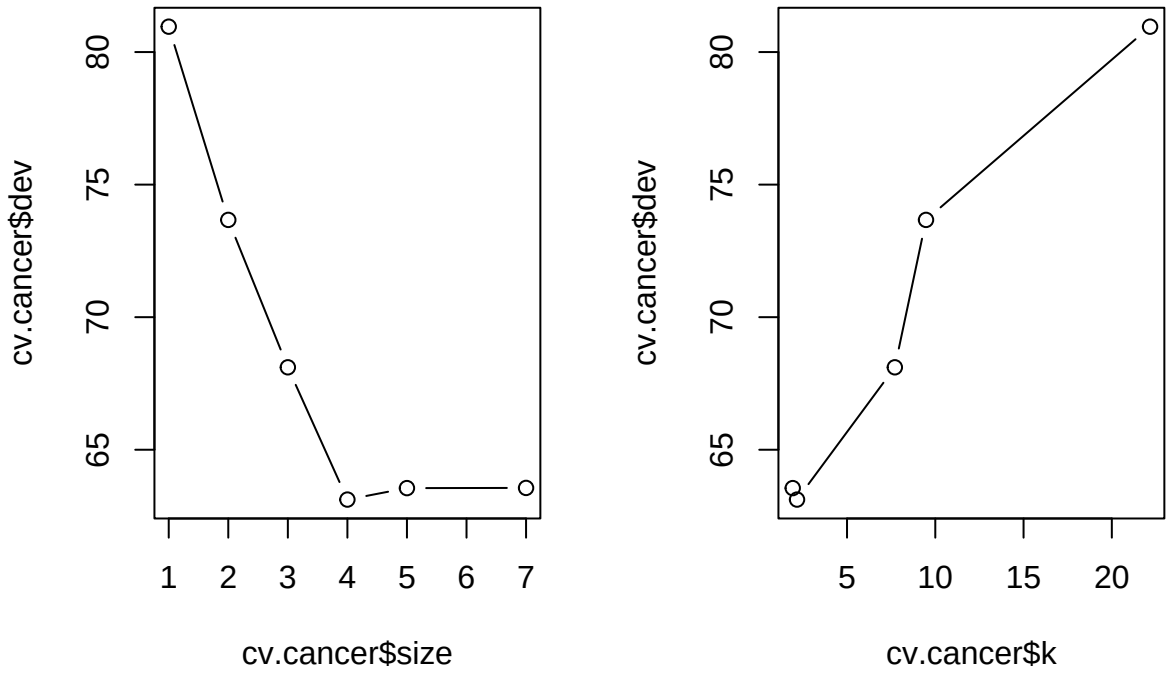
```
## [1] 0.3767986
```

```
## [1] 0.3705559
```

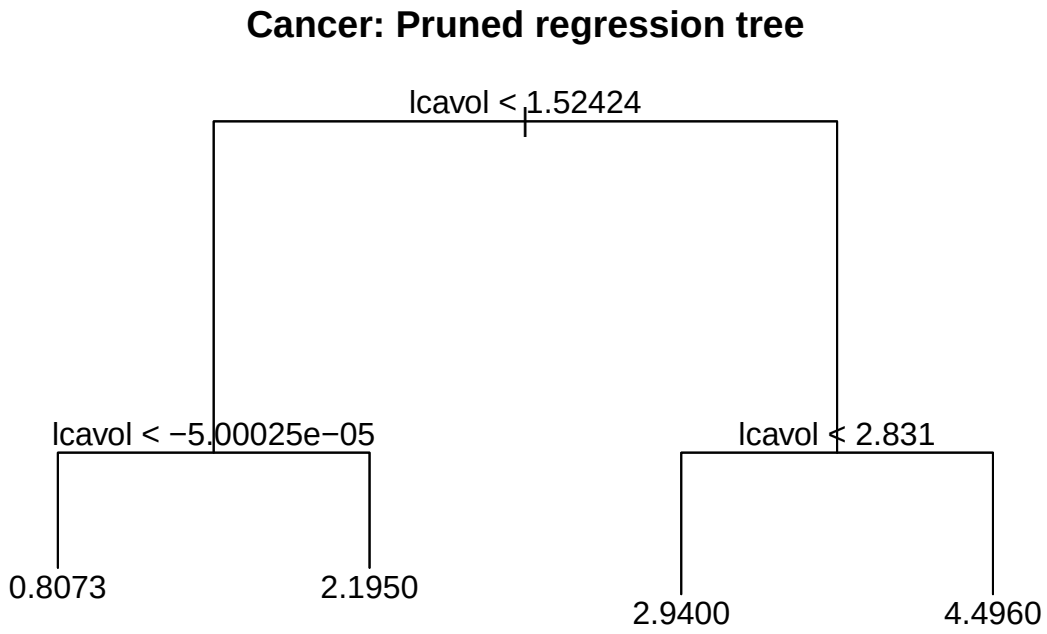
The train set MSE is 0,3767, while the test set MSE is 0,3705. The error values are low because the tree was fit on the entire dataset. I do the same fitting the tree only on the training dataset, and the MSE is 0,8237.

```
## [1] 0.8236777
```

Now I consider whether pruning the tree might lead to improved results, since the complexity of the decision tree can lead to overfitting. I use the cv.tree function with default prune.tree and K=10 in order to select a sequence of trees with different complexity, and then to determine the optimal level of complexity (number of terminal nodes) according to the deviance values. I plot the deviance as a function of both size and k (cost-complexity parameter).



The tree with 4 terminal nodes turns out the one with the lowest deviance as we can see from the plot, and the optimal k parameter is 4,4459. Then I applied the prune.tree function in order to prune the tree already created to obtain the 4-node tree.




```
##
## Regression tree:
## snip.tree(tree = tree_cancer, nodes = 6:5)
## Variables actually used in tree construction:
## [1] "lcavol"
## Number of terminal nodes:  4
## Residual mean deviance:  0.6158 = 27.09 / 44
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.79100 -0.48870  0.07751  0.00000  0.59290  1.24000
```

As we can see from the summary the residual mean deviance in this case is 0,5625, which is quite higher than the one of the unpruned tree. This was exprected because of the reduction of the complexity of the tree. The variables used to make the prediction are two : lcavol and lweight, while the number of terminal nodes is 4. Looking at the plot we I can say that this tree is less complex that the unpruned one : for values of "lcavol" higher than 2,4616 the "lpsa" value is 3,7650. For lower values of "lcavol", if this variable has values lower than -0,4785, the "lpsa" value is 0,6017, while instead we have to consider the variable "lweight". If "lweight" has values lower than 3,6888 the "lpsa" value is 2,0330, while it it has values higher than that, the "lpsa" value is "2,7120".

RANDOM FOREST

I consider now Random Forest,a method in which a number of decision trees is built on bootstrapped training sample; when building these decision trees, each time a split in a tree is considered, a random sample of "m" predictors is choosen as split candidates from the full set of p predictors; the strength of Random Forest is that it provides a way to decorrelates the trees. Here I define a k-fold cross-validation schema for the

selection of the best tuning parameter "m". I will even calculate the average cross validation error and the OOB error corresponding to each choice of "m".

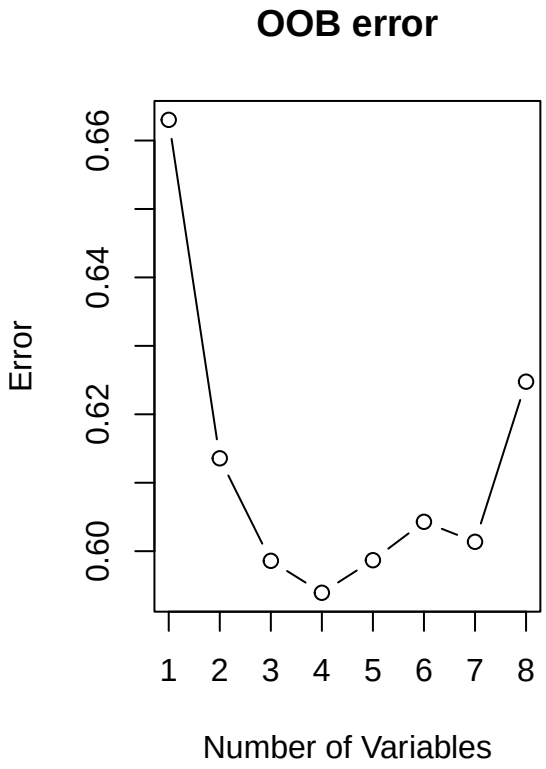
```
library(randomForest)
nvar<-ncol(prostate)-1
m<-c(1:nvar)
predictors<-prostate[,-9]
topred<-prostate[,9]
matrixerror<-matrix(nrow=nvar, ncol=2)
set.seed(1234)
for (i in m) {
      t<-tuneRF(predictors, topred, mtryStart = i,
                ntreeTry=2500,stepFactor=1,
                improve=0.5,trace=TRUE, plot=FALSE, doBest=FALSE, importance=TRUE)
      matrixerror[i,1]<-t[2]
}
```

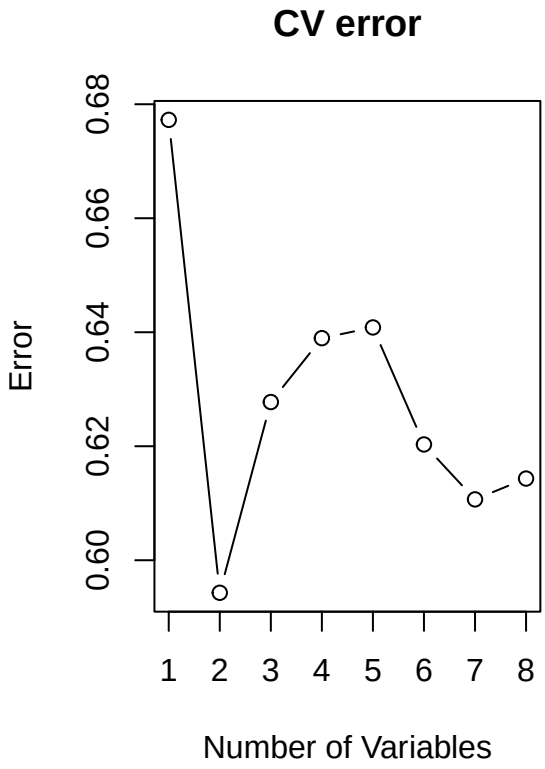
```
## mtry = 1  OOB error = 0.6630186
## Searching left ...
## Searching right ...
## mtry = 2  OOB error = 0.6135698
## Searching left ...
## Searching right ...
## mtry = 3  OOB error = 0.5985894
## Searching left ...
## Searching right ...
## mtry = 4  OOB error = 0.5939249
## Searching left ...
## Searching right ...
## mtry = 5  OOB error = 0.5986813
## Searching left ...
## Searching right ...
## mtry = 6  OOB error = 0.6042925
## Searching left ...
## Searching right ...
## mtry = 7  OOB error = 0.6013676
## Searching left ...
## Searching right ...
## mtry = 8  OOB error = 0.6247654
## Searching left ...
## Searching right ...
```

```
set.seed(1234)
rfres<-rfcv(predictors,topred,cv.fold=10,mtry = identity,
              scale="new",step=-1, ntree=2500)
```

```
##         OOB        CV
## 1 0.6630186 0.6772554
## 2 0.6135698 0.5942965
## 3 0.5985894 0.6277512
## 4 0.5939249 0.6389634
## 5 0.5986813 0.6408419
## 6 0.6042925 0.6203156
```

```
## 7 0.6013676 0.6106790
## 8 0.6247654 0.6143323
```

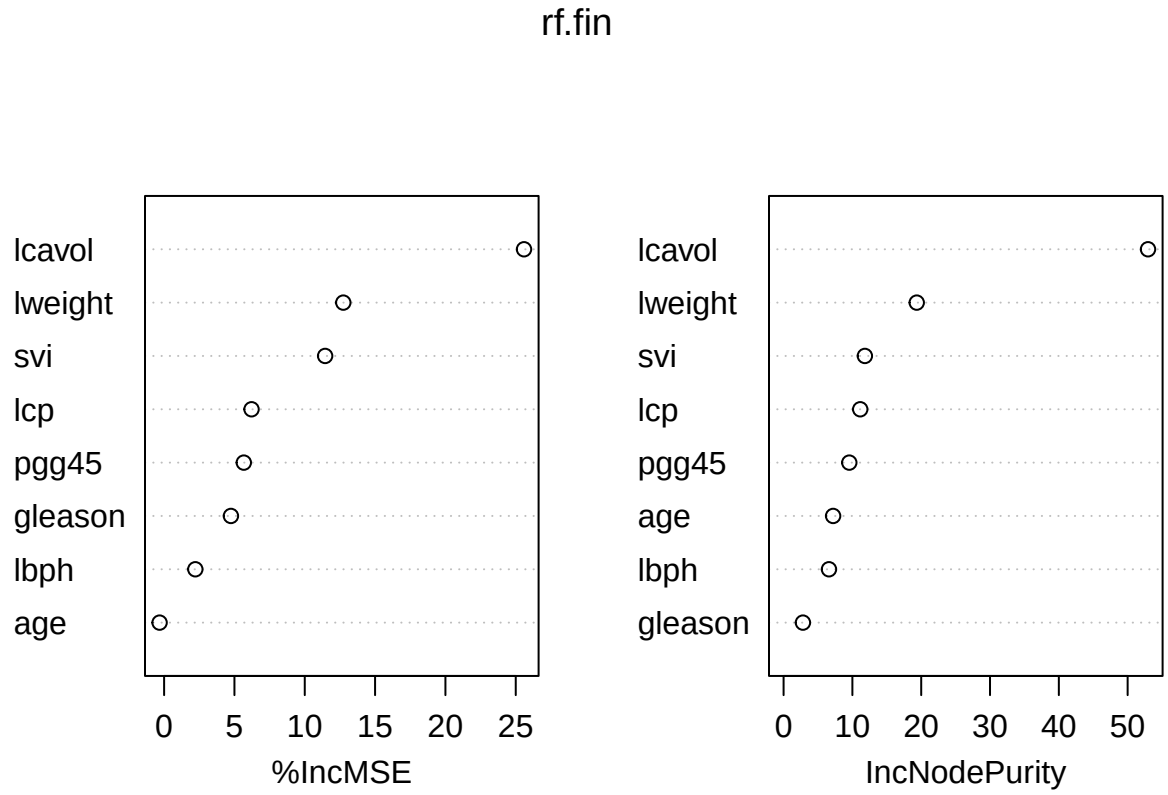





```
## 4
## 4

## 2
## 2

## [1] 0.5939249

## [1] 0.5942965
```

Looking at the graphs that I obtain from the Random Forest cross-validation(rfcv function and tuneRF function) I can see that the number of variables for which the Random Forest regression have the minimum OOB error value is 4 (the minimum error is 0,5939), while the number of variables for which the Random Forest Cross Validation have the minimum error value is 2 (the minimum error is 0,5942). So the minimum error value is reached for different values of m, even if they are quite near. An OOB error prediction can be obtained for each of the n observations of the data set and in the regression setting it represents the MSE; the overall OOB MSE is computed as the mean of the total OOB error of all the observations. Since I used a 5-fold cross validation to calculate the CV error, the difference between the OOB error and the CV error is that they assume different size of learning samples. In 10-fold cross-validation for each sample the learning set is 90%, while the testing set is 10%. In OOB, given an observation, the probability that this observation is not in a given bootstrapped sample is 0,368, so each sample will use approximately 63% of the data, therefore there are 37% of unseen test observation.

I decide to consider 4 as the number of variables to be set as tuning parameter of the Random Forest that I will model on the whole data set. I want now to evaluate what are the most important variables in my model.



The first graph indicates the mean decrease in accuracy in predictions on the out of bag samples when a given variable is permuted, while the second graph indicates the total decrease in node impurity that results from splits over that variable, averaged over all trees. The results indicates that "lcavol", "lweight" and "svi" are useful variables for the prediction(%incMSE>10% and), in particular "lcavol" is by far the variable which has the biggest decrease in node impurity (IncNodePurity>50%).

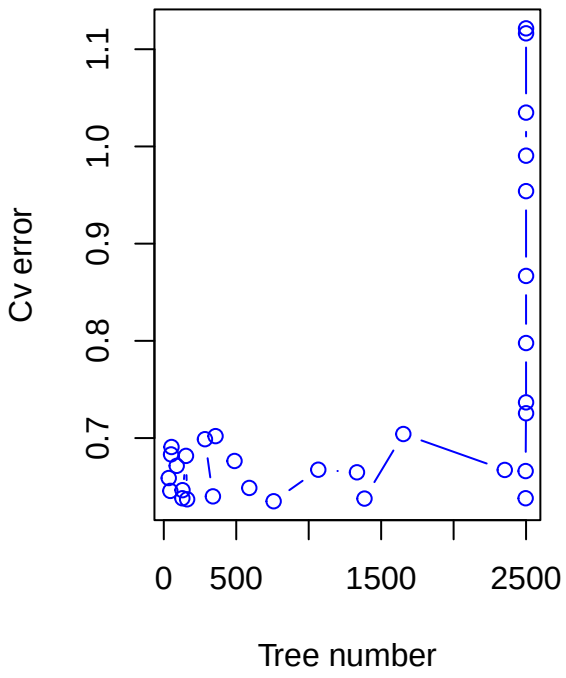
BOOSTING

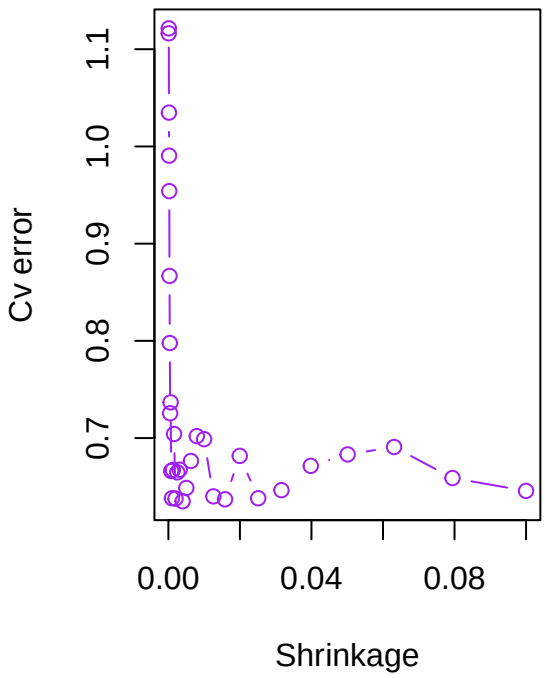
Now I want to evaluate different performance of boosting algorithm (using gbm function) on the whole data set, evaluating it on different values of the shrinkage parameter (learning rate) and for each of them I will execute a cross-validation in order to make a selection of the number of boosting iterations to use in my optimal model.

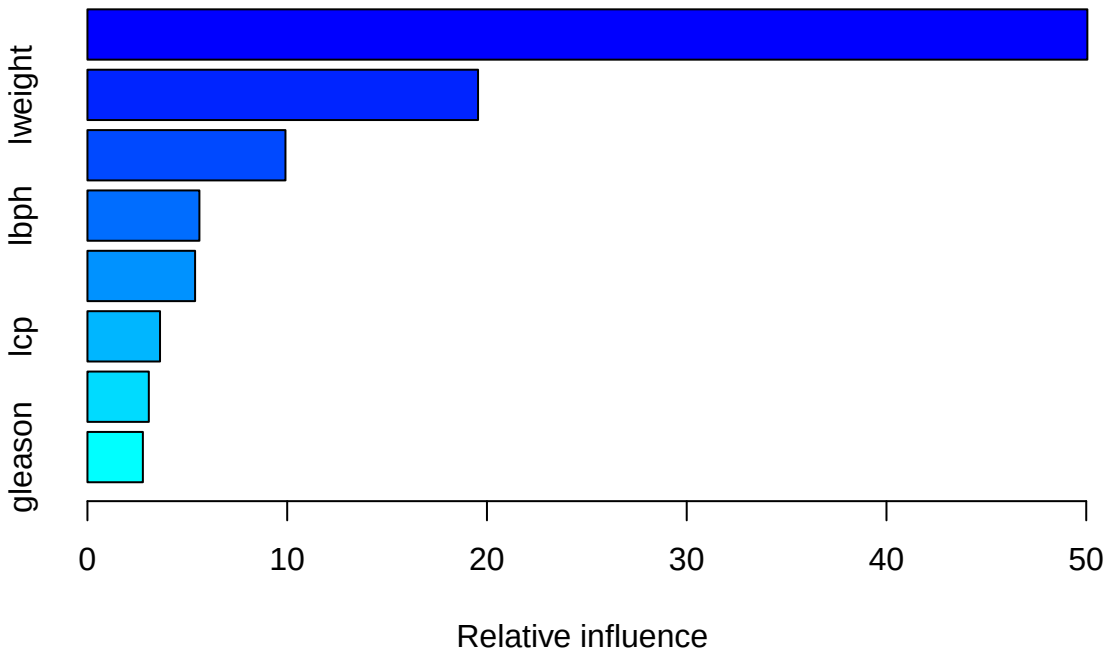
```
library(gbm)
shrinkagevec<- c(10^seq(-4,-1,by=0.1))
nlambda<-length(shrinkagevec)
cverr<-c()
clentree<-c()
boostrainerr<-c()
boosttesterr<-c()
set.seed(1234)
j=0
for (i in shrinkagevec) {
    j<-j+1

    boostedcv<-gbm(lpsa ~ .,data=prostate, distribution="gaussian",
```

```
                n.trees=2500, interaction.depth=4,
                shrinkage=i, verbose=F, cv.fold=5)
    cverr[j]<- min(boostedcv[["cv.error"]])
    clentree[j] <- which.min(boostedcv[["cv.error"]])
    boost_fit <- gbm(lpsa~ ., data=prostate, distribution = "gaussian",
                        n.trees = which.min(boostedcv[["cv.error"]]), shrinkage = i,
                        interaction.depth = 4 )

}
```

```
##                          CVerr Tree number
## 1e-04                1.1163634        2500
## 0.000125892541179417 1.1214825        2500
## 0.00015848931924611 1.0348147        2500
## 0.000199526231496888 0.9904898        2500
## 0.00025118864315095 0.9540290        2500
## 0.000316227766016838 0.8667724        2500
## 0.000398107170553497 0.7977537        2500
## 0.000501187233627273 0.7255067        2500
## 0.000630957344480193 0.7367271        2500
## 0.000794328234724281 0.6660622        2497
## 0.001                0.6381144        2497
## 0.00125892541179417  0.6672201        2353
## 0.00158489319246111  0.7041697        1653
## 0.00199526231496888  0.6377493        1385
## 0.00251188643150958  0.6647787        1333
## 0.00316227766016838  0.6674487        1066
## 0.00398107170553497  0.6350111         758
## 0.00501187233627272  0.6487005         590
## 0.00630957344480193  0.6764298         488
## 0.00794328234724282  0.7019973         357
## 0.01                 0.6988335         284
## 0.0125892541179417   0.6400132         339
## 0.0158489319246111   0.6370135         161
## 0.0199526231496888   0.6816546         153
## 0.0251188643150958   0.6380812         128
## 0.0316227766016838   0.6463885         129
## 0.0398107170553497   0.6714263          89
## 0.0501187233627272   0.6831854          50
## 0.0630957344480194   0.6907937          53
## 0.0794328234724282   0.6588592          34
## 0.1                  0.6456472          45
```


Cv error
0.7
0.8
0.9
1.0
1.1
0
500
1500
2500
Tree number


Cv error
0.7
0.8
0.9
1.0
1.1
0.00
0.04
0.08
Shrinkage



	var	rel.inf
lcavol	lcavol	50.056403
lweight	lweight	19.553950
svi	svi	9.912859
lbph	lbph	5.604910
pgg45	pgg45	5.389294
lcp	lcp	3.633166
age	age	3.073622
gleason	gleason	2.775797

After performing 5-fold cross validation boosting of 2500 trees on the whole dataset for different values of shrinkage I plot the graphs above. As I can see from the first graph the number of tree to be used in the boosting that has the lowest cv.error is 758, even if I can see that a similar value can be reached at 1500 and 2500 trees.Its corresponding cv.error is 0,635. In the second graph instead I can check that the shrinkage value which have the lowest cv.error is 0,0039. The cv.error tends to increase while the shrinkage parameter increase, but I cannot see a specific trend in the graph. The value 758 is a good choice of Tree number(N) because increasing N reduces the error on training set, but setting it too high may lead to over-fitting, so this will lead to a lower MSE when the model is tested on unseen data. With regard to shrinkage it is used for reducing the impact of each additional fitted base-learner(tree), so it penalizes the importance of each consecutive iteration. A shrinkage value of 0,0039 is a slow learn rates and it is good for small data sets. I fit a Boosting using the whole dataset tuning the parameter with the best number of trees and the best value of shrinkage, and I checked the relative influence statistics. I can see that the variables which have an higher relative influence are “lcavol”, “lweight” and “svi”, the same variables of the random forest.

COMPARING THE PERFORMANCE OF ALL THE METHODS.

Here I want to compare the performance of the three methods (cost-complexity decision trees, random forests and boosting) using cross-validation. Now I'm going to fit the model on different training sets using cross validation and to evaluate the performance on different test set. The model complexity will be re-optimized at each choice of the training set (either using another CV or using the OOB error). Now I compute the mean MSE of the training set and of the test set of a cross validation of different pruned tree in which the model complexity is re-optimized at each random sampling. Then I do the same with Random Forest and boosting, re-optimizing at each cycle the tuning parameters.

```
#cross validation pruned tree
library(rsample)
mseregtree<-c()
trainmseregtree<-c()
set.seed(1234)
K<-seq(1:10)
for (i in K){
  my_split <- initial_split(prostate, prop = 0.7)
  canc_train <- training(my_split)
  canc_test <- testing(my_split)
  tree_cancer1 <- tree(lpsa ~ ., canc_train)
  cv.fincancer <- cv.tree(tree_cancer1)
  opt.size1 <- cv.fincancer$size[which.min(cv.fincancer$dev)]
  prune_cancer1 <- prune.tree(tree_cancer1, best=opt.size)
  p<-predict(prune_cancer1, newdata=canc_test)
  mseregtree[i]<-compute_mse(p, canc_test$lpsa)
  pt<-predict(prune_cancer1, newdata=canc_train)
  trainmseregtree[i]<-compute_mse(pt, canc_train$lpsa)
}
mse1mean<-mean(mseregtree) # 0,84
mset1mean<-mean(trainmseregtree)# 0,50

#random forest
mserftree<-c()
mserftreet<-c()
m<-c(1:nvar)
vta<-c()
set.seed(1234)
for (i in K){
  my_split <- initial_split(prostate, prop = 0.7)
  canc_train <- training(my_split)
  canc_test <- testing(my_split)
  predittori<-canc_train[,-9]
  dapred<-canc_train[,9]
  m<-seq(1:nvar)
      for (j in m) {
        t<-tuneRF(predittori, dapred, mtryStart = j,
               ntreeTry=2500,stepFactor=1,
               improve=0.5,trace=FALSE, plot=FALSE, doBest=FALSE, importance=TRUE)
        vta[j]<-t[2]
      }
  u<-min(vta)
  rfcanc <- randomForest(lpsa ~ ., data=canc_train, mtry=u, importance=TRUE, ntree=1000)
  yhat.rf <- predict(rfcanc, newdata=canc_test)
  mserftree[i]<-compute_mse(yhat.rf,canc_test$lpsa)
  yhat.rftr <- predict(rfcanc, newdata=canc_train)
```

```
  mserftreet[i]<-compute_mse(yhat.rftr,canc_train$lpsa)

}
mse2<-mean(mserftree) #0,6142
mse2t<-mean(mserftreet) #0,3015


#boosting not working code.
# prostate$svi<-as.numeric(prostate$svi)
# Z<-seq(1:5)
# msebotr<-c()
# msebotru<-c()
# shrvec<-c(10^seq(-4,-2,by=0.1))
# set.seed(1234)
# for (i in Z){
#   my_split <- initial_split(prostate, prop = 0.7)
#   canc_train <- training(my_split)
#   canc_test <- testing(my_split)
#   cverr3<-c()
#   clentree3<-c()
#   set.seed(1234)
#   j=0
#   for (u in shrvec) {
#       j<-j+1
#       booste<-gbm(lpsa ~ .,canc_train, distribution="gaussian",
#                   n.trees=2000, interaction.depth=4,
#                   shrinkage=0,001, verbose=F, cv.fold=5)
#       cverr3[j]<- min(booste[["cv.error"]])
#       clentree3[j] <- which.min(booste[["cv.error"]])
#
#     }
#   errtreemse<- matrix(nrow=length(shrvec),ncol=2)
#   errtreemse[,1]<- cverr3
#   errtreemse[,2]<-clentree3
#   rownames<-shrvec
#   minntree<-errtreemse[which.min(errtreemse[,1]),2]
#   shrin<-which.min(errtreemse[,1])
#   boost_fi <- gbm(lpsa~ ., data=canc_train, distribution = "gaussian",
#                         n.trees = minntree,shrinkage= shrin, interaction.depth = 4)
#   preds_bo <- predict(boost_fi, newdata=canc_test,n.trees = 500)
#   msebotr[i]<-compute_mse(preds_bo,canc_test$lpsa)
#   preds_bot<-predict(boost_fi, newdata=canc_train,n.trees = 500)
#   msebotru[i]<-compute_mse(preds_bo,canc_train$lpsa)
#
# }
# mean(msebotr)
# mean(msebotru)
library(gbm)
Z<-seq(1:10)
msebotr<-c()
msebotru<-c()
set.seed(1234)
for (i in Z){
```

```
  my_split <- initial_split(prostate, prop = 0.7)
  canc_train <- training(my_split)
  canc_test <- testing(my_split)
  canc_train<-as.data.frame(canc_train)
  boost_fi <- gbm(lpsa~ ., data=canc_train , distribution = "gaussian",
                        n.trees = minntree,shrinkage= s, interaction.depth = 4)
  preds_bo <- predict(boost_fi, newdata=canc_test ,n.trees = 500)
  msebotr[i]<-compute_mse(preds_bo,canc_test$lpsa)
  preds_bot<-predict(boost_fi, newdata=canc_train,n.trees = 500)
  msebotru[i]<-compute_mse(preds_bot,canc_train$lpsa)

}
mean(msebotr)
```

```
## [1] 0.7372205
```

```
mean(msebotru)
```

```
## [1] 0.5268341
```

The mean MSE of the pruned tree after the cross validation tested on the test set has a value of 0.84, while tested on the training set has a value of 0,50. The mean MSE of the Random Forest after the cross-validation tested on the test set has a value of 0,6142, while tested on the training set has a value of 0,3015 Since the code I tried to use for tuning parameter on each sample for the cross validation didn't work, I used cross validation boosting with the parameters choosen in the boosting performed in the previous point : ntree=758, shrinkage value=0,0039. The mean MSE of Boosting after the cross-validation tested on the test set has a value of 0,7372, while tested on the training set has a value of 0,5268 From this results we can say that the model that performs better on the prostate dataset in order to predict the 'lpsa' variable is Random Forest. A decision tree that is very deep, like the first decision tree I performed, tends to overfit the data leading to low bias but high variance. Pruning is a suitable approach used in decision trees to reduce overfitting, in fact the performance of the pruned tree has improved the accuracy over the first decision tree. Regarding the performance of the other two methods, the main difference between random forests and gradient boosting lies in how the decision trees are created and aggregated. Unlike random forests, the decision trees in gradient boosting are built additively; it means that each decision tree is built one after another. This is in contrast to random forests which build and calculate each decision tree independently. In random forests, the results of decision trees are aggregated at the end of the process. Gradient boosting doesn't do this and instead aggregates the results of each decision tree along the way to calculate the final result. Random Forest has the lowest training error, and a quite high test error, the model is overfitting the training data, while the boosting model is not performing really well not in training neither in the test data because I did not retune for each sampling the 2 parameters of shrinkage and number of trees, so its performance can be improved. An important thing that came out from all the models I used is that `lcavol`(log(cancer volume in cm3), `lweight`(log(prostate weight in g)) and `svi`(seminal vesicle invasion (1 = yes, 0 = no)) are the most important variables to determine the values of `lpsa`(ng/ml and log scaled).