Homework 2

Lorenzo Bocchi

2022-05-16

Introduction

This analysis on cancer data will investigate the correlation between a 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

We start by loading the data from the 'prostate.csv' file:

```
c_data <- read.csv('./prostate.csv')
summary(c_data)
```

```
##      lcavol           lweight           age             lbph        
##  Min.   :-1.3471   Min.   :2.375   Min.   :41.00   Min.   :-1.3863  
##  1st Qu.: 0.5128   1st Qu.:3.376   1st Qu.:60.00   1st Qu.:-1.3863  
##  Median : 1.4469   Median :3.623   Median :65.00   Median : 0.3001  
##  Mean   : 1.3500   Mean   :3.629   Mean   :63.87   Mean   : 0.1004  
##  3rd Qu.: 2.1270   3rd Qu.:3.876   3rd Qu.:68.00   3rd Qu.: 1.5581  
##  Max.   : 3.8210   Max.   :4.780   Max.   :79.00   Max.   : 2.3263  
##       svi              lcp             gleason          pgg45       
##  Min.   :0.0000   Min.   :-1.3863   Min.   :6.000   Min.   :  0.00  
##  1st Qu.:0.0000   1st Qu.:-1.3863   1st Qu.:6.000   1st Qu.:  0.00  
##  Median :0.0000   Median :-0.7985   Median :7.000   Median : 15.00  
##  Mean   :0.2165   Mean   :-0.1794   Mean   :6.753   Mean   : 24.38  
##  3rd Qu.:0.0000   3rd Qu.: 1.1787   3rd Qu.:7.000   3rd Qu.: 40.00  
##  Max.   :1.0000   Max.   : 2.9042   Max.   :9.000   Max.   :100.00  
##       lpsa        
##  Min.   :-0.4308  
##  1st Qu.: 1.7317  
##  Median : 2.5915  
##  Mean   : 2.4784  
##  3rd Qu.: 3.0564  
##  Max.   : 5.5829
```

We can check if any data is missing:

```
any(is.na(c_data))
```

```
## [1] FALSE
```

Decision tree

Since nothing is missing, we can proceed to create the training and test sets. For that, we will use caret's 'createDataPartition' to get a randomly sampled partition of the dataset:

```
set.seed(1)
sample <- caret::createDataPartition(c_data$lpsa, p=0.8)
train_data <- c_data[sample$Resample1, ]
test_data <- c_data[-sample$Resample1, ]
x_train <- model.matrix(lpsa ~ ., data = train_data)[, -1]
x_test <- model.matrix(lpsa ~ ., data = test_data)[, -1]
y_train <- train_data$lpsa
y_test <- test_data$lpsa
```

Then, we can fit the model:

```
set.seed(1)
tree_cancer <- tree(lpsa ~ ., c_data)
summary(tree_cancer)
```

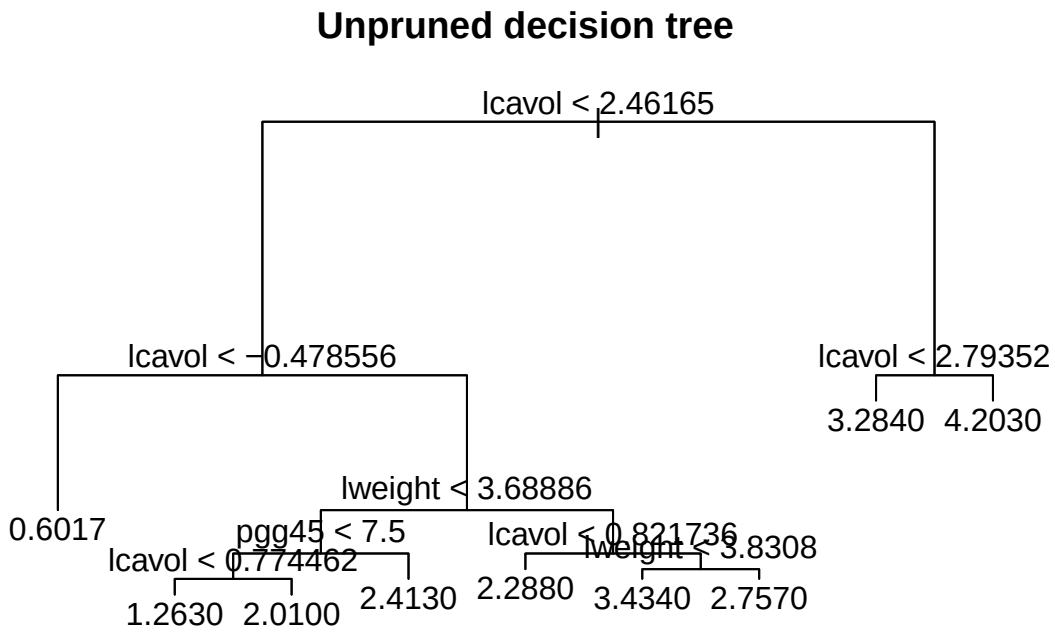
```
##
## Regression tree:
## tree(formula = lpsa ~ ., data = c_data)
## Variables actually used in tree construction:
## [1] "lcavol"  "lweight" "pgg45"
## Number of terminal nodes:  9
## Residual mean deviance:  0.4119 = 36.24 / 88
## Distribution of residuals:
##      Min.   1st Qu.    Median    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

The summary shows that only three variables were used to create the tree, namely:

- lcavol
- lweight
- pgg45

We can visualize the tree by plotting it:

```
plot(tree_cancer)
text(tree_cancer, pretty = 0)
title(main = "Unpruned decision tree")
```

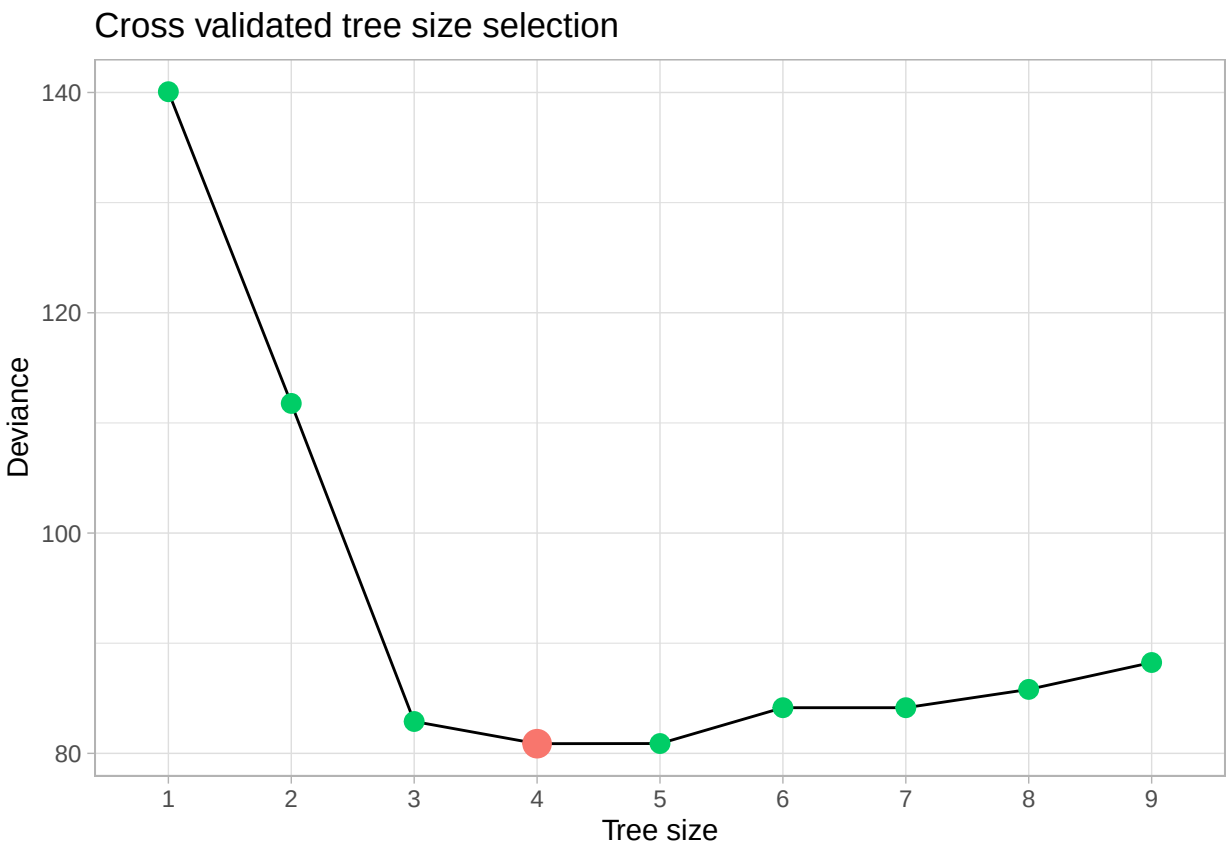


This tree predicts that higher of lcavol (>= 2.79352), higher pgg45 (>= 7.5) and an lweight between 3.69 and 3.83 give a higher level of lpsa. In particular, a very high or very low value for lcavol give an immediate prediction, while intermediate values are also influenced based on other predictors.

Cross validation

Now we cross validate the tree and compare the results. To do that, we will use the 'cv.tree' function, which selects the best tree based on the deviance obtained with a certain tree size:

```
set.seed(1)
cv_tree <- cv.tree(tree_cancer)
ggplot(NULL, aes(x=cv_tree$size, y=cv_tree$dev)) +
  theme_light() +
  geom_line() +
  geom_point(col="springgreen3", size=3) +
  geom_point(data = NULL,
             aes(x=cv_tree$size[which.min(cv_tree$dev)],
                 y=min(cv_tree$dev),
                 colour="red",
                 size = 3)) +
  labs(title="Cross validated tree size selection",
       y="Deviance",
       x="Tree size") +
  guides(size = "none", colour = "none") +
  scale_x_discrete(limits=1:9, labels = c(1:9))
```

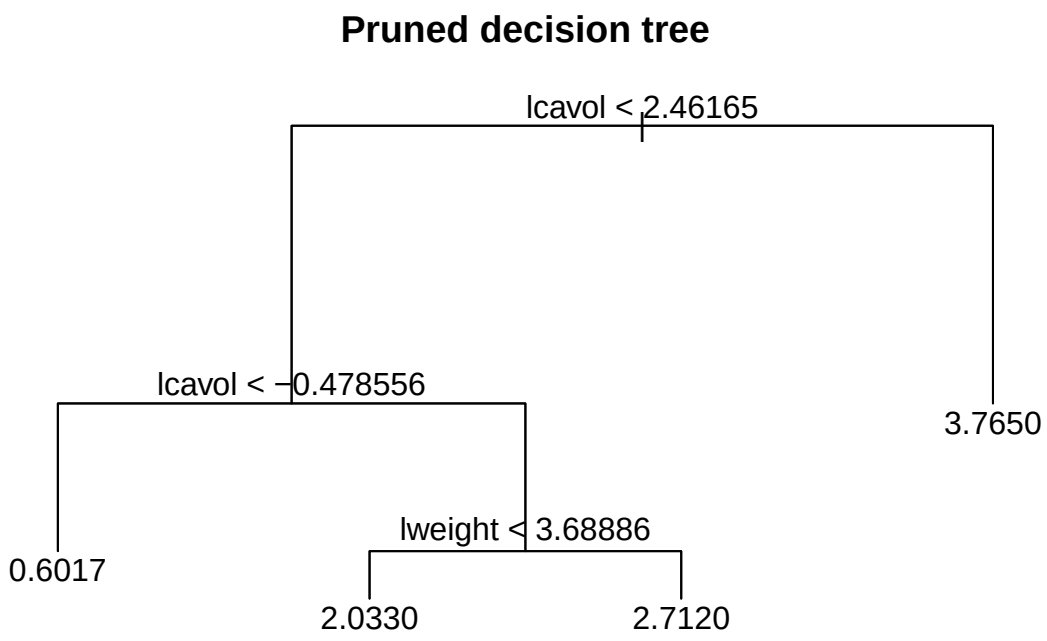


The cross validation tells us that a tree with size of four gives the lowest deviance and therefore the best result.

Pruning the tree

We can prune the tree with the best parameter that we just found and visualize it:

```
pruned_tree <- prune.tree(tree_cancer, best=4)
plot(pruned_tree)
text(pruned_tree, pretty=0)
title(main="Pruned decision tree")
```



This tree predicts that higher lcavol (>= 2.46165) and a higher lweight (>= 3.68886) give a higher level of lpsa. In this case, pgg45 is not a considered predictor. Another thing to notice is that lcavol remains a strong predictor which gives a direct prediction if its value is very high or very low.

Result

Finally, we refit the model only on the train data and make the predictions to calculate both the MSE and RMSE:

```
set.seed(1)
tree_cancer <- tree(lpsa ~ ., train_data)
pruned_tree <- prune.tree(tree_cancer, best=4)
yHat <- predict(pruned_tree, test_data)
mse_decision <- mean((yHat - test_data$lpsa)^2)
paste("MSE:", mse_decision)
```

```
## [1] "MSE: 1.26696430704869"
```

```
paste("RMSE:", sqrt(mse_decision))
```

```
## [1] "RMSE: 1.12559509018505"
```

The test MSE for this decision (regression) tree is ~1.266964 and the RMSE ~1.125595. This means that the predicted lpsa is within ~1.125595 the median value.

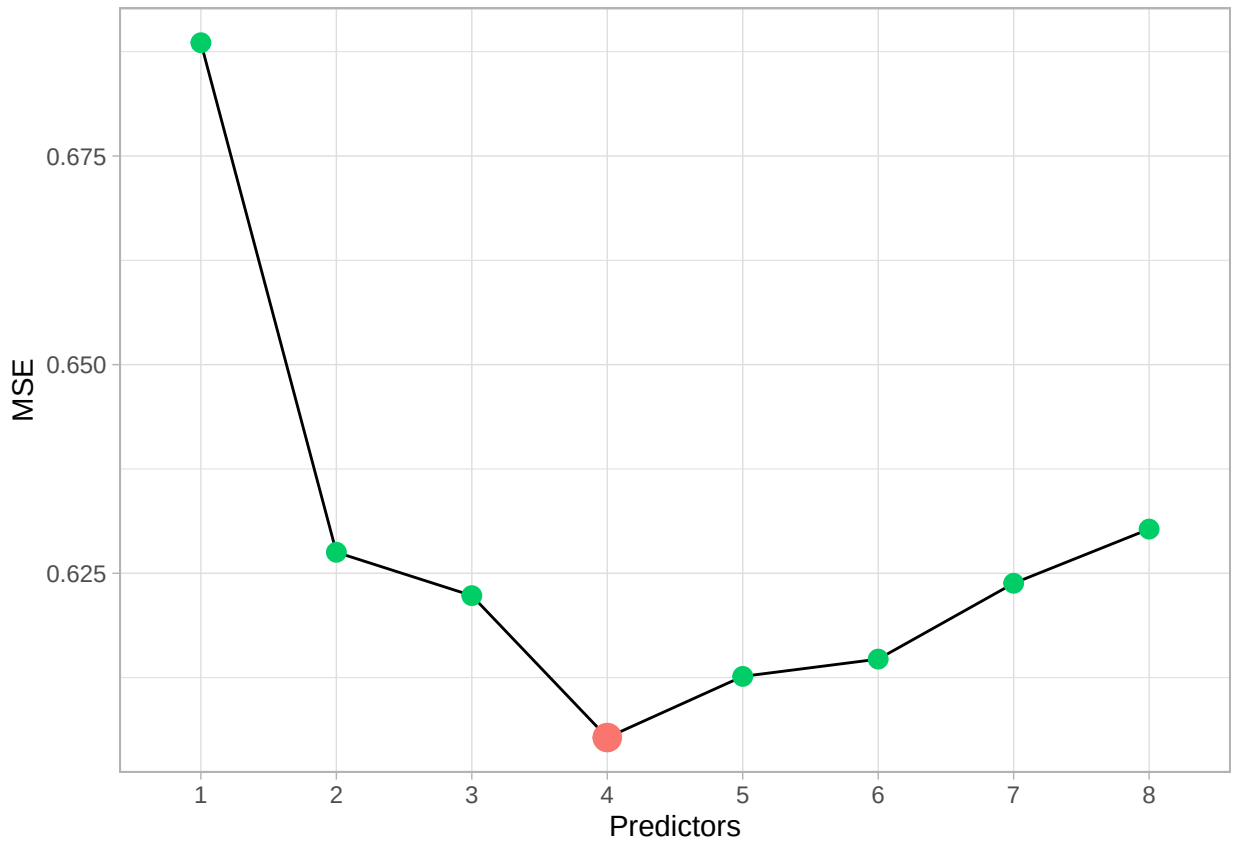
Random forest

After a decision tree, we now consider a random forest. Let's start by fitting the model. To do the cross validation, we will use caret's 'createFolds' function which allows us to specify the number of folds. Since the dataset is relatively small, we will do a 5-fold validation:

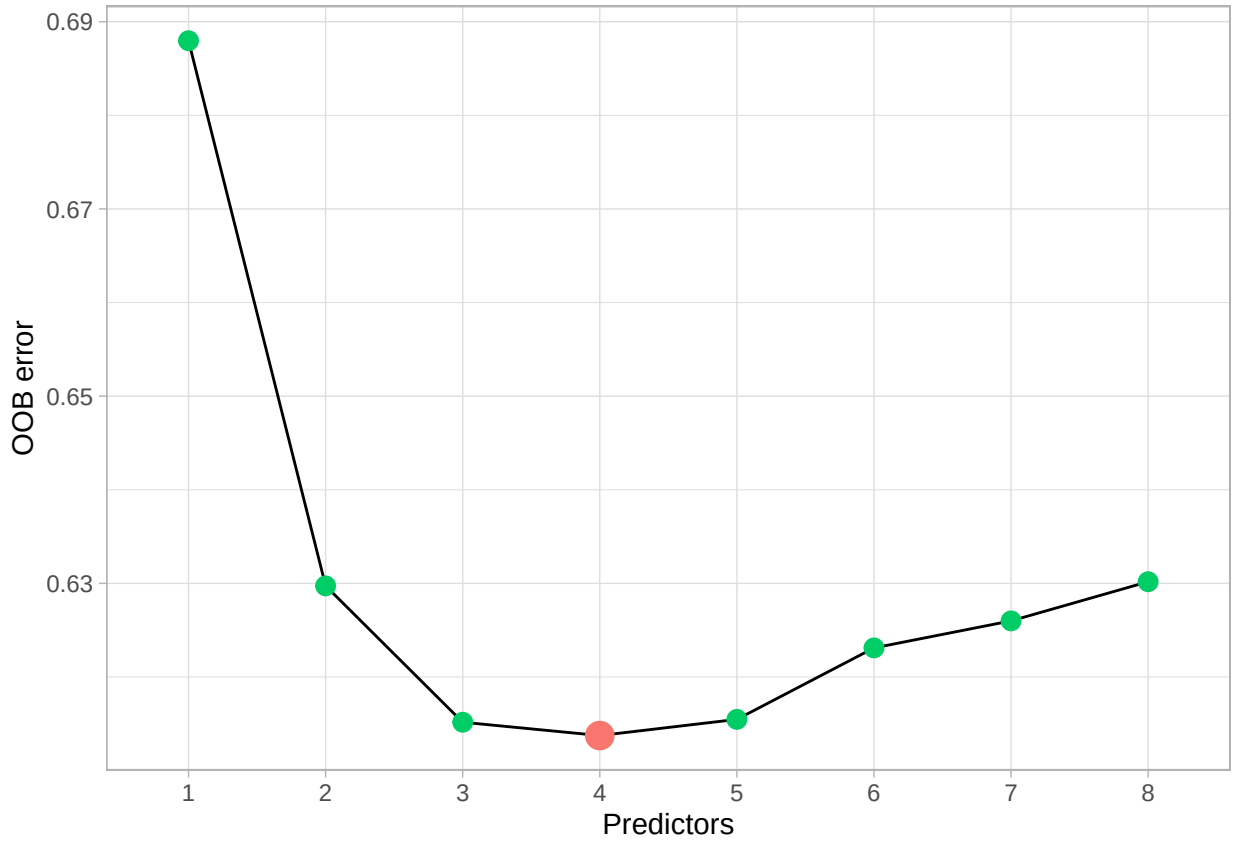
```
set.seed(1)
n_pred <- ncol(c_data) - 1

err_matrix <- matrix(0, nrow = n_pred, ncol = 2)

folds <- createFolds(c_data$lpsa, k = 5)
MSE <- c()
OOB <- c()
for(i in 1:n_pred){
  for(f in folds){
    rf <- randomForest(lpsa ~ .,
                       data = c_data[-f, ],
                       mtry = i,
                       importance = TRUE)

    yHat_rf <- predict(rf, newdata=c_data[f, ])
    MSE <- c(MSE, mean((yHat_rf - c_data[f, ]$lpsa)^2))
    OOB <- c(OOB, mean((rf$predicted - c_data[-f,]$lpsa)^2))
  }
  err_matrix[i, 1] <- mean(MSE)
  err_matrix[i, 2] <- mean(OOB)
  MSE <- c()
  OOB <- c()
}
```

Let's plot the results:

```
ggplot(NULL, aes(x=c(1:8), y=err_matrix[,1])) +
  theme_light() +
  geom_line() +
  geom_point(col="springgreen3", size=3) +
  geom_point(data = NULL, aes(x=c(1:8)[which.min(err_matrix[,1])],
                              y=min(err_matrix[,1]),
                              colour="red",
                              size = 3)) +
  labs(y="MSE", x="Predictors") +
  guides(size = F, colour = F) +
  scale_x_discrete(limits=1:8, labels = c(1:8))
```




```
ggplot(NULL, aes(x=c(1:8), y=err_matrix[,2])) +
  theme_light() +
  geom_line() +
  geom_point(col="springgreen3", size=3) +
  geom_point(data = NULL, aes(x=c(1:8)[which.min(err_matrix[,2])],
                              y=min(err_matrix[,2]),
                              colour="red",
                              size = 3)) +
  labs(y="OOB error", x="Predictors") +
  guides(size = F, colour = F) +
  scale_x_discrete(limits=1:8, labels = c(1:8))
```




```
print(paste(paste0("Best MSE: ", min(err_matrix[,1]), "."),
            "Best number of predictors:", which.min(err_matrix[,1])))
```

```
## [1] "Best MSE: 0.605315462178251. Best number of predictors: 4"
```

```
print(paste(paste0("Best OOB error: ", min(err_matrix[,2]), "."),
            "Best number of predictors:", which.min(err_matrix[,2])))
```

```
## [1] "Best OOB error: 0.613745212534146. Best number of predictors: 4"
```

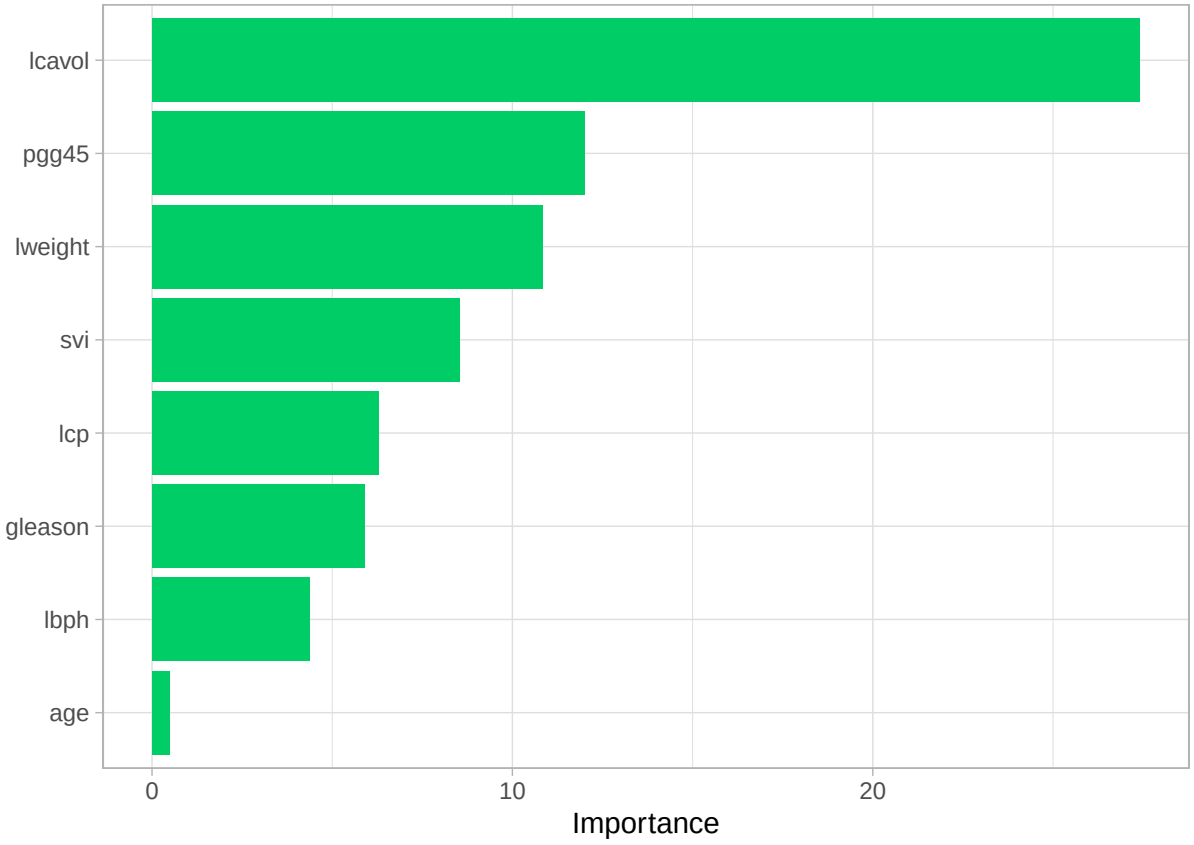
```
colnames(err_matrix) <- c("MSE", "OOB error")
err_matrix
```

```
##            MSE OOB error
## [1,] 0.6885735 0.6879685
## [2,] 0.6275178 0.6297347
## [3,] 0.6223219 0.6151682
## [4,] 0.6053155 0.6137452
## [5,] 0.6126424 0.6154720
## [6,] 0.6147152 0.6231050
## [7,] 0.6238021 0.6259936
## [8,] 0.6302943 0.6301774
```

Adjusting predictors

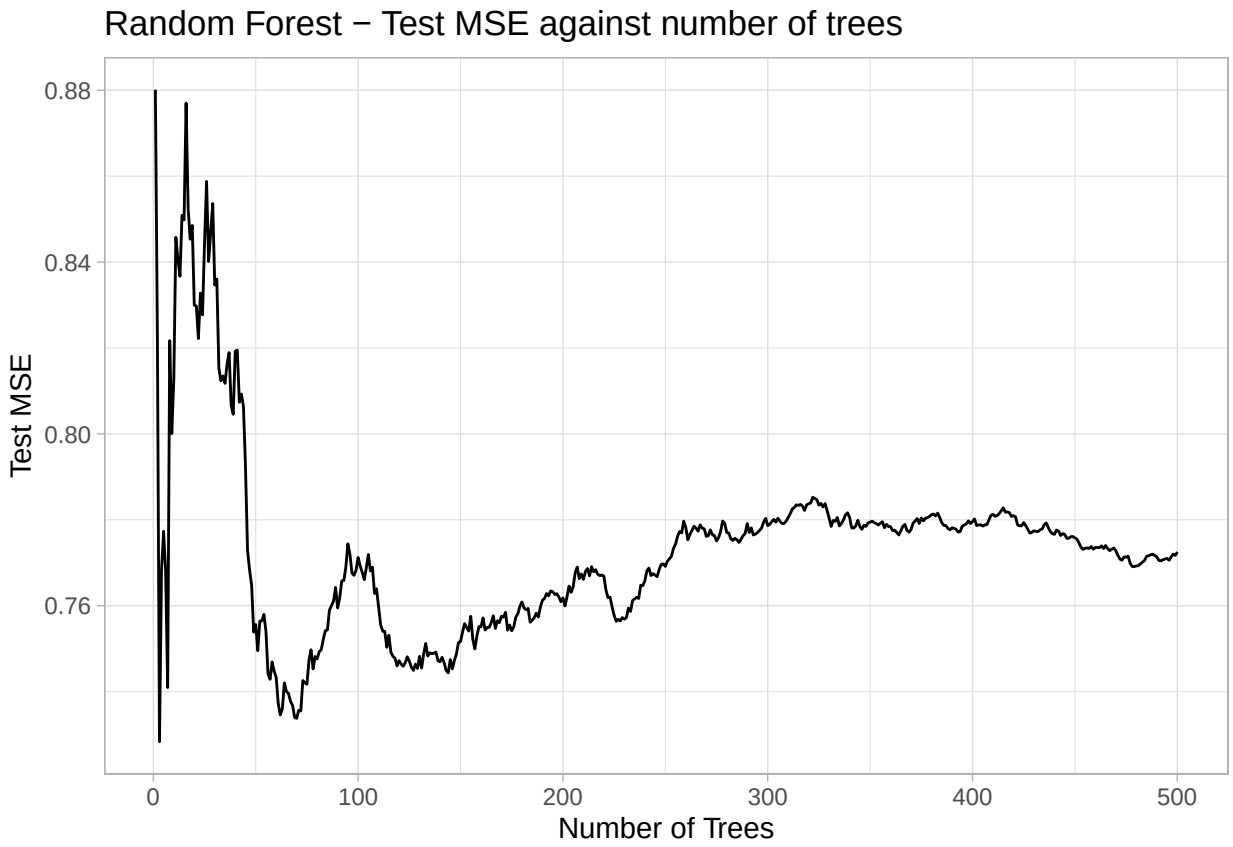
Both the MSE and OOB error are very close for all numbers of predictors and, according to them, the best model is the one with 4 variables. We can proceed to create the model using the training and testing dataset splits and analyze it more in depth:

```
set.seed(1)
rf <- randomForest(x_train,
                   y_train,
                   xtest = x_test,
                   ytest = y_test,
                   mtry = 4,
                   importance = TRUE)
vip(rf, aesthetics = c(fill = "springgreen3")) +
  theme_light()
```



The most important predictor is lcavol, followed by pgg45 and lweight. Both lcavol and lweight were also prominent in the previously analyzed decision tree, while pgg45 is a new addition. Let's also visualize the error:

```
ggplot(NULL, aes(x=c(1:500), y=rf$test$mse)) +
  theme_light() +
  geom_line() +
  labs(title="Random Forest - Test MSE against number of trees",
       y="Test MSE",
       x="Number of Trees")
```



Result

We can proceed to make our predictions on the test data:

```
yHat_rf <- rf$test$predicted
mse_rf <- mean((yHat_rf - test_data$lpsa)^2)
paste("MSE:", mse_rf)
```

```
## [1] "MSE: 0.77251813599546"
```

```
paste("Square root of MSE:", sqrt(mse_rf))
```

```
## [1] "Square root of MSE: 0.87893010870914"
```

Here, the result is better than the decision tree. The test MSE is ~0.772518 and the square root of it is ~0.878930. This means that the predicted lpsa is within ~0.878930 the median value.

Boosted regression tree

The last method we will use is the boosted regression tree. Once again, we start by fitting the model on the whole dataset. We will also do a cross validation to find the best number of boosting iteration by setting the parameter 'cv.folds' of the gbm function to 5, so to have the same number of folds as used for the random forest. Regarding the interaction depth, generally smaller trees are sufficient since in boosting the growth of a tree takes into account the previously grown trees. But, according to the book "An Introduction to Statistical Learning" by Gareth James et al., generally a depth of 1 or 2 works best, while according to Friedman [1] a range of 2<=d<=8 works best. Therefore, to accomodate both opinions, we will validate the performance with a depth for the trees ranging between 1 and 8. The number of trees preferred by the studies in the same paper ranged between 100 and 500, while other studies [2] recommend to fit at least 1000 trees as

[1] https://jerryfriedman.su.domains/ftp/trebst.pdf
[2] https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/j.1365-2656.2008.01390.x

a rule of thumb. We will choose 1000 as the upper limit and then eventually raise it if necessary:

```
set.seed(1)
mat_boost <- matrix(0, nrow=8, ncol = 3)
for(d in 1:8){
  boost <- gbm(lpsa ~ .,
               data = c_data,
               distribution = "gaussian",
               n.trees = 1000,
               interaction.depth = d,
               cv.folds = 5,
               verbose = F,
               n.cores = 4)
  mat_boost[d,1] <- min(boost$cv.error)
  mat_boost[d,2] <- which.min(boost$cv.error)
  mat_boost[d,3] <- gbm.perf(boost,
                             plot.it=F,
                             oobag.curve=F,
                             method="OOB")
}
print(paste("The cross validated MSE suggests",
            mat_boost[which.min(mat_boost[,1]),2],
            "iterations (trees) and a depth of",
            which.min(mat_boost[,1])))
```

```
## [1] "The cross validated MSE suggests 35 iterations (trees) and a depth of 6"
```

```
print(paste("The OOB error suggests",
            mat_boost[which.min(mat_boost[,1]),3],
            "iterations (trees) and a depth of",
            which.min(mat_boost[which.min(mat_boost[,1]),2])))
```

```
## [1] "The OOB error suggests 23 iterations (trees) and a depth of 1"
```

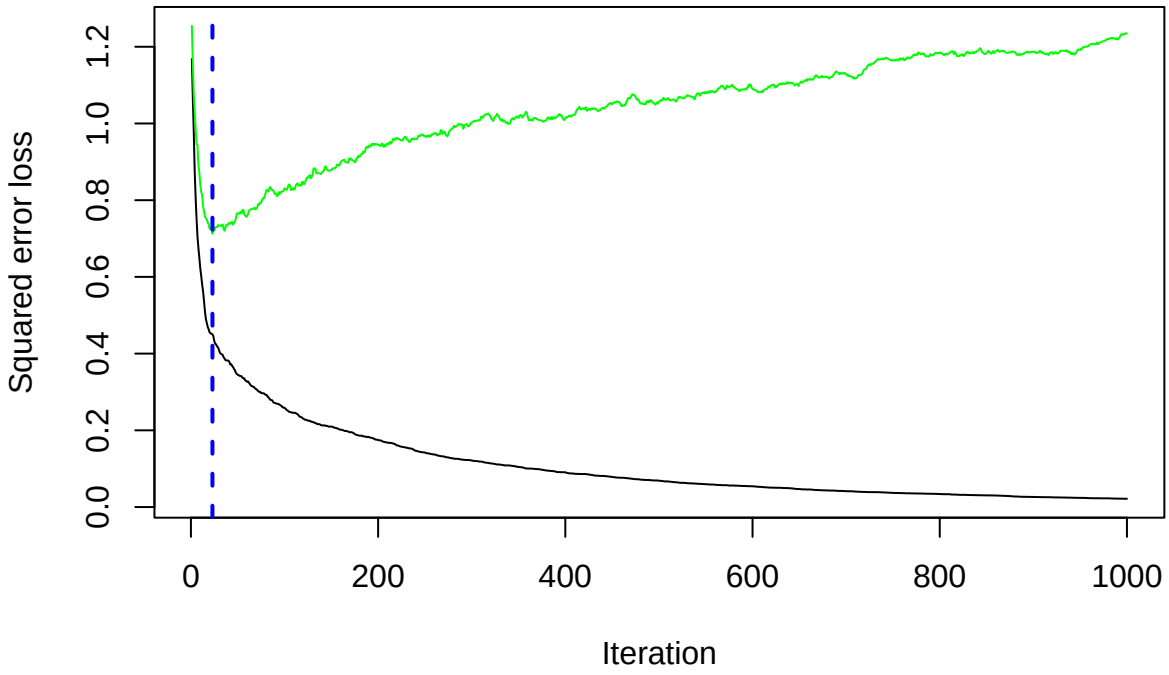
```
colnames(mat_boost) <- c("MSE|", "Iterations (MSE)|", "Iterations (OOB)")
mat_boost
```

```
##           MSE| Iterations (MSE)| Iterations (OOB)
## [1,] 0.7048545                23               26
## [2,] 0.6310488                26               23
## [3,] 0.6561338                32               23
## [4,] 0.6302493                36               23
## [5,] 0.6414287                27               23
## [6,] 0.6180743                35               23
## [7,] 0.8115481                22               22
## [8,] 0.6293219                33               22
```

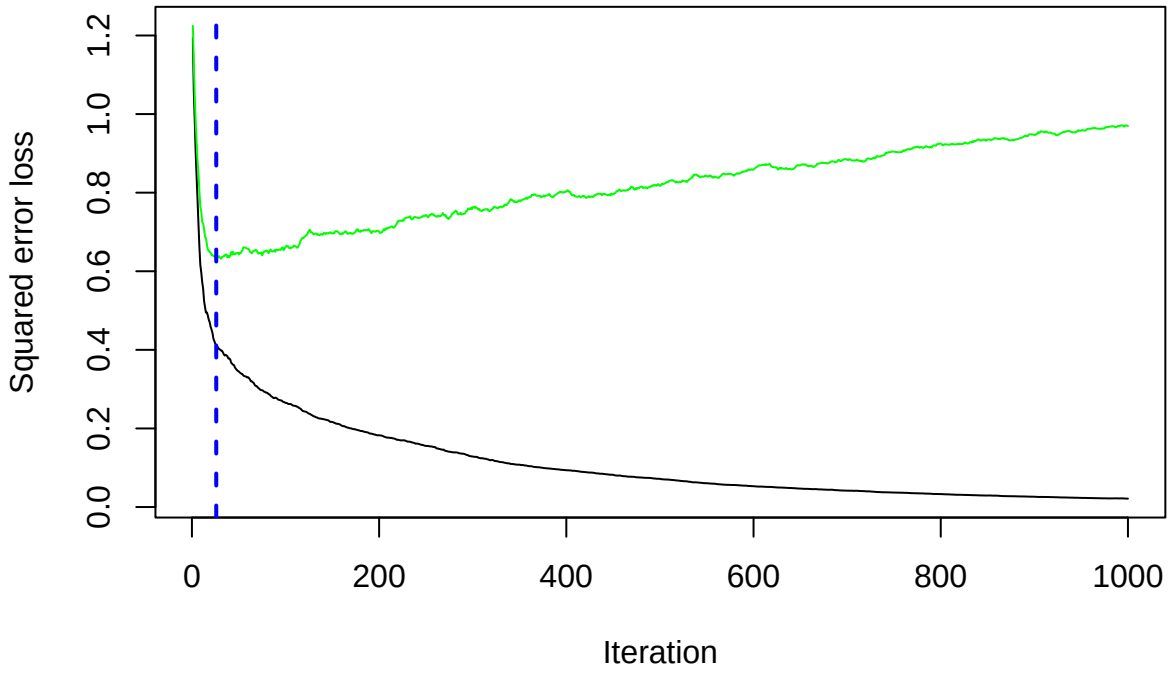
Both using cross validation and OOB estimation, we get two different suggestions. The lowest cv error is obtained with 35 iterations and a depth of 6, while the best OOB error is obtained with 23 trees and a depth of 7. We can actually check the cross validation error by plotting it with the 'gbm.perf' function and compare the value with the OOB selected one:

```
set.seed(1)
mat_boost <- matrix(0, nrow=2, ncol = 3)
boost <- gbm(lpsa ~ .,
             data = c_data,
             distribution = "gaussian",
```

```
             n.trees = 1000,
             interaction.depth = 6,
             cv.folds = 5,
             verbose = F,
             n.cores = 4)
mat_boost[1,1] <- min(boost$cv.error)
mat_boost[1,2] <- gbm.perf(boost,
                           plot.it=F,
                           oobag.curve=F,
                           method="OOB")
mat_boost[1,3] <- gbm.perf(boost, method="cv")
```




```
boost <- gbm(lpsa ~ .,
             data = c_data,
             distribution = "gaussian",
             n.trees = 1000,
             interaction.depth = 7,
             cv.folds = 5,
             verbose = F,
             n.cores = 4)
mat_boost[2,1] <- min(boost$cv.error)
mat_boost[2,2] <- gbm.perf(boost,
                           plot.it=F,
                           oobag.curve=F,
                           method="OOB")
mat_boost[2,3] <- gbm.perf(boost, method="cv")
```




```
print(paste("MSE selected model -> MSE:", mat_boost[1,1],
            "CV iterations:", mat_boost[1,3],
            "OOB iterations:", mat_boost[1,2]))
```

```
## [1] "MSE selected model -> MSE: 0.712752643718914 CV iterations: 23 OOB iterations: 22"
```

```
print(paste("OOB selected model -> MSE:", mat_boost[2,1],
            "CV iterations:", mat_boost[2,3],
            "OOB iterations:", mat_boost[2,2]))
```

```
## [1] "OOB selected model -> MSE: 0.631048769643227 CV iterations: 26 OOB iterations: 22"
```

The OOB selected model with a depth of 7 has a clearly lower MSE cross validated error. Therefore, we do a final comparison of the model using a depth of 7 and a number of trees of both 26 and 22 to check which one gives the lower MSE:

```
set.seed(1)
first_mse <- 0
second_mse <- 0
boost <- gbm(lpsa ~ .,
             data = c_data,
             distribution = "gaussian",
             n.trees = 26,
             interaction.depth = 7,
             cv.folds = 5,
             verbose = F,
             n.cores = 4)
first_mse <- boost$cv.error[length(boost$cv.error)]
```

```
boost <- gbm(lpsa ~ .,
             data = c_data,
             distribution = "gaussian",
             n.trees = 22,
             interaction.depth = 7,
             cv.folds = 5,
             verbose = F,
             n.cores = 4)
second_mse <- boost$cv.error[length(boost$cv.error)]

print(paste("26 trees -> MSE:", first_mse))
```

```
## [1] "26 trees -> MSE: 0.728978156920438"
```

```
print(paste("22 trees -> MSE:", second_mse))
```

```
## [1] "22 trees -> MSE: 0.699213790624418"
```

Since using 22 trees gives a better cross validated MSE, we will keep that as the better iteration parameter.

Adjusting the parameters

Having found the best parameters, we can proceed to fit the model on the training data and calculate the MSE and RMSE on the test data:

```
#we don't need cross validation this time
set.seed(1)
boost <- gbm(lpsa ~ .,
             data = train_data,
             distribution = "gaussian",
             n.trees = 22,
             interaction.depth = 7,
             n.cores = 4,
             verbose = F)
yHat_boost <- predict(boost, test_data)
mse_boost <- mean((yHat_boost - test_data$lpsa)^2)
paste("MSE-selected model -> MSE:", mse_boost,
      "RMSE:", sqrt(mse_boost))
```

```
## [1] "MSE-selected model -> MSE: 0.604065507849782 RMSE: 0.777216512851973"
```

In this case, we have a better result than both the random forest and the decision tree. We have an MSE of ~0.604066 and an RMSE of ~0.777217. This means that the predicted lpsa is within ~0.777217 the median value.

Performance comparison

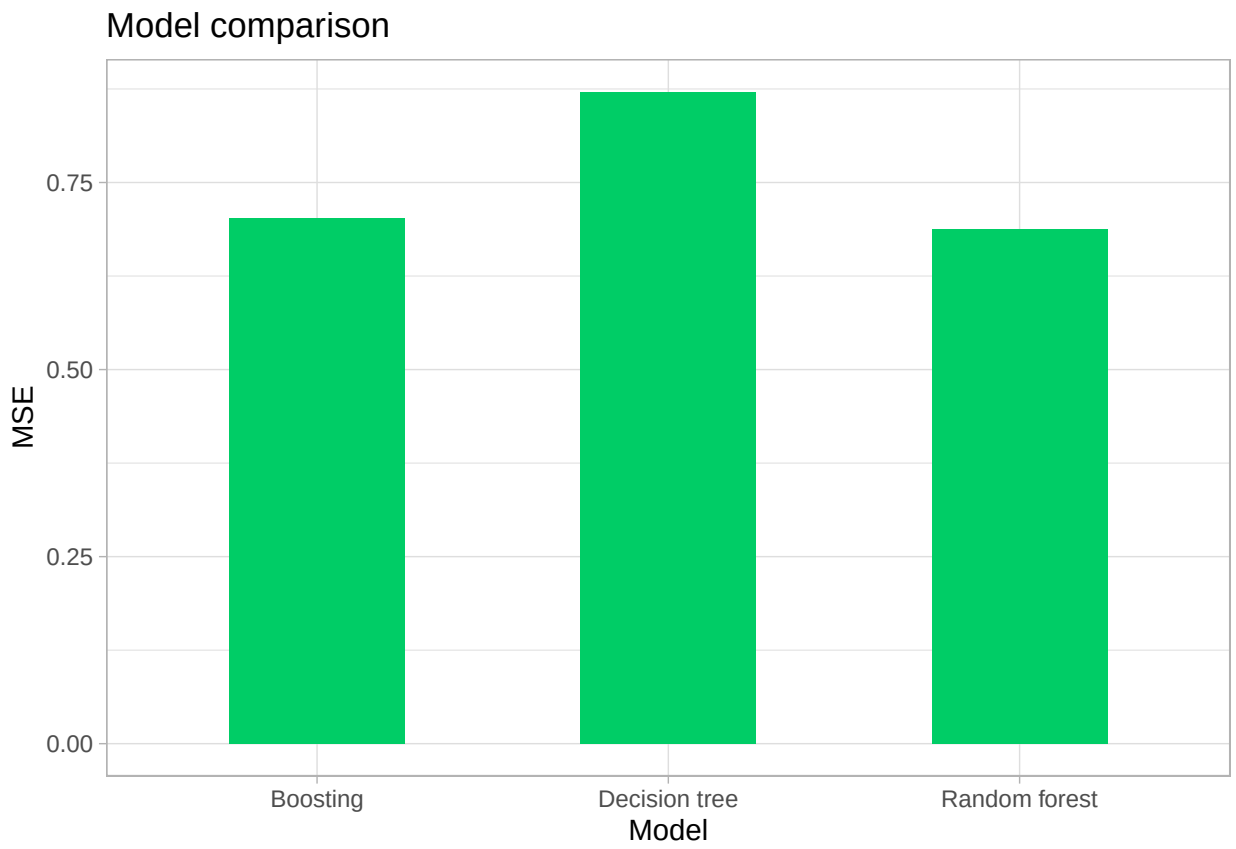
So far, we have only compared the models standalone. Now we will do a nested cross validation and optimize each model in the outer layer as the inner layer gives us the best parameter. For every outer fold, we create 5 inner folds on the remaining data and cross validate with these folds. After the inner folds have ended, we test the best parameter against the outer fold and save the MSE:

```
set.seed(1)
outerFolds <- createFolds(c_data$lpsa, k = 5)
decision_mse <- c()
rf_mse <- c()
boosting_mse <- c()
trees_boost <- c()
r <- 1
for(f in outerFolds){
  r2 <- 1
  innerFolds <- createFolds(c_data[-f,]$lpsa, k = 5)
  innerParam <- matrix(0, nrow = length(innerFolds), ncol = 2)
  innerParamRF <- matrix(0, nrow = length(innerFolds), ncol = 2)
  innerParamBoosting <- matrix(0, nrow = length(innerFolds), ncol = 2)
  for(innF in innerFolds){
    # Decision tree
    tree_nested <- tree(lpsa ~ ., c_data[-f,][-innF,])
    innerMSE_pruning <- c()
    preds <- c()
    for(i in 2:(n_pred+1)){
      pruned_tree_nested <- prune.tree(tree_nested, best=i)
      yHat_nd <- predict(pruned_tree_nested, c_data[-f,][innF,])
      innerMSE_pruning <- c(innerMSE_pruning,
                            mean((yHat_nd - c_data[-f,][innF,]$lpsa)^2))
    }

    innerParam[r2, 1] <- min(innerMSE_pruning)
    innerParam[r2, 2] <- which.min(innerMSE_pruning)

    # Random forest
    mse_rf_inner <- c()
    for(i in 1:n_pred){
      rf <- randomForest(lpsa ~ .,
                         data = c_data[-f,][-innF, ],
                         mtry = i,
                         importance = TRUE)

      yHat_rf <- predict(rf, newdata=c_data[-f,][innF, ])
      mse_rf_inner <- c(mse_rf_inner, mean((yHat_rf - c_data[-f,][innF, ]$lpsa)^2))
    }
    innerParamRF[r2, 1] <- min(mse_rf_inner)
    innerParamRF[r2, 2] <- which.min(mse_rf_inner)

    # Boosting
    boosting_inner <- matrix(0, nrow = 8, ncol = 2)
    for(d in 1:8){
      boost <- gbm(lpsa ~ .,
                   data = c_data[-f,][-innF,],
                   distribution = "gaussian",
                   n.trees = 1000,
                   interaction.depth = d,
                   n.cores = 4,
                   verbose = F)
      trees_boost <- c(trees_boost, gbm.perf(boost,
                                             plot.it=F,
                                             oobag.curve=F,
                                             method="OOB"))
      boosting_inner[d,1] <- d
      yHat_boost <- predict(boost, c_data[-f,][innF,])
```

```
      boosting_inner[d,2] <- mean((yHat_boost - c_data[-f,][innF,]$lpsa)^2)
    }
    innerParamBoosting[r2, 1] <- min(boosting_inner[,2])
    innerParamBoosting[r2, 2] <- boosting_inner[which.min(boosting_inner[,2]), 1]

    r2 <- r2 + 1
  }
  # Testing the models in the outer fold
  tree_nested <- tree(lpsa ~ ., c_data[-f,])
  pruned_tree_nested <- prune.tree(tree_nested, best=innerParam[which.min(innerParam[1,]), 2])
  yHat_nd <- predict(pruned_tree_nested, c_data[f,])
  decision_mse <- c(decision_mse, mean((yHat_nd - c_data[f,]$lpsa)^2))

  rf <- randomForest(lpsa ~ .,
                     data = c_data[-f,],
                     mtry = innerParamRF[which.min(innerParamRF[,1]), 2],
                     importance = TRUE)
  yHat_rf <- predict(rf, newdata=c_data[f,])
  rf_mse <- c(rf_mse, mean((yHat_rf - c_data[f,]$lpsa)^2))

  boost <- gbm(lpsa ~ .,
               data = c_data[-f,],
               distribution = "gaussian",
               n.trees = Mode(trees_boost),
               interaction.depth = innerParamBoosting[which.min(innerParamBoosting[,1]), 2],
               n.cores = 4,
               verbose = F)
  yHat_boost <- predict(boost, c_data[f,])
  boosting_mse <- c(boosting_mse, mean((yHat_boost - c_data[f,]$lpsa)^2))

  r <- r + 1
}
```

After the nested computation has ended, we can check the average MSE of each method:

```
df_MSE <- data.frame(model=c("Decision tree", "Random forest", "Boosting"),
                mse=c(mean(decision_mse), mean(rf_mse), mean(boosting_mse)),
                rmse=c(sqrt(mean(decision_mse)), sqrt(mean(rf_mse)), sqrt(mean(boosting_mse))))

ggplot(data=df_MSE, aes(x=model, y=mse)) +
  geom_bar(stat="identity", fill = "springgreen3", width = 0.5) +
  labs(title="Model comparison", y="MSE", x="Model") +
  theme_light()
```




```
print(df_MSE)
```

```
##           model       mse      rmse
## 1 Decision tree 0.8713180 0.9334442
## 2 Random forest 0.6871178 0.8289257
## 3      Boosting 0.7017415 0.8377001
```

Conclusion

We can conclude the analysis saying that there is a good advantage in using a random forest as opposed to a simple decision tree and also a small improvement compared to a boosted model. This is not conclusive evidence of a model being always better than the others, but for this particular dataset the random forest method outperforms the others. As we have seen, it's important to validate the parameter choices in order to achieve the best possible performance with all methods.