Homework 2

Antonio Padalino

2022-05-07

Contents

Homework 2

```
# general purpose
library('tidyverse')

# plotting
library('ggplot2')

# splitting and CV
library('rsample')
library('caret')

# tree-based methods
library('randomForest')
library('tree')
library('gbm')
```

The presented dataset comes from a study conducted on a sample of 97 men who were about to receive a radical prostatectomy, aimed at investigating the correlation between the level of prostate-specific antigen `lpsa` and a number of clinical measures.

1. Exploratory analysis

First, the `prostate.csv` dataset take a look at some descriptive summary statistics.

The dataset includes the above-mentioned `lpsa` variable, our target variable (in ng/ml and log scaled), and 8 other predictive measures. Such factors available are:

- `lcavol`: cancer volume in cm3 - logarithmic
- `lweight`: prostate weight in g - logarithmic
- `age` in years
- `lbph`: amount of benign prostatic hyperplasia in cm2 - logarithmic
- `svi`: seminal vesicle invasion - as a 1/0 dummy variable
- `lcp`: capsular penetration in cm - logarithmic
- `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 score

```
df <- read_csv('prostate.csv')

head(df)
```

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

```
summary(df)
```

```
##      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
```

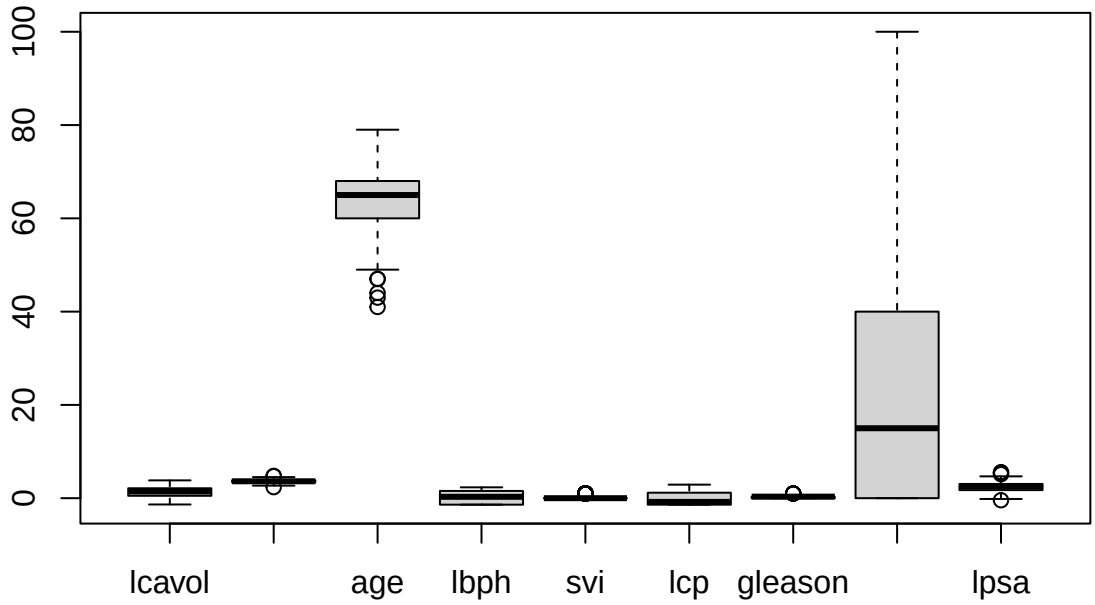
Some observation:

- the `gleason` variable is a score represented by an integer between 6 and 9:
  it may be useful to convert it to a continue scale 0-1

```
df$gleason <- round((df$gleason-6)/3, 2)
```

- Also, it seems like no NA values are included in the dataframe.
- It still might be interesting to inspect the distribution of each variable:

```
boxplot(df)
```



It looks like the `age` variable is the one containing the most outliers. Using the Inter-Quantile Range approach, we identify such outliers.

We define the normal data range $[\ Q_1 - 1.5 * IQR,\ Q_3 + 1.5 * IQR\ ]$ and inspect data point outside it:

```
Q1 <- quantile(df$age,.25)
Q3 <- quantile(df$age,.75)
IQR <- IQR(df$age)

outliers <- df$age[df$age < Q1-1.5*IQR]

cat('Outliers for variable `age` are:', outliers)
```

```
## Outliers for variable ‘age‘ are: 47 41 43 47 44
```

On a total number of rows of 97, 5 outliers observations has been detected. Given the quite small size of the dataset, it would be useful to preserve as many observations as possible. Hence we decide not to drop the rows containing outliers for the variable `age`.

2. Fitting a Decision Tree

A usual, before getting into operations which implies some kind of randomness, a seed is set to make results in this examination perfectly reproducible.

```
set.seed(99)
```

It is now time to split the dataset into train-test partitions. To do so, the `initial_split()` function from the `rsample` package is used to perform a 0.7-0.3 split.

```
df_split <- initial_split(df, prop=0.7)

x_train <- training(df_split)
x_test <- testing(df_split)
y_test <- x_test$lpsa
```

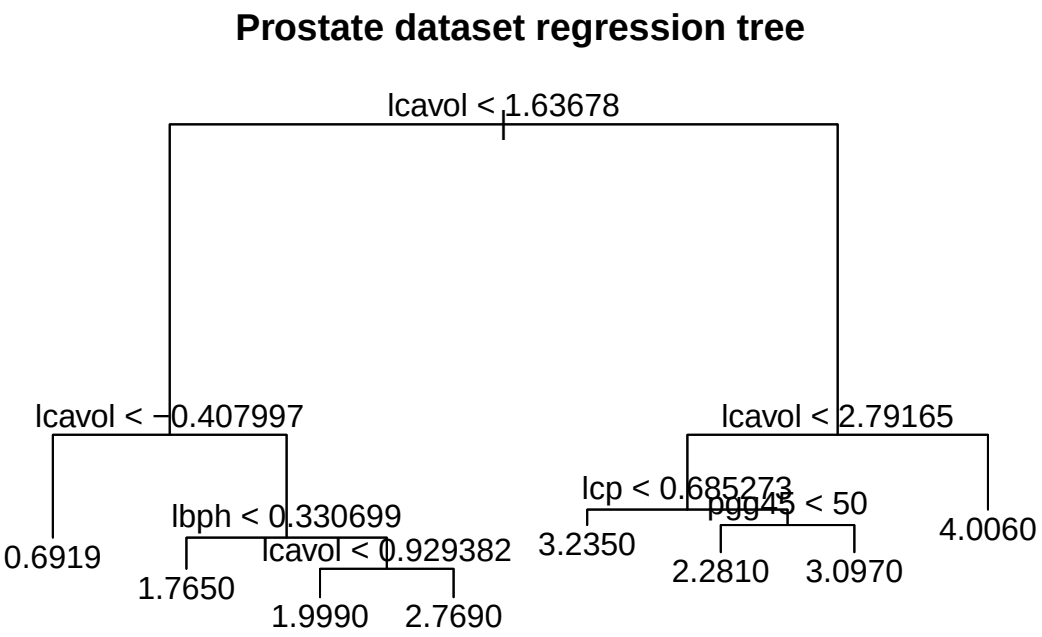
Now the actual decision tree is fitted. Only the `x_train` dataset is used.

```
tree.prostate <- tree(lpsa ~ ., data=x_train)
summary(tree.prostate)
```

```
##
## Regression tree:
## tree(formula = lpsa ~ ., data = x_train)
## Variables actually used in tree construction:
## [1] "lcavol" "lbph"   "lcp"    "pgg45"
## Number of terminal nodes:  8
## Residual mean deviance:  0.3426 = 20.21 / 59
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.30100 -0.38050 -0.02116  0.00000  0.39490  1.57700
```

The variable used in the decision tree are `lcavol`, `lweight`, `age` ad `pgg45`, and the tree looks as below:

```
plot(tree.prostate)
text(tree.prostate, pretty=0)
title(main="Prostate dataset regression tree")
```



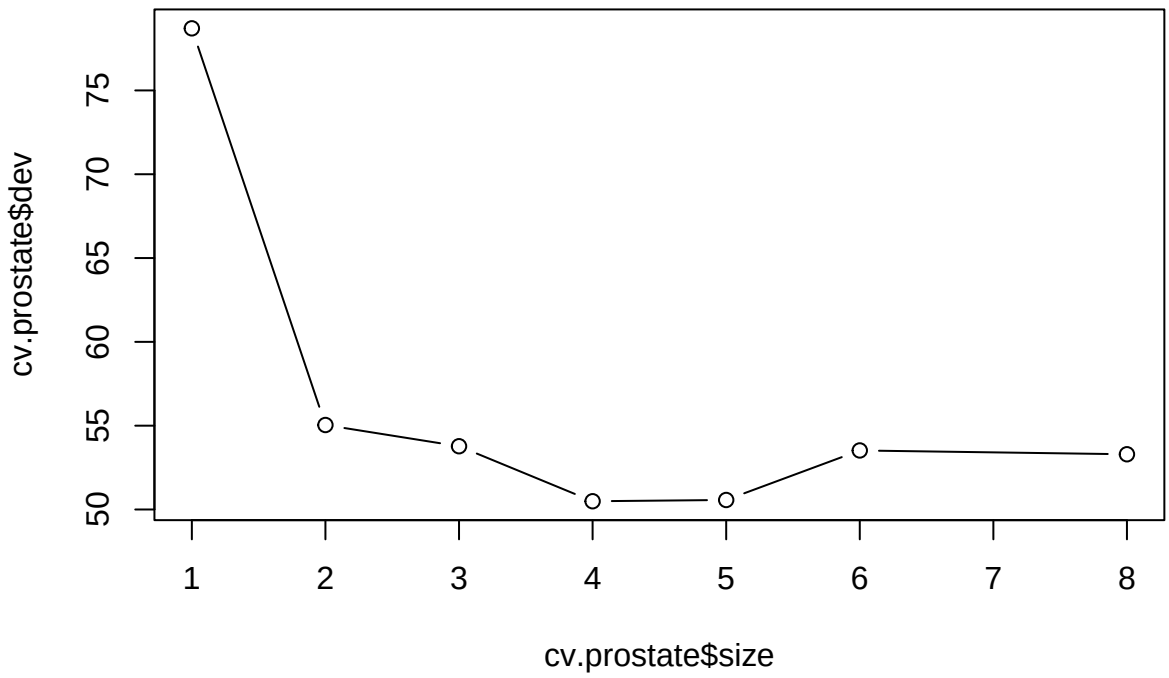
What's the Mean Squared Error on such decision tree?

```
preds = predict(tree.prostate, x_test)

print(paste('MSE for unpruned tree is', mean((preds - y_test)^2)))
```

```
## [1] "MSE for unpruned tree is 0.938728831506062"
```

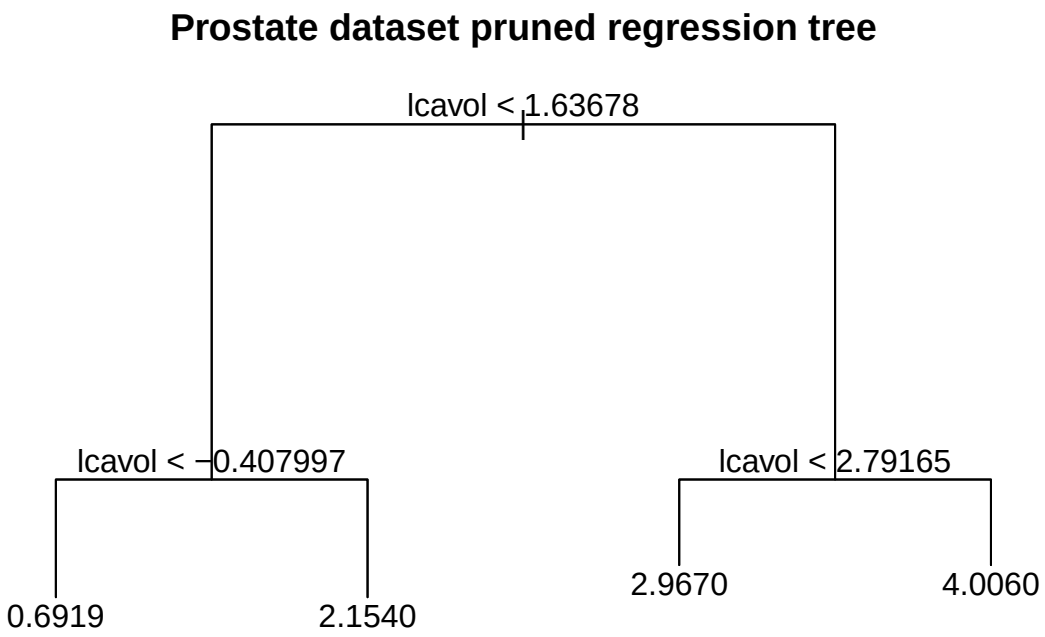
Let's see if pruning the tree does any good to our score. The best number of nodes is found through cross validation, guided by the number of misclassifications made by the tree. Such process is carried out automatically by the `cv.tree()` function, thanks to which we can also visualize the CV error as a function of either `size` and `k`.

```
cv.prostate <- cv.tree(tree.prostate)
plot(cv.prostate$size, cv.prostate$dev, type="b")
```



It seems like the best `size` is 4. A pruned tree is fit with such value.

```
pruned.prostate <- prune.tree(tree.prostate, best=4)
plot(pruned.prostate)
text(pruned.prostate, pretty=0)
title(main="Prostate dataset pruned regression tree")
```



Again, the `lcavol` variable plays a huge role: in this pruned version of our decision tree it is the only variable guiding the prediction of the target `lpsa`.

The MSE is computed again on the pruned tree:

```
preds = predict(pruned.prostate, x_test)

print(paste('MSE for unpruned tree is', mean((preds - y_test)^2)))
```

```
## [1] "MSE for unpruned tree is 1.01774401953848"
```

It is higher than before. While it may seems weird at first, we must consider that a `size` value with lower `dev` in the CV process (as 4 was chosen) is not guaranteed to yield the best MSE in a decision tree afterwards.

3. Fitting a Random Forest

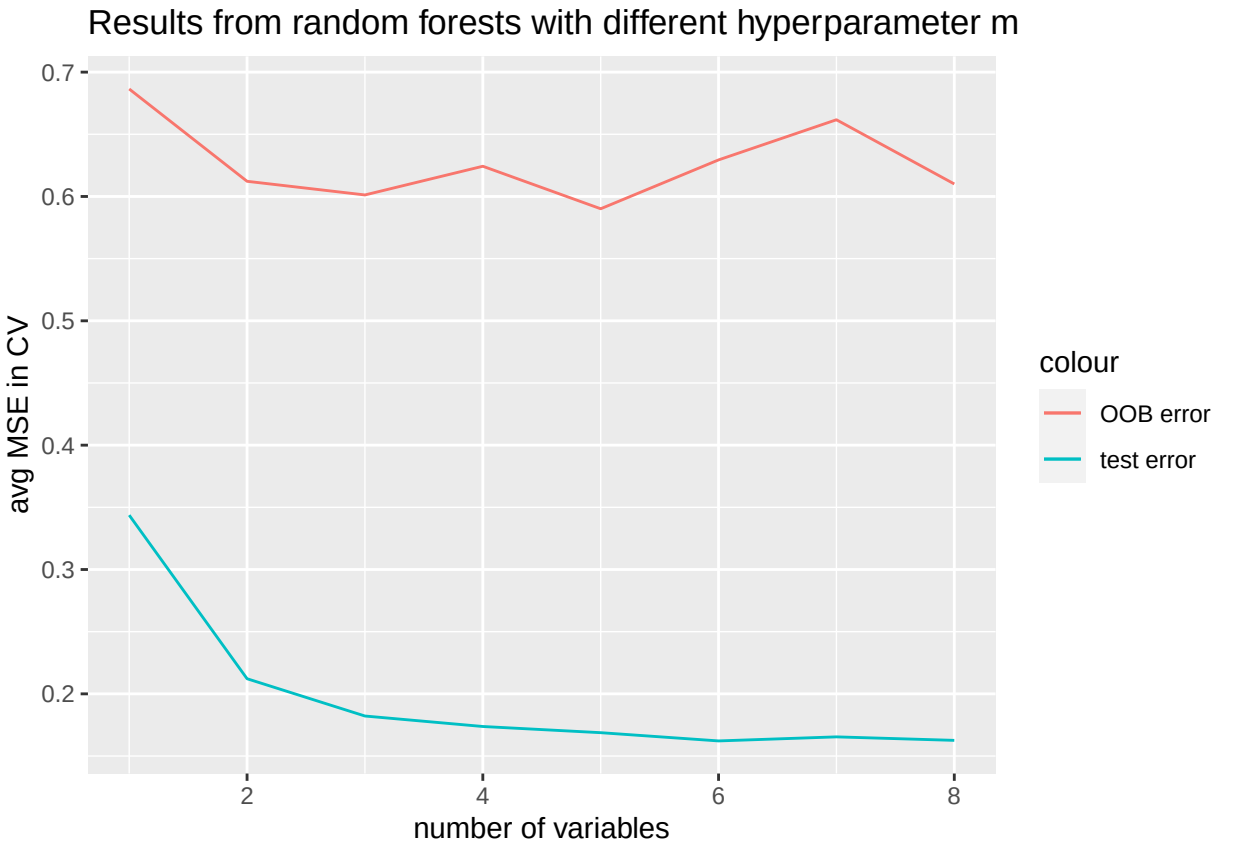
A Random Forest is fitted on our dataset as well.

To select the right tuning parameter m (the number of variables to consider at each split), CV is carried out with $k = 10$ and a list of possible m going from 1 to the number of variables in the dataset (8).

To evaluate each forest, both the MSE on the test dataset and the OOB error on the whole dataset are considered.

```r
nvar = ncol(df)-1

cv_mses <- vector()
oob_mses <- vector()


for (m in 1:nvar) {

  folds_ids <- caret::createFolds(df$lpsa, k = 10, list = TRUE, returnTrain = TRUE)

  all_mses <- vector()

      for (f in names(folds_ids)) {

        df_train <- df[folds_ids[[f]],]
        df_val <- df[-folds_ids[[f]],]

        x_val <- df[,-9]
        y_val <- df[, 9]

        cur_forest.prostate <- randomForest(lpsa ~ .,
                                            data=df_train,
                                            mtry=m,
                                            ntree=250,
                                            importance=TRUE)

        preds = predict(cur_forest.prostate, x_val)

        all_mses <- c(all_mses, mean(t(preds - y_val)^2))

    }

    cv_mses <- c(cv_mses, mean(all_mses))


    mth_forest <- randomForest(lpsa ~ .,
                               data=x_train,
                               mtry=m,
                               ntree=250,
                               importance=TRUE)

    oob_mses <- c(oob_mses, mean(mth_forest$mse))

}

rf_results <- data.frame(m = seq(1:nvar),
                         mse = cv_mses,
                         oob_mse = oob_mses)

ggplot(rf_results) +
  geom_line(aes(x = m, y = mse, col='test error')) +
  geom_line(aes(x = m, y = oob_mse, col='OOB error')) +
  labs(title = 'Results from random forests with different hyperparameter m',
```

```
       x = 'number of variables',
       y = 'avg MSE in CV')
```




```
print(paste('MSE is minimum on', which(cv_mses == min(cv_mses))))
```

```
## [1] "MSE is minimum on 6"
```

```
print(paste('OOB is minimum on',which(oob_mses == min(oob_mses))))
```

```
## [1] "OOB is minimum on 5"
```

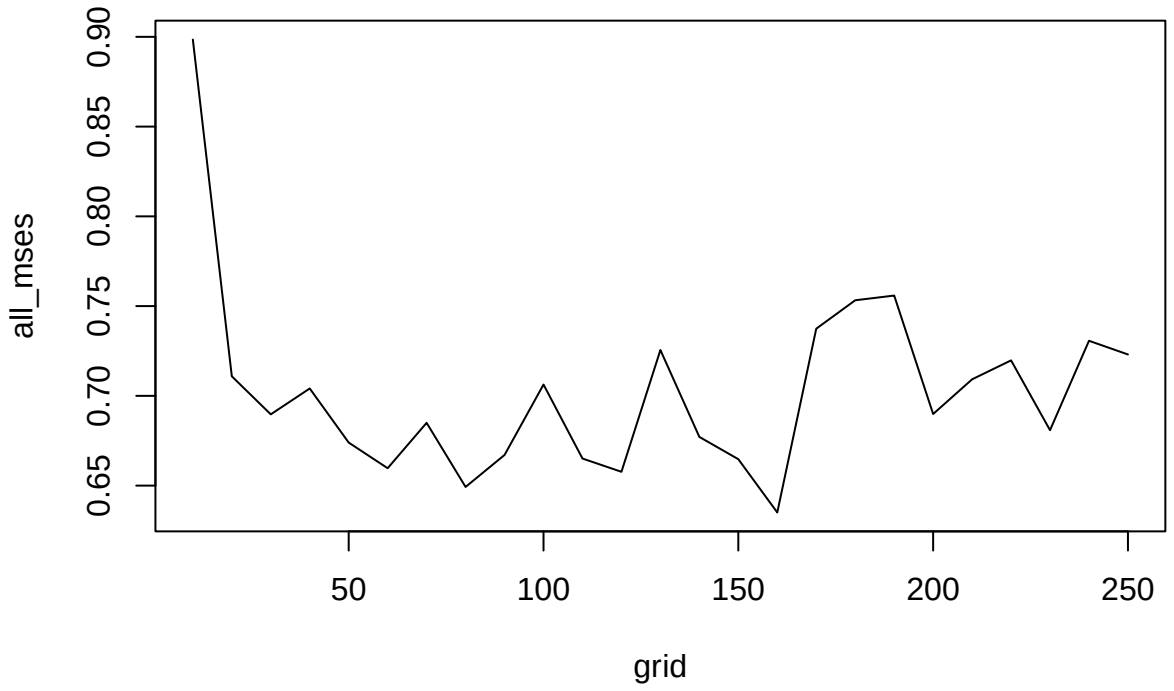
The OOB error reaches its minimum on $m = 5$, while the test MSE is minimized on $m = 6$.

The Mean Square Error on test follows a more definite distribution over the different number of variables. It may be more appropriate to use it (instead of the OOB error) for future evaluations on our model.

4. Fitting boosted Regression Trees

A boosted regression trees is now fitted on our dataset, after selecting the optimal number of boosting iterations (`n.trees`) through CV, cycling through a vector of different 'candidates'.

```
all_mses <- vector()

grid <- seq(10,250,10)

for (n in grid) {

  boosted.trees <- gbm(lpsa ~ ., data=df, distribution="gaussian", n.trees=n, cv.folds=5)

  all_mses <- c(all_mses, boosted.trees$cv.error[length(boosted.trees$cv.error)])

}

plot(grid, all_mses, 'line')
```




```
print(paste('minimum MSE on n.trees =',grid[which(all_mses==min(all_mses))]))
```

```
## [1] "minimum MSE on n.trees = 160"
```

Knowing 160 to be the optimal hyperparameter for `n.trees`, we fit a model with `gbm()`:

```
boosted <- gbm(lpsa ~ ., data=df_train, distribution="gaussian", n.trees=160)

preds <- predict(boosted, newdata=x_test)

print(paste('MSE for boosted decision tree is', mean((preds - y_test)^2)))
```

```
## [1] "MSE for boosted decision tree is 0.464170243538387"
```

5. Comparing the models

For the last section of this examination, the 3 approaches that has been used will now be evaluated in CV.

Each model (Decision Tree, Random Forest, Boosted Random Forest) will be fitted on each training fold, with $k = 10$ folds, and evaluated on each respective fold. At each of the 10 iterations of the CV process, the model is re-optimized to yield the best score. In particular, such optimization refers to:

- pruning the tree, if necessary, for the Decision Tree
- finding the best number m of predictors to use at each split for Random Forest
- finding the best number n of boosting iterations for Boosted Random Forest

To make the process more readable and easy to approach, it is useful to define 3 functions that, for given train and test datasets, return the best model (automatically performing optimization) for each of the 3 models. Such functions are defined as follows:

```
best_dTree <- function(df, target, type, warnings=F) {


  df_split <- initial_split(df, prop=0.7)

  df_train <- training(df_split)
  x_test <- testing(df_split)
  y_test <- x_test$lpsa


  formula = as.formula(paste(target,'~.'))

  first.tree <- tree(formula, data=df_train)

  preds <- predict(first.tree, x_test)

  first.mse <- mean((preds - y_test)^2)

  cv.tree <- cv.tree(first.tree)

  best.size <- cv.tree$size[which(cv.tree$dev == min(cv.tree$dev))]

  pruned.tree <- prune.tree(first.tree, best=best.size)

  preds = predict(pruned.tree, x_test)

  new.mse <-  mean((preds - y_test)^2)

  out.tree <- pruned.tree
  out.mse <- new.mse

  if (new.mse > first.mse) {

    if (warnings==T) {cat('[!] Pruned tree yields higher MSE than unpruned tree.\n')}
```

```
    out.tree <- first.tree
    out.mse <- first.mse

  }

  if (type == 'model') {return (out.tree)}
  if (type == 'mse') {return (out.mse)}

}
```

```
best_rForest <- function(df, target, type) {

  nvar = ncol(df)-1

  cv_mses <- vector()

  for (m in 1:nvar) {

    folds_ids <- caret::createFolds(unlist(select(df, target)),
                                    k = 10,
                                    list = TRUE,
                                    returnTrain = TRUE)

    all_mses <- vector()

        for (f in names(folds_ids)) {

          df_train <- df[folds_ids[[f]],]
          df_val <- df[-folds_ids[[f]],]

          x_val <- (select(df,-lpsa))
          y_val <- (select(df, lpsa))

          formula <- as.formula(paste(target,'~.'))

          cur_forest <- randomForest(formula,
                                     data=df_train,
                                     mtry=m,
                                     ntree=250,
                                     importance=TRUE)

          preds = predict(cur_forest, x_val)

          all_mses <- c(all_mses, mean(t(preds - y_val)^2))

      }

      cv_mses <- c(cv_mses, mean(all_mses))

  }

  best.m <- which(cv_mses == min(cv_mses))
```

```
  best.forest <- randomForest(formula,
                              data=df_train,
                              mtry=best.m,
                              ntree=250,
                              importance=TRUE)

  if (type == 'model') {return (best.forest)}
  if (type == 'mse') {return (best.mse)}

}
```

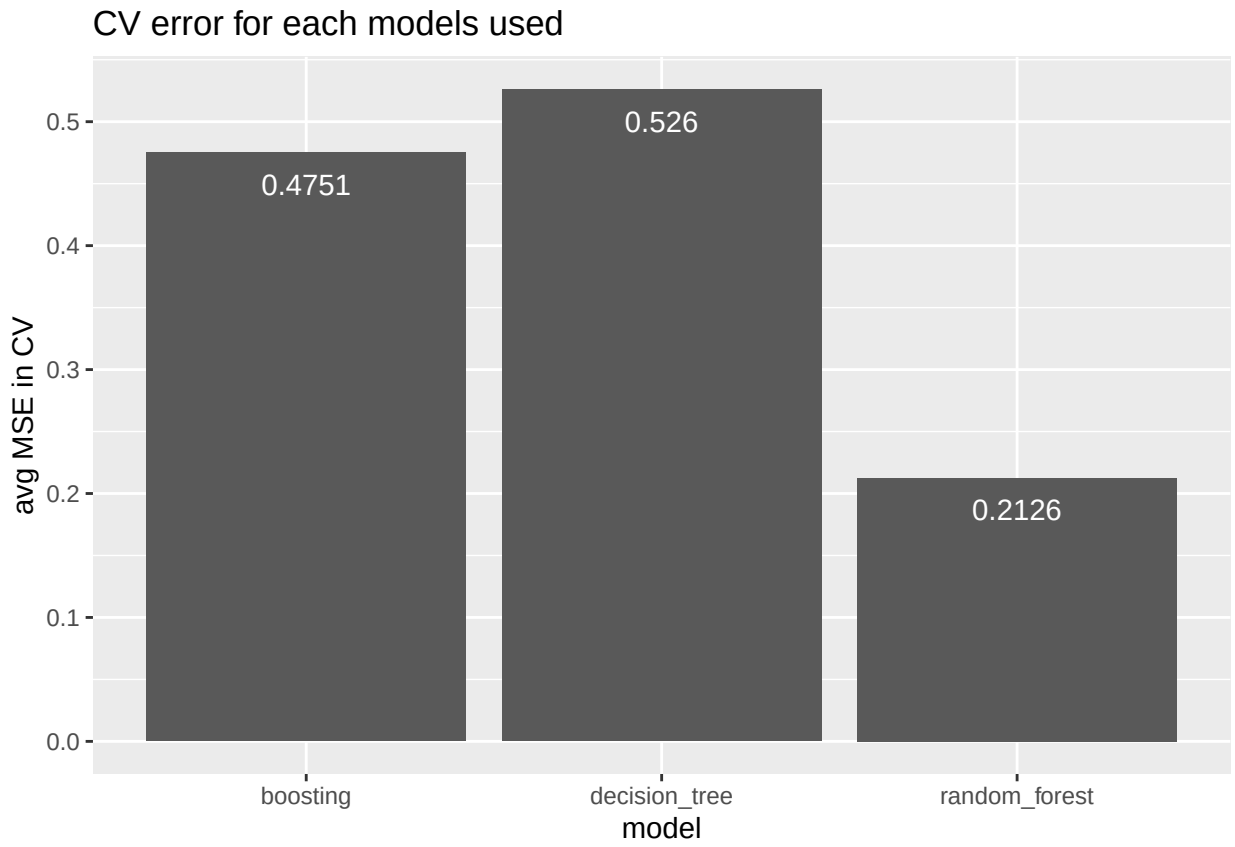
```
best_boosting <- function(df, target, type) {

  all_mses <- vector()

  grid <- seq(10,250,10)

  for (n in grid) {

    formula <- as.formula(paste(target,'~.'))

    boosted.trees <- gbm(formula, data=df, distribution="gaussian", n.trees=n, cv.folds=5)

    all_mses <- c(all_mses, boosted.trees$cv.error[length(boosted.trees$cv.error)])

  }

  best.n <- grid[which(all_mses==min(all_mses))]

  best.boosted <- gbm(formula, data=df, distribution="gaussian", n.trees=best.n)

  return (best.boosted)

}
```

Now for the actual process:

```
folds_ids <- caret::createFolds(df$lpsa, k = 10, list = TRUE, returnTrain = TRUE)


dt.cv_err <- vector()
rf.cv_err <- vector()
brf.cv_err <- vector()

for (f in names(folds_ids)) {

  df_train <- df[folds_ids[[f]],]
  df_test <- df[-folds_ids[[f]],]

  x_test <- select(df,-lpsa)
  y_test <- unlist(select(df, lpsa))

  best.dt <- best_dTree(df_train, 'lpsa', 'model')
```

```
  best.rf <- best_rForest(df_train, 'lpsa', 'model')
  best.brf <- best_boosting(df_train, 'lpsa')

  dt.cv_err <- c(dt.cv_err, mean((predict(best.dt, x_test) - y_test)^2))
  rf.cv_err <- c(rf.cv_err, mean((predict(best.rf, x_test) - y_test)^2))
  brf.cv_err <- c(brf.cv_err, mean((predict(best.brf, x_test) - y_test)^2))

}
```

```
to_plot <- data.frame(model=c('decision_tree', 'random_forest', 'boosting'),
                      error=c(mean(dt.cv_err),mean(rf.cv_err),mean(brf.cv_err)))

ggplot(to_plot, aes(x=model, y=error)) +
  geom_bar(stat='identity') +
  geom_text(aes(label=round(error,4)), vjust=2, color='white') +
  labs(title='CV error for each models used',
       y='avg MSE in CV',
       x='model')
```



The model which performed better in the current examination was Random Forest, with an average MSE in cross validation of ≈ 0.2.

This was - at least partially - to be expected: a Random Forest generally provides a higher level of accuracy over a Decision Tree algorithm, thanks to a number of reasons. The main one is surely the use of bagging, meaning that subsets of features are randomly chosen at each iteration, to make the final model less dependent on specific features (thus reducing overfitting). Decision Trees are clearly a second-choice.

Random Forest are more similar to Boosting Trees - in a sense - than Decision Trees. The main difference is the way in which trees are trained in the different models: in Boosting, we fit trees sequentially, each to correct the errors of the previous one, while in Random Forest each tree is fitted independently from the others. Also, in the 'prediction' phase Random Forest uses a vote mechanisms which aggregates predictions from independent trees, while Boosting only admits a sequential evaluation (being the trees fitted in a certain fixed order).

Such elements may have impacted on the difference in score between Random Forest and Boosting Trees in this analysis. One cannot say with absolute certainty that one method is assured to outperform the other: they are two equally valid and flexible ensemble methods.