Statistical Learning, Homework #2

Annalisa Xamin

Introduction

In the following analysis, we will focus on cancer data to investigate the correlation 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 9 explanatory variables are:

- `lcavol`: log(cancer volume in cm^3)
- `lweight`: log(prostate weight in g)
- `age` in years
- `lbph`: log(amount of benign prostatic hyperplasia in cm^2)
- `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

During the analysis we will use three different methods (cost-complexity decision trees, random forests and boosting) to later compare their performances.

To start, the data is loaded and a summary is printed.

```
df_prostate <- read.csv("./prostate.csv", sep = ",")
summary(df_prostate)
```

```
##      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
```

As we can see from the summary, in the data there are no NAs.

Decision Tree

We split the data into training and test sets using the `caret` package and fit a decision tree on the whole data set.

```
set.seed(1)

sample <- caret::createDataPartition(df_prostate$lpsa, p = 0.8)
train_data <- df_prostate[sample$Resample1, ]
test_data <- df_prostate[-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

# Fit the model on the whole data
tree_prostate <- tree(lpsa ~ ., df_prostate)
summary(tree_prostate)
```

```
##
## Regression tree:
## tree(formula = lpsa ~ ., data = df_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 tree was created and only tree variables were used, namely `lcavol`, `lweight`, `pgg45`. We then procede by plotting the tree.

```
plot(tree_prostate)  # plot the branches
text(tree_prostate, pretty = 0)  # add the labels to the branches
title(main = "Unpruned tree")
```

The most important predictor is at the stump of the tree. As we could already saw from the summary of the tree, it has 9 terminal nodes which identify the partitions of the feature space. Each terminal node displays the average of the `lpsa` values for the corresponding region, which is the prediction for the observations that will fall within it.

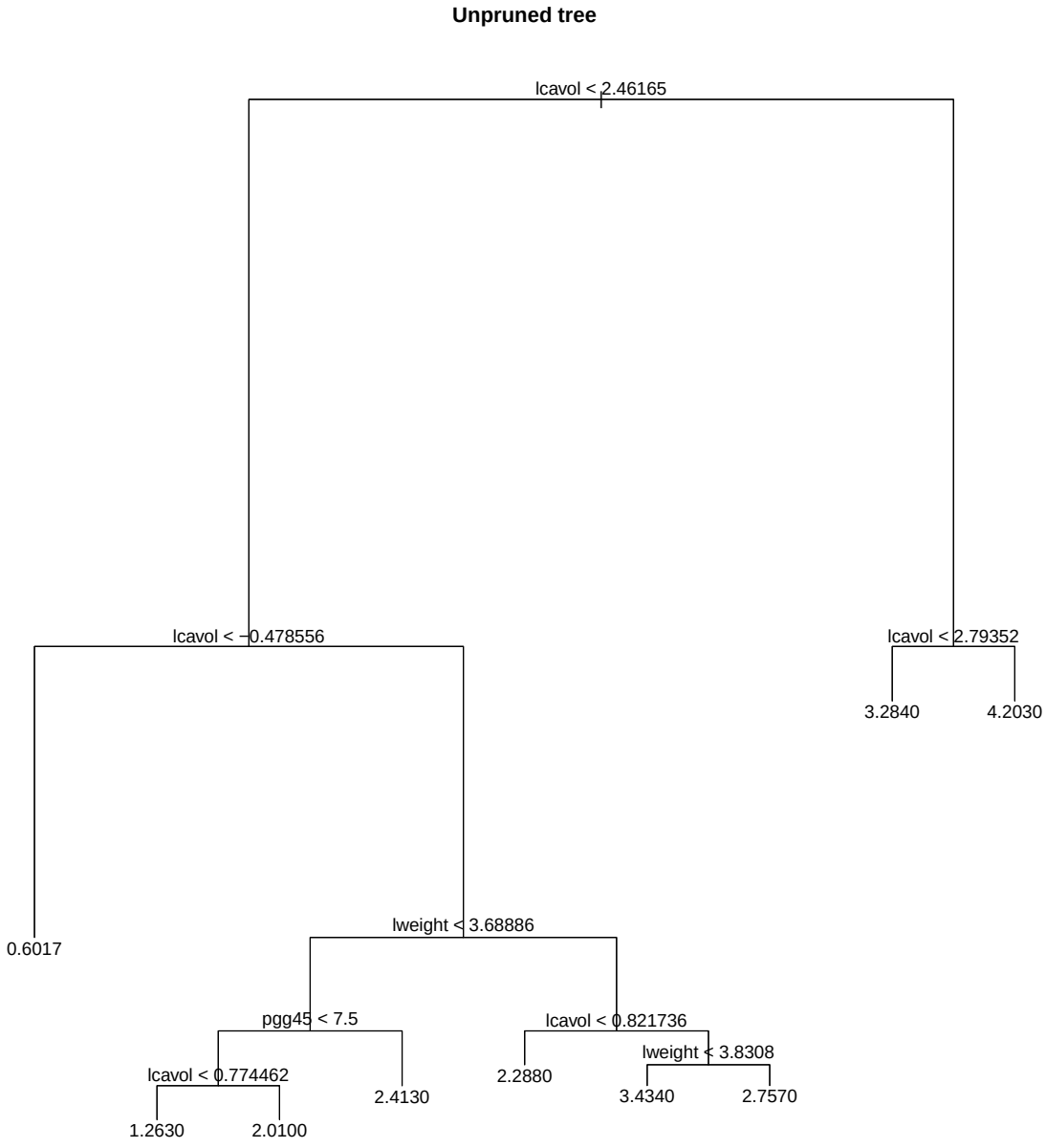


Figure 1: Unpruned decision tree

Cross-validation

Since the unpruned tree may overfit the data, we will proceed with the cross-validation in order to choose the tree complexity and decide whether we should prune the tree based on the result.

```
set.seed(1)
cv_tree <- cv.tree(object = tree_prostate)

graph <- tibble(dev = cv_tree$dev, tsize = cv_tree$size, alpha = cv_tree$k)
ggplot(data = graph, aes(x = tsize, y = dev)) + geom_line(color = "darkgrey") +
    geom_point(color = "black", size = 3) + geom_point(data = NULL,
    aes(x = cv_tree$size[which.min(cv_tree$dev)], y = min(cv_tree$dev),
        colour = "red", size = 3)) + labs(y = "Deviance", x = "Tree size") +
    guides(size = "none", colour = "none") + scale_x_discrete(labels = 1:9,
    limits = 1:9)
```

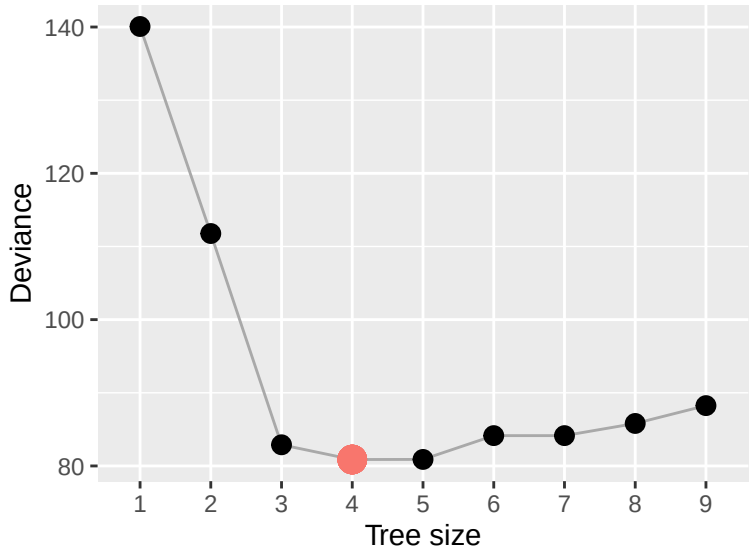


Figure 2: Cross validated tree size selection

From the cross-validation, we get that the lowest deviance, therefore the best result, is obtained with size of 4.

Pruning the tree

We use the result obtained from the cross-validation to prune the tree.

```
opt.size <- cv_tree$size[which.min(cv_tree$dev)]  # optimal size = 4
pruned_prostate <- prune.tree(tree_prostate, best = opt.size)  # prune using the optimal size

# plot the pruned tree
plot(pruned_prostate)
text(pruned_prostate, pretty = 0)
title(main = "Pruned tree")
```

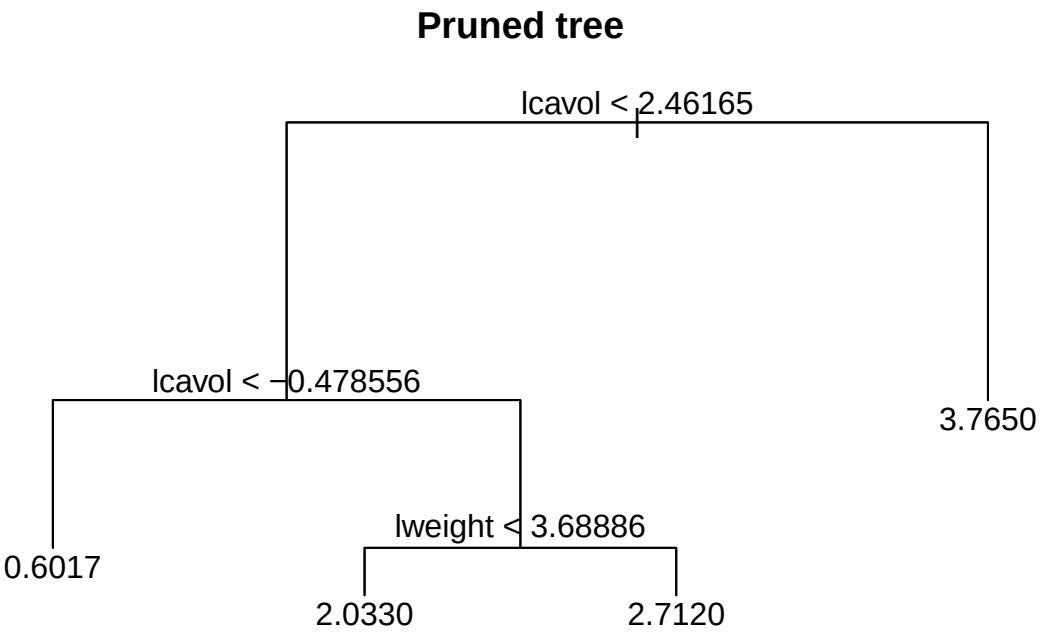


Figure 3: Pruned decision tree

As we can see from the plot of the pruned tree, `pgg45` is no longer considered as predictor. The variables used in tree construction are only 2: `lcavol` and `lweight`.

Fit the model on the training data

Finally, we refit the model only on the training data and make predictions.

```
set.seed(1)
tree_prostate <- tree(lpsa ~ ., train_data)
pruned_prostate <- prune.tree(tree_prostate, best = 4)
y_hat <- predict(pruned_prostate, test_data)
```

To simplify the computation of MSE, we define a function for that:

```
compute_mse <- function(preds, truth) {
    mean((preds - truth)^2)
}
```

```
MSE_dt <- compute_mse(y_hat, test_data$lpsa)
paste("MSE: ", MSE_dt)
```

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

The test MSE associated to this decision tree is ≈ 1.266964. This means that this models leads to test predictions that are (on average) ≈ 1.125595 (RMSE) of the true median value.

Random Forest

We consider now a random forest. To begin, we define the number of variables that Random Forest models will examine at each split in `m` and the 5 folds for the k-fold cross validation. In particular, we use the `createFolds` function to specify the number of folds.

```
set.seed(1)

nvar <- ncol(df_prostate) - 1  # number of predictors
m <- seq(1, nvar)

# initialize empty matrix that will contains the errors
err_matrix <- matrix(nrow = nvar, ncol = 2)

folds <- createFolds(df_prostate$lpsa, k = 5)  # 5 folds

MSE <- c()  # initialize empty list to store MSE
OOB <- c()  # initialize empty list to store OOB error

for (i in m) {
    # iterate over m iterate over the folds
    for (f in folds) {
        rf <- randomForest(lpsa ~ ., data = df_prostate[-f, ],
            mtry = i, importance = TRUE)
```

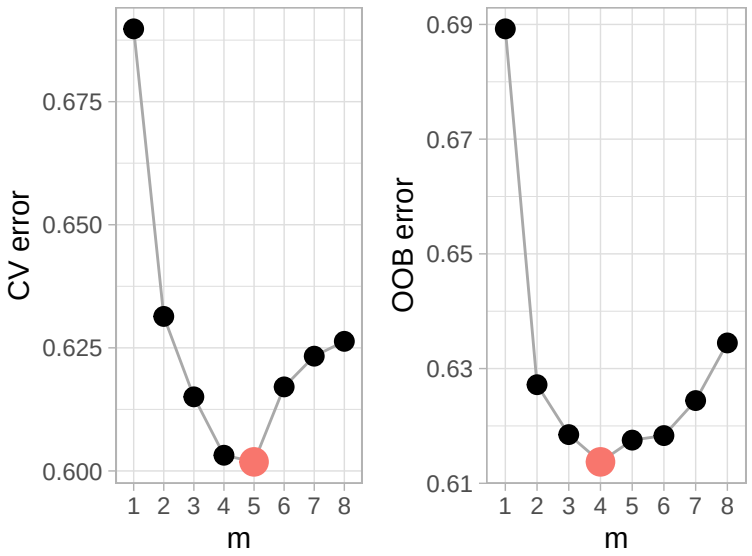
```
        y_hat_rf <- predict(rf, newdata = df_prostate[f, ])
        MSE <- c(MSE, compute_mse(y_hat_rf, df_prostate[f, ]$lpsa))
        OOB <- c(OOB, compute_mse(rf$predicted, df_prostate[-f,
            ]$lpsa))
    }
    err_matrix[i, 1] <- mean(MSE)
    err_matrix[i, 2] <- mean(OOB)
    MSE <-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

Now, we procede to plot the CV and OOB results.

```
CV_plot <- ggplot(NULL, aes(x = c(1:8), y = err_matrix[, 1])) +
    theme_light() + geom_line(color = "darkgrey") + geom_point(color = "black",
    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 = "CV error", x = "m") + guides(size = F, colour = F) +
    scale_x_discrete(limits = 1:8, labels = c(1:8))

OOB_plot <- ggplot(NULL, aes(x = c(1:8), y = err_matrix[, 2])) +
    theme_light() + geom_line(color = "darkgrey") + geom_point(color = "black",
    size = 3) + geom_point(data = NULL, aes(x = c(1:8)[which.min(err_matrix[,
    2])], y = min(err_matrix[, 2]), color = "red", size = 3)) +
    labs(y = "OOB error", x = "m") + guides(size = F, colour = F) +
    scale_x_discrete(limits = 1:8, labels = c(1:8))
```

```
plot_grid(CV_plot, OOB_plot, labels = "", ncol = 2, nrow = 1)
```



As we can notice, CV and OOB errors do not reach the minimum at the same value of m: for CV the minimum is reached by 5, while for the OOB error the minimum is 4.

The optimal model

We proceed our analysis by creating another model using the training and testing datasets splits and setting `mtry = 5`, as suggested from the CV error. We choose the result suggested from the CV over the one obtained from the OOB, because CV is more precise. OOB can be useful with a higher number of data, since it is more computational feasible. But in our case we have small data, so CV works just fine.

```
set.seed(1)
rf_adj <- randomForest(x_train, y_train, xtest = x_test, ytest = y_test,
    mtry = 5, importance = TRUE)

vip(rf_adj, aesthetics = c(fill = "blue")) + theme_light()
```

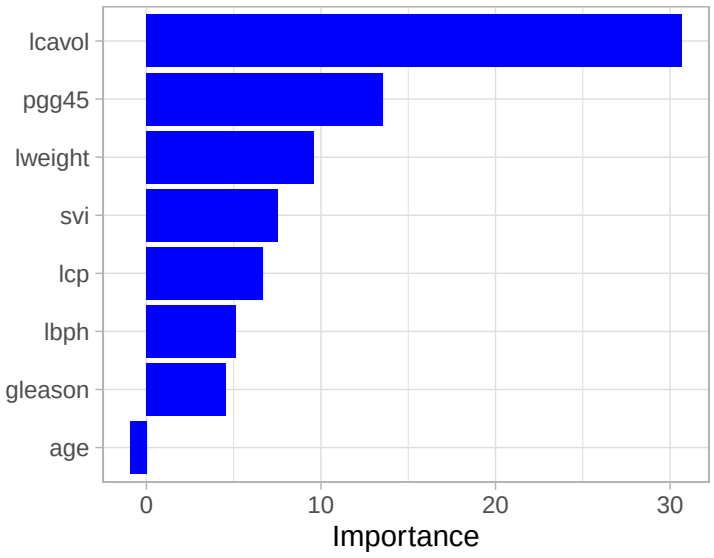


Figure 4: Variable importance plot

From the plot, we can see that the most important predictor is `lcavol`, followed by `pgg45` and `lweight`. Both `lcavol` and `lweight` were present also in the pruned tree, while `pgg45` were not.

We visualize the error.

```
ggplot(NULL, aes(x = c(1:500), y = rf_adj$test$mse)) + theme_light() +
    geom_line() + labs(y = "Test set MSE", x = "Number of Trees")
```

```
y_hat_rf <- rf_adj$test$predicted
MSE_rf <- compute_mse(y_hat_rf, test_data$lpsa)
paste("MSE: ", MSE_rf)
```

```
## [1] "MSE:  0.749221108181112"
```

The test MSE is ≈ 0.749221, which is better than the decision tree's result.

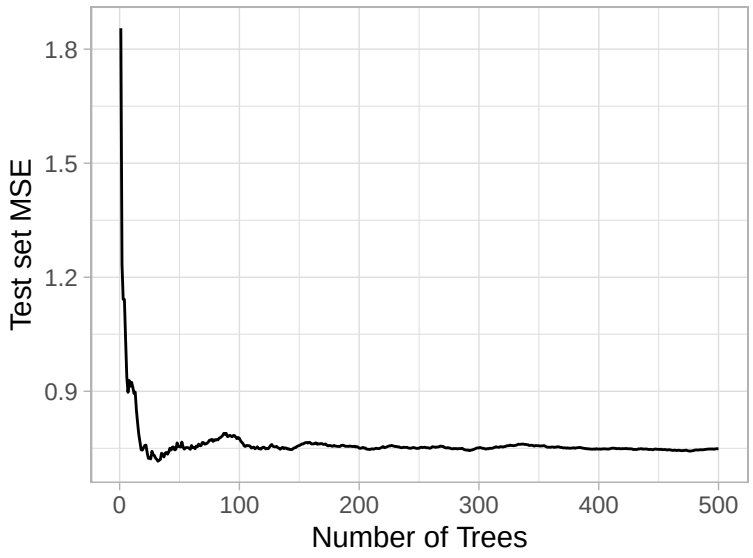


Figure 5: Random Forest: Test set MSE vs number of trees

Boosted regression trees

To fit boosted regression trees on the whole dataset we will use the function `gbm`. To find the optimal model, we need to find the best number of boosting iterations (`n.trees`). To do that, we will do a k-fold cross validation, already implemented by `gbm`.

```
set.seed(1)
matrix_boost <- matrix(0, nrow = 20, ncol = 3)  # initialize matrix

for (d in 1:20) {
    boost <- gbm(lpsa ~ ., data = df_prostate, distribution = "gaussian",
        n.trees = 1000, interaction.depth = d, cv.folds = 5,
        verbose = F, n.cores = 4)

    matrix_boost[d, 1] <- min(boost$cv.error)
    matrix_boost[d, 2] <- which.min(boost$cv.error)
    matrix_boost[d, 3] <- gbm.perf(boost, plot.it = F, oobag.curve = F,
        method = "OOB")
}
```

```
paste("The cross validated MSE suggests", matrix_boost[which.min(matrix_boost[,
    1]), 2], "iterations (trees) and a depth of", which.min(matrix_boost[,
    1]))
```

```
## [1] "The cross validated MSE suggests 31 iterations (trees) and a depth of 15"
```

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

```
## [1] "The OOB error suggests 22 iterations (trees) and a depth of 1"
```

```
colnames(matrix_boost) <- c("MSE|", "Iterations (MSE)|", "Iterations (OOB)")
knitr::kable(matrix_boost)
```

MSE|	Iterations (MSE)|	Iterations (OOB)
0.7048545	23	26
0.6310488	26	23
0.6561338	32	23
0.6302493	36	23
0.6414287	27	23
0.6180743	35	23
0.8115481	22	22
0.6293219	33	22
0.7318021	26	20
0.6482029	31	22
0.5878319	66	22
0.6718510	26	21
0.6600228	41	22
0.7024704	25	22
0.5790538	31	22
0.6178872	53	22
0.6788299	46	22
0.6754484	46	22
0.6644193	32	22
0.6266475	22	23

As we can notice from the table above, we get different suggestion depending on the type of method we choose, CV or OOB.

Optimal model

As we mentioned before, we prefer the results obtained from the CV over the ones obtained from the OOB. So, for the optimal, we set `n.trees = 35` and `interaction.depth = 6`.

```
set.seed(1)
boost_opt <- gbm(lpsa ~ ., data = df_prostate, distribution = "gaussian",
    n.trees = 35, interaction.depth = 6, cv.folds = 5, verbose = F,
    n.cores = 4)

y_hat_boost_opt <- predict(boost_opt, test_data)
mse_boost_opt <- compute_mse(y_hat_boost_opt, test_data$lpsa)
paste("MSE optimal model: ", mse_boost_opt)
```

```
## [1] "MSE optimal model:  0.454368755372244"
```

In this case, the MSE resulted from the boosted regression tree is much lower than the one obtained in the decision tree and in the random forest.

Performance comparison

Until now, we compared the models standalone. Thus, we will now do a cross validation to compare all of them.

```
set.seed(1)

out_folds <- createFolds(df_prostate$lpsa, k = 5)  # outer folds

# initalize lists to store MSE
dt_mse <- c()
rf_mse <- c()
boosting_mse <- c()
trees_boost <- c()

r <- 1

for (f in out_folds) {
    # iterate over outer folds
    r2 <- 1
    in_folds <- createFolds(df_prostate[-f, ]$lpsa, k = 5)  # inner folds
    in_param <- matrix(0, nrow = length(in_folds), ncol = 2)  # inner parameters decision tree
    in_param_rf <- matrix(0, nrow = length(in_folds), ncol = 2)  # inner parameters random forest
    in_param_boost <- matrix(0, nrow = length(in_folds), ncol = 2)  # inner parameters boosting

    for (f2 in in_folds) {
        # iterate over inner folds Decision tree
        tree_nested <- tree(lpsa ~ ., df_prostate[-f, ][-f2,
            ])
        in_MSE_pruning <- c()
        preds <- c()

        for (i in 2:(nvar + 1)) {
            pruned_tree_nested <- prune.tree(tree_nested, best = i)
            y_hat_nd <- predict(pruned_tree_nested, df_prostate[-f,
                ][f2, ])
            in_MSE_pruning <- c(in_MSE_pruning, compute_mse(y_hat_nd,
                df_prostate[-f, ][f2, ]$lpsa))
        }
        in_param[r2, 1] <- min(in_MSE_pruning)
        in_param[r2, 2] <- which.min(in_MSE_pruning)

        # Random forest
        mse_rf_in <- c()
        for (i in 1:nvar) {
            rf <- randomForest(lpsa ~ ., data = df_prostate[-f,
                ][-f2, ], mtry = i, importance = TRUE)

            y_hat_rf <- predict(rf, newdata = df_prostate[-f,
                ][f2, ])
            mse_rf_in <- c(mse_rf_in, compute_mse(y_hat_rf, df_prostate[-f,
                ][f2, ]$lpsa))
        }
```

```
        in_param_rf[r2, 1] <- min(mse_rf_in)
        in_param_rf[r2, 2] <- which.min(mse_rf_in)

        # Boosting
        boosting_in <- matrix(0, nrow = 8, ncol = 2)
        for (d in 1:8) {
            boost <- gbm(lpsa ~ ., data = df_prostate[-f, ][-f2,
                ], 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_in[d, 1] <- d
            y_hat_boost <- predict(boost, df_prostate[-f, ][f2,
                ])
            boosting_in[d, 2] <- compute_mse(y_hat_boost, df_prostate[-f,
                ][f2, ]$lpsa)
        }

        in_param_boost[r2, 1] <- min(boosting_in[, 2])
        in_param_boost[r2, 2] <- boosting_in[which.min(boosting_in[,
            2]), 1]
        r2 <- r2 + 1

    }

    # Testing the models in the outer fold
    tree_nested <- tree(lpsa ~ ., df_prostate[-f, ])
    pruned_tree_nested <- prune.tree(tree_nested, best = in_param[which.min(in_param[1,
        ]), 2])
    y_hat_nd <- predict(pruned_tree_nested, df_prostate[f, ])
    dt_mse <- c(dt_mse, compute_mse(y_hat_nd, df_prostate[f,
        ]$lpsa))

    rf <- randomForest(lpsa ~ ., data = df_prostate[-f, ], mtry = in_param_rf[which.min(in_param_rf[,
        1]), 2], importance = TRUE)

    y_hat_rf <- predict(rf, newdata = df_prostate[f, ])
    rf_mse <- c(rf_mse, compute_mse(y_hat_rf, df_prostate[f,
        ]$lpsa))
    boost <- gbm(lpsa ~ ., data = df_prostate[-f, ], distribution = "gaussian",
        n.trees = Mode(trees_boost), interaction.depth = in_param_boost[which.min(in_param_boost[,
            1]), 2], n.cores = 4, verbose = F)

    y_hat_boost <- predict(boost, df_prostate[f, ])
    boosting_mse <- c(boosting_mse, compute_mse(y_hat_boost,
        df_prostate[f, ]$lpsa))
    r <- r + 1
}
```

In the following table and plot, we can visualize the results of the comparison between the 3 different models.

```
df_comparison = data.frame(Model = c("Decision Tree", "Random Forest",
    "Boosted Regression Tree"), MSE = c(mean(dt_mse), mean(rf_mse),
    mean(boosting_mse)))
knitr::kable(df_comparison)
```

Model	MSE
Decision Tree	0.9379300
Random Forest	0.6170095
Boosted Regression Tree	0.6911314

```
ggplot(data = df_comparison, aes(x = Model, y = MSE)) + geom_bar(stat = "identity",
    fill = "blue", width = 0.5) + labs(y = "MSE", x = "Model") +
    theme_light()
```

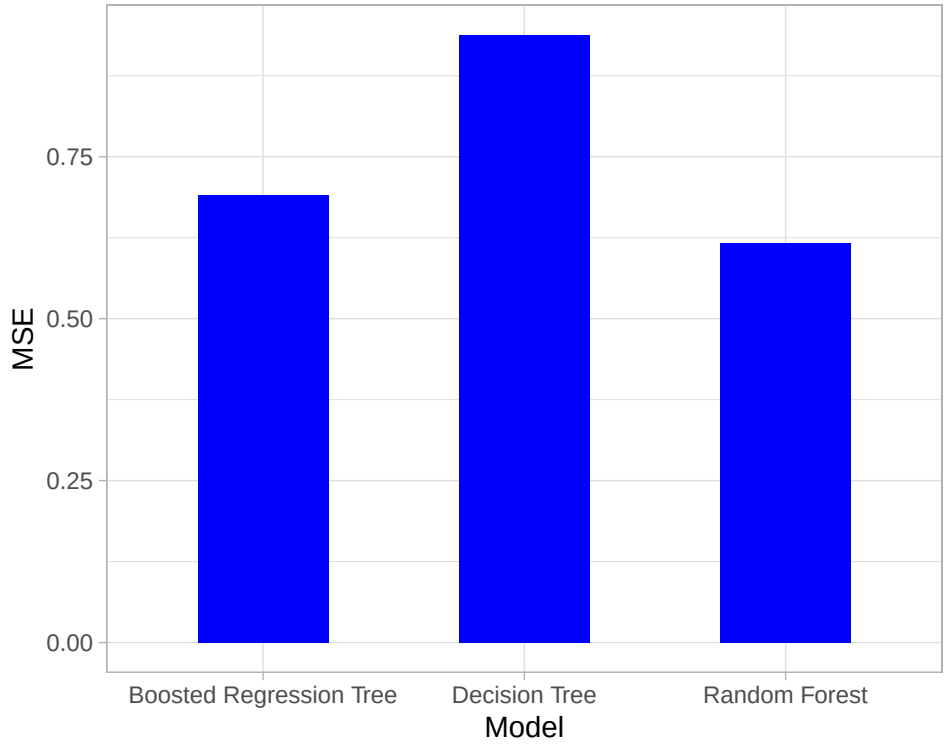


Figure 6: Performance comparison between models

Conclusion

As we can see from the last plot, the lower MSE is obtained by the random forest, meaning that this is the best model for this data set. The results agree with what we expected: decision trees are easy to interpret but can be very non-robust and suffers from high variance (in other words, a small change in the data can cause a large change in the final estimated tree), while methods like random forest and boosting are more robust and their predictive performance is substantially improved.