hw5_g2

Jacopo Liera

2023-11-13

Exercise 4

In this exercise we want to assess the effects of the hyperparameters for a Random Forest predictor on the icu dataset. We set the seed for replication and get the data.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(aplore3)

set.seed(1391927)

#load data
data(icu, package = "aplore3")
head(icu)
```

```
##   id   sta age gender  race      ser can crn inf cpr sys hra pre      type fra
## 1  4  Died  87 Female White Surgical  No  No Yes  No  80  96  No Emergency Yes
## 2  8 Lived  27 Female White  Medical  No  No Yes  No 142  88  No Emergency  No
## 3 12 Lived  59   Male White  Medical  No  No  No  No 112  80 Yes Emergency  No
## 4 14 Lived  77   Male White Surgical  No  No  No  No 100  70  No  Elective  No
## 5 27  Died  76 Female White Surgical  No  No Yes  No 128  90 Yes Emergency  No
## 6 28 Lived  54   Male White  Medical  No  No Yes  No 142 103  No Emergency Yes
##     po2      ph   pco   bic    cre     loc
## 1 <= 60  < 7.25  > 45 >= 18 <= 2.0 Nothing
## 2  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 3  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 4  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 5  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 6  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
```

Secondly, we want to determine over which values we should do the tuning of the hyper-parameters. We set high number of values for ntree to see if there is a clear impact on how many trees we grow and the out of bag error, for which we can intuitively guess that it is not going to be as relevant as the number of variables sampled for each split.

```
X <- icu[, 3:ncol(icu)]
y <- icu$sta

mtry_vec <- seq(1, ncol(X))
ntree_vec <- c(100, 200, 400, 800, 1200)
```

We carry out the out of bag error estimates by modelling with Random Forest and then store the results in a matrix.

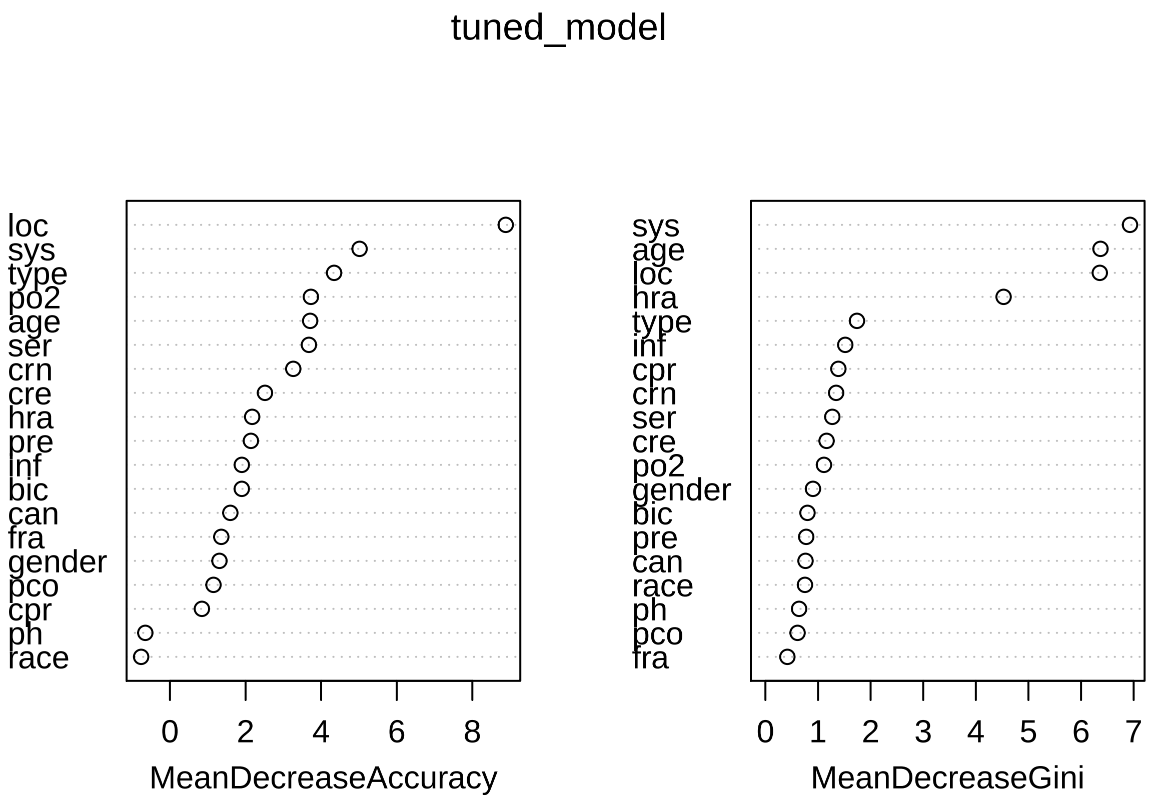
```
# Compute models
oob_errors <- function(X, y, h1 = mtry_vec, h2 = ntree_vec){

  # Matrix to store the out-of-bag errors for each combination of mtry and ntree
  oob_mat <- matrix(rep(0, length(ntree_vec) * length(mtry_vec)), ncol = length(ntree_vec))
  rownames(oob_mat) <- mtry_vec
  colnames(oob_mat) <- ntree_vec

  for(m in 1:length(mtry_vec)){
    for(n in 1:length(ntree_vec)){

      tmp <- randomForest(X, y,  mtry = h1[m], ntree = h2[n], proximity = TRUE)
      oob_mat[m, n] <- tmp$err.rate[nrow(tmp$err.rate), "OOB"]
    }
  }
  return(oob_mat)
}

errors <- oob_errors(X, y, h1 = mtry_vec, h2 = ntree_vec)
errors
```

```
##      100   200   400   800  1200
## 1  0.200 0.200 0.200 0.200 0.200
## 2  0.145 0.175 0.155 0.160 0.165
## 3  0.170 0.150 0.150 0.165 0.160
## 4  0.150 0.160 0.160 0.150 0.150
## 5  0.170 0.150 0.165 0.160 0.165
## 6  0.160 0.160 0.150 0.155 0.150
## 7  0.150 0.165 0.150 0.175 0.180
## 8  0.170 0.165 0.150 0.155 0.160
## 9  0.160 0.160 0.155 0.160 0.165
## 10 0.165 0.170 0.180 0.165 0.165
## 11 0.170 0.185 0.160 0.170 0.165
## 12 0.150 0.175 0.160 0.170 0.170
## 13 0.165 0.145 0.170 0.180 0.175
## 14 0.160 0.170 0.170 0.170 0.170
## 15 0.175 0.170 0.170 0.165 0.160
## 16 0.185 0.165 0.155 0.175 0.175
## 17 0.170 0.170 0.165 0.165 0.170
## 18 0.185 0.160 0.170 0.170 0.165
## 19 0.175 0.180 0.175 0.165 0.160
```

We should carry out this exercise many times to have an average estimate on the importance. In this specific iteration we see that already with 2 candidates for the split and really number of trees we get the minimum out of bag error. The second minimum is located with high number of candidates (13) and 200 trees, which is somewhat more informative.

```r
# Find best combination based on oob
best_combo <- which(errors == min(errors), arr.ind = TRUE)

# If multiple minima are achieved, just take the lowest amount of iterations
if(dim(best_combo)[1]>1) best_combo <- best_combo[1, ]

ntree <- as.integer(colnames(errors)[best_combo[2]])
mtry <- as.integer(rownames(errors)[best_combo[1]])
```

```r
#fit the model with the optimal values
tuned_model <- randomForest(x = X, y, mtry = mtry, ntree = ntree, importance = TRUE)
varImpPlot(tuned_model)
```



```r
sapply(X, is.numeric)
```

```
##    age gender   race    ser    can    crn    inf    cpr    sys    hra    pre
##   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE  FALSE
##   type    fra    po2     ph    pco    bic    cre    loc
##  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
```

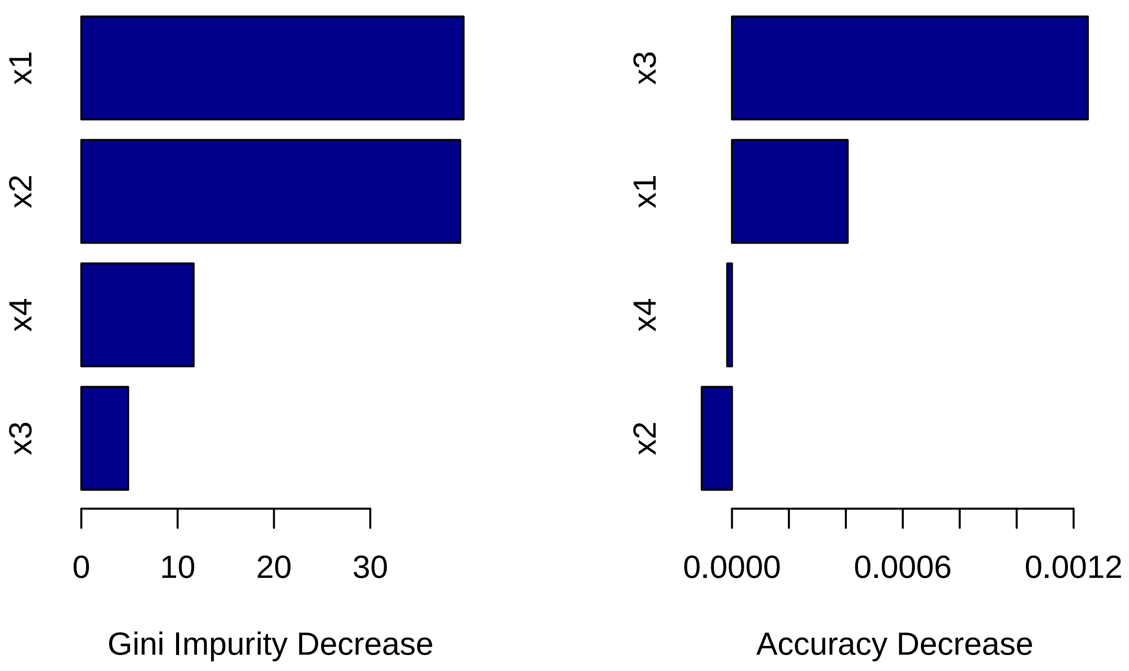
The difference between the two graphs is immediately clear if we look at the most important variable sys for gini and loc for mean decrease in accuracy. Gini Impurity is a measure which really just depends on the training data and makes no effort into making sure that such models would survive in a validation set, or based on out of bag errors. Hence, the use of numerical variables makes it easier to overfit the data in a splitting setting, hence why we see the differences.

Exercise 5

This task connects with the preceding one, in the sense that we will see differences in Gini and Accuracy decrease for each variable in the simulation study. We fit random forests algorithm with ntree = 100 which was previously the minimum in exercise 4. We see that once again that the Gini heavily relies to the training data and not on OOB samples. The accuracy measure is based on permutation feature importance, which consists in measuring the increase in the model error related to the permutation of a feature's values. In the end, all variables are ranked, where the most important ones are those which most contributed in rendering the model worse once their values have been shuffled. On the other hand, if by permuting the values of a feature, the model error does not increase, this would signify that the variable does not influence the model predictions. This measure is based on OOB mean squared error with permutated variables minues the normal OOB squared error, hence the difference in results.

```
library(randomForest)

datagen <- function(N){

  # Predictors
  x1 <- rnorm(N, mean = 0, sd = 1)
  x2 <- runif(N, min = 0, max = 1)
  x3 <- sample(1:2, N, replace = TRUE)
  x4 <- sample(1:5, N, replace = TRUE)

  # Dependent variable
  y <- as.factor(sample(0:1, N, replace = TRUE))

  return(data.frame(y, x1, x2, x3, x4))
}

VI_calculate <- function(x){

  model <- randomForest(y ~ ., data = x, importance = TRUE, ntree = 100)

  accuracy <- model$importance[,"MeanDecreaseAccuracy"]
  gini <- model$importance[, "MeanDecreaseGini"]

  return(list(gini = gini, accuracy = accuracy))
}

datasets <- replicate(100, datagen(N = 200), simplify = FALSE)
res <- lapply(datasets, VI_calculate)

Gini <- sapply(res, function(x) x$gini)
Accuracy <- sapply(res, function(x) x$accuracy)

mean_accuracy <- rowMeans(Accuracy)
mean_gini <- rowMeans(Gini)

par(mfrow = c(1, 2))
```

```r
barplot(mean_gini[order(mean_gini)], horiz = TRUE, xlab = "Gini Impurity Decrease", col = "darkblue")
barplot(mean_accuracy[order(mean_accuracy)], horiz = TRUE, xlab = "Accuracy Decrease", col = "darkblue"
```


x1
x2
x4
x3
Gini Impurity Decrease
x3
x1
x4
x2
Accuracy Decrease