Introduction to classification and regression trees, random forests and model-based recursive partitioning in R

Day 2: Stability and tree ensembles

Exercise 3: Glaucoma data revisited (2)

a) Fit a random forest using the `cforest` function from package `party`, with either bootstrap sampling with size N or subsampling with size $.632 \cdot N$.

```
library(party)
data("GlaucomaM", package = "TH.data")
cfor_contr_bs <- cforest_control(replace = TRUE)
cfor_contr_ss <- cforest_control(replace = FALSE, fraction = 0.632)

set.seed(2908)
for_glauc_bs <- cforest(Class ~ ., data = GlaucomaM, control = cfor_contr_bs)
set.seed(21)
for_glauc_ss <- cforest(Class ~ ., data = GlaucomaM, control = cfor_contr_ss)
```

b) Using the `predict()` function, compute the percentage of correctly classified observations — based on all trees and based only on those trees for which the observations are out-of-bag (by adding `OOB = FALSE` or `OOB = TRUE` to the function call) — and compare your results. Determine the predicted response class of each case and create a confusion matrix.

```
# calculate training errors:
#
# bootstrap sampling
predicted_class <- predict(for_glauc_bs)
table(GlaucomaM$Class, predicted_class)
```

```
##           predicted_class
##            glaucoma normal
##   glaucoma       86     12
##   normal          4     94
```

```
# subsampling
predicted_class <- predict(for_glauc_ss)
table(GlaucomaM$Class, predicted_class)
```

```
##           predicted_class
##            glaucoma normal
##   glaucoma       84     14
##   normal          5     93
```

```
sum(GlaucomaM$Class == predicted_class)/nrow(GlaucomaM)
```

```
## [1] 0.9030612
```

```
# OOB errors
#
# bootstrap sampling
predicted_class <- predict(for_glauc_bs, OOB = TRUE)
table(GlaucomaM$Class, predicted_class)
```

```
##           predicted_class
##            glaucoma normal
##   glaucoma       79      19
##   normal         15      83
```

```
#
# subsampling
predicted_class <- predict(for_glauc_ss, OOB = TRUE)
table(GlaucomaM$Class, predicted_class)
```

```
##           predicted_class
##            glaucoma normal
##   glaucoma       76     22
##   normal         10     88
```

```
sum(GlaucomaM$Class==predicted_class)/nrow(GlaucomaM)
```

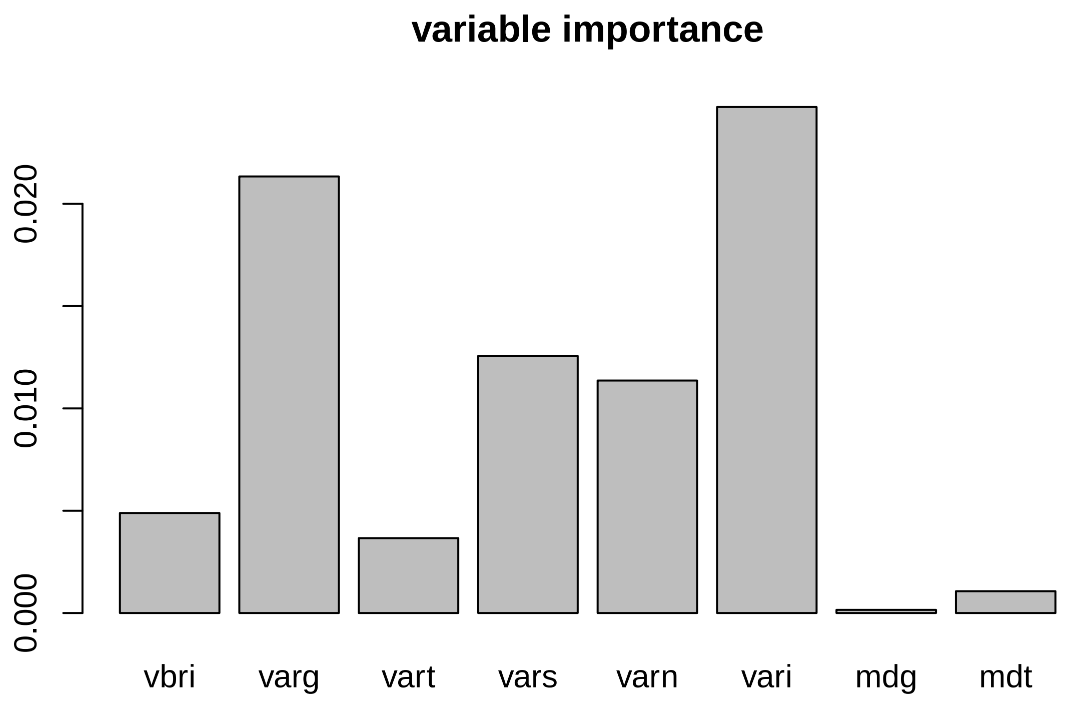
```
## [1] 0.8367347
```

c) Compute and plot the standard permutation importance for the predictor variables.

```
set.seed(14)
imp <- varimp(for_glauc_bs)
imp[order(imp, decreasing = TRUE)]
```

```
##          vari          varg          vars          varn           tmg
##  2.473086e-02  2.133656e-02  1.256608e-02  1.136261e-02  8.918427e-03
##          phci           hic           tmi          mhci           rnf
##  8.809597e-03  7.904260e-03  7.788858e-03  7.385486e-03  7.355221e-03
##          phcg           tms          abri          mdic          abrs
##  6.945339e-03  6.389034e-03  6.312554e-03  5.436358e-03  5.119845e-03
##          vbri          mhcg          phcn          vart           hvc
##  4.888464e-03  4.000060e-03  3.968590e-03  3.657933e-03  3.077886e-03
##           emd          abrg            as          mhct            ag
##  2.682900e-03  2.475112e-03  1.977648e-03  1.475119e-03  1.395082e-03
##          vass           tmt          mhcs          vbrg          vbsi
##  1.287167e-03  1.218121e-03  1.204970e-03  1.172589e-03  1.122333e-03
##           mdn           tmn            ai           mdt          vasg
##  1.114135e-03  1.088222e-03  1.067874e-03  1.066057e-03  1.041898e-03
##          vbrs            at           eas          phcs          vbsg
##  1.039984e-03  1.021327e-03  9.937465e-04  8.209638e-04  8.047762e-04
##          abrn          phct          vbst            mr            an
##  7.697780e-04  7.635571e-04  6.894481e-04  6.429624e-04  6.317809e-04
##          vbrn          mhcn          vbss           mdi          vasi
##  5.379197e-04  5.239234e-04  5.154076e-04  5.045643e-04  5.044782e-04
##           eat           eag          abrt           eai          vast
##  4.305072e-04  3.883825e-04  3.855851e-04  3.524894e-04  3.418676e-04
##          vbrt           mdg           mds          vbsn            mv
##  3.186083e-04  1.548512e-04  1.379876e-04  6.698471e-05 -1.158468e-04
##          vasn           ean
## -1.915117e-04 -3.556480e-04
```

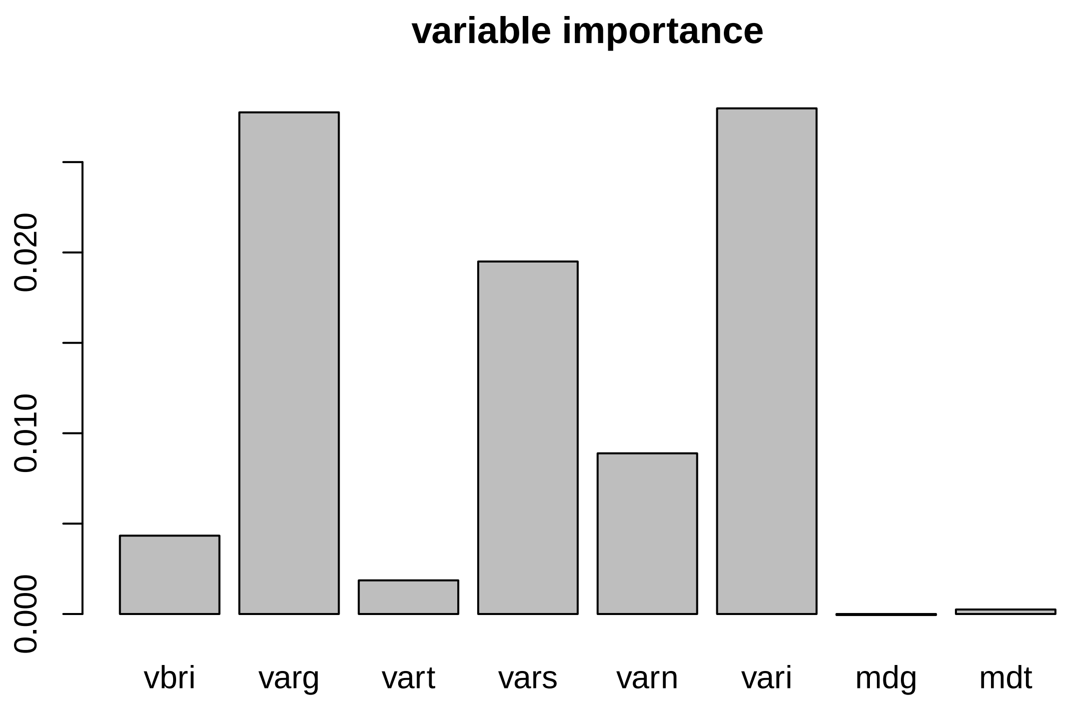
```
# plot importances for only a subset of the variables:
barplot(imp[42:49], names.arg = names(GlaucomaM[,42:49]), main = "variable importance")
```



```r
set.seed(15)
# plot importacnes for only a subset of the variables:
imp <- varimp(for_glauc_ss)
imp[order(imp, decreasing = TRUE)]
```

```
##         vari         varg         vars          tmg          tmi
## 2.797222e-02 2.775000e-02 1.950000e-02 1.197222e-02 1.025000e-02
##          hic          tms         varn         phcg         phci
## 1.002778e-02 9.916667e-03 8.888889e-03 8.444444e-03 7.833333e-03
##         mhci         abri          rnf         mdic         vbri
## 6.500000e-03 6.361111e-03 6.333333e-03 5.166667e-03 4.333333e-03
##         phcn         abrg         abrs         vbsi         vbrs
## 3.638889e-03 3.277778e-03 3.250000e-03 3.083333e-03 2.583333e-03
##          hvc         mhcn         mhcg          emd         mhcs
## 2.277778e-03 2.138889e-03 2.111111e-03 1.972222e-03 1.861111e-03
##         vart          tmt         vasi         phcs           ai
## 1.861111e-03 1.833333e-03 1.444444e-03 1.361111e-03 1.250000e-03
##           mr         vasg         vass           an         vbrg
## 1.166667e-03 1.027778e-03 9.722222e-04 7.777778e-04 7.500000e-04
##          ean           as          mdn          tmn         abrn
## 6.944444e-04 6.666667e-04 6.388889e-04 6.388889e-04 6.111111e-04
##         vbsn         vbsg         abrt           ag         phct
## 5.833333e-04 5.555556e-04 4.722222e-04 4.722222e-04 4.722222e-04
##          eag         vbrn           mv          eai         vbst
## 4.722222e-04 3.888889e-04 3.055556e-04 2.777778e-04 2.500000e-04
##         vbss          mdt           at          mds         vast
##  2.500000e-04  2.500000e-04  2.500000e-04  1.944444e-04  1.944444e-04
##           eat          vbrt           eas           mdi          vasn
##  1.388889e-04  8.333333e-05  8.333333e-05 -5.551115e-20 -2.777778e-05
##           mdg          mhct
## -5.555556e-05 -5.555556e-05
```

```
barplot(imp[42:49], names.arg = names(GlaucomaM[,42:49]), main = "variable importance")
```



```
# Note: Before interpreting any variable importances make sure that
# your results are stable by means of setting two different
# random seeds, before fitting the forest and computing the
# variable importance. If the ranking of the top variable importances
# changes between the two runs, this means that your forest was not
# large enough (as compared to the number of variables) to generate
# stable results and you should increase the number of trees in the
# forest (ntree).
```

d) Do the `stabletree()` function and 'cforest()' function give the same 3 most important variables?

Set random seeds everywhere in the code where random sampling is involved.

```
# According to the stablelearner analyses, vari, vasg and varg appear most often
# in trees build on bootstrap datasets.
# According to the random forest importances, vari, varg and vars are most
# important.
# The methods do no completely agree, but the number of times a variables appears
# in trees grown on random samples of the data may not be the best indicator of
# influence (importance), and can be expected to yield different results than
```

```r
# calculating variable importances in a random forest.
```

Exercise 4: Carseat sales revisited

Using the `cforest()` function from the `party` package, fit a random forest and a bagged ensemble to the 300 training observations. Using the `predict()` function, calculate the mean squared error (MSE) for:

$$MSE = \frac{1}{N} \sum_i (y_i - \hat{y}_i)^2$$

a) The training observations

```r
data("Carseats", package = "ISLR")
set.seed(42)
train <- sample(1:400, 300)
library(party)
bagged_cs <- cforest(Sales ~ ., data = Carseats[train,], controls = cforest_unbiased(mtry = NULL))
ranfor_cs <- cforest(Sales ~ ., data = Carseats[train,])

mean((Carseats[train,"Sales"] - predict(bagged_cs))^2)
```

```
## [1] 2.10367
```

```r
mean((Carseats[train,"Sales"] - predict(ranfor_cs))^2)
```

```
## [1] 2.241822
```

b) The OOB predictions for training observations (add `OOB = TRUE` to the function call)

```r
mean((Carseats[train,"Sales"] - predict(bagged_cs, OOB = TRUE))^2)
```

```
## [1] 3.204182
```

```r
mean((Carseats[train,"Sales"] - predict(ranfor_cs, OOB = TRUE))^2)
```

```
## [1] 3.174919
```

c) The test observations (specify `newdata` argument)

```r
mean((Carseats[-train,"Sales"] - predict(bagged_cs, newdata = Carseats[-train,]))^2)
```

```
## [1] 3.237652
```

```r
mean((Carseats[-train,"Sales"] - predict(ranfor_cs, newdata = Carseats[-train,]))^2)
```

```
## [1] 3.389324
```

d) Are the OOB training error estimates indeed less optimistic / more realistic estimates of the test error?

```r
# Yes, the error estimated based on OOB observations more closely resembles the error on test data
```

Exercise 5: Boston housing data

Fit a random forest and bagged ensemble of CART trees to the Boston housing data. This dataset reports the median value (`medv`) of owner-occupied homes in 506 districts in the Boston area (USA), together with 13 potential predictor variables reporting sociodemographic and other characteristics of the districts:

```r
library("randomForest")
```

```
## randomForest 4.6-12

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

```r
?randomForest
```

```
## starting httpd help server ...

##  done
```

```r
data("Boston", package = "MASS")
?MASS::Boston
```

Use the `randomForest()` function from the `randomForest` package to predict the median value of owner occupied home. Fit bth a random forest and a bagged ensemble. You could create the ensembles using a formula statement (i.e., `medv ~ .`), but if you supply arguments `x`, `y`, `xtest` and `ytest`, you can evaluate the test error for random forests consisting of 1 through the specified number of trees to grow. You will use these estimates to evaluate the effect of the number of trees on accuracy and to compare the OOB error estimates with the error estimated on test data. Therefore, first separate the data into a training and test set:

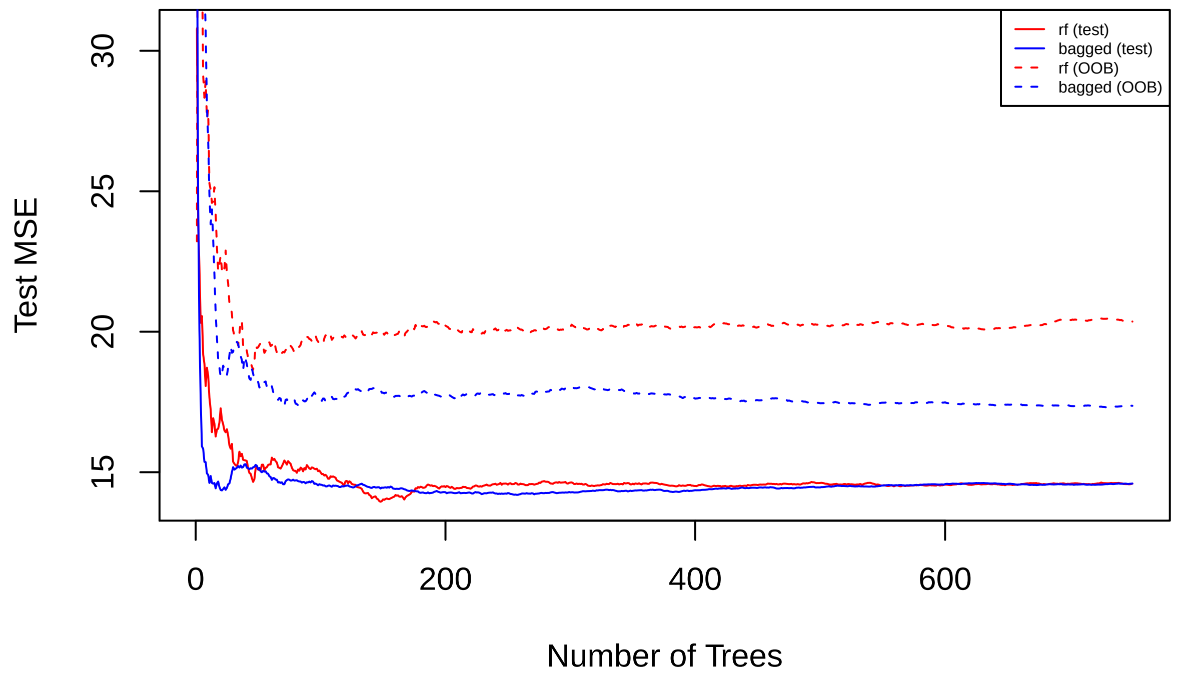
```r
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/4)
x <- Boston[train, -14]
y <- Boston[train, 14]
xtest <- Boston[-train, -14]
ytest <- Boston[-train, 14]
```

a) Set the `ntree` argument to 750, and create two ensembles: a bagged ensemble, using all 13 predictor variables for selecting each split, and a random forest using `sqrt(13)` for selecting each split.

```r
set.seed(1)
rf.boston <- randomForest(x = x, y = y, xtest = xtest, ytest = ytest, mtry = sqrt(13),
                          ntree = 750, data = Boston)
bag.boston <- randomForest(x = x, y = y, xtest = xtest, ytest = ytest, mtry = 13,
                           ntree = 750)
```

b) Inspect the test error (which can be found in the `$test$mse` slot of the randomForest object) and the OOB error (which can be found in the `$mse` slot of the randomForest object) for both ensembles. Create a plot where the number of trees in the ensemble is on the x-axis, and the MSE is on the y-axis. Draw different lines for the OOB and test error of the bagged ensemble, as well as the OOB and test error of the random forest. (Hint: use functions `plot()` and `lines()` and the `type`, `lty` and `col` arguments)

```r
plot(1:750, rf.boston$test$mse, col = "red", type = "l", xlab = "Number of Trees",
     ylab = "Test MSE")
lines(1:750, rf.boston$mse, col = "red", type = "l", lty = 2)
lines(1:750, bag.boston$test$mse, col = "blue", type = "l")
lines(1:750, bag.boston$mse, col = "blue", type = "l", lty = 2)
legend("topright", paste(rep(c("rf", "bagged"), times=2),
                         rep(c("(test)", "(OOB)"), each = 2)),
       col = rep(c("red", "blue"), times = 2), cex = .5, lty = rep(1:2, each = 2))
```



c) After which number of trees does the MSE start to stabilize?

```
# After about 200 trees.
```

d) Which of the two ensembles is most accurate on the test data?

```
# If we would use 600-750 trees for prediction, the bagged and ranfom forest
# ensemble seem equally accurate.
```

e) Does the OOB error give a better estimate of test error for the random forest, or for the bagged ensemble? Can you explain the difference?

```
# The OOB error gives an estimate of prediction error that is closer to the test
# error for the bagged ensemble (at least, in this case it does).
#
# The OOB error overestimates the test error more for the random forest than
# for the bagged ensemble. This may be because the individual trees in the random
# forest may be somewhat less accurate, because they cannot always pick up the
# best predictor variables (due to the random selection of predictor variables
# as splitting candidates). As a result, the OOB error may be somewhat higher
# (less optimistic) for random forests than for bagged ensembles.
```