Exercise 11

Business Analytics and Data Science WS18/19

Introduction

With increasing complexity of models interpretability becomes an issue. 'Old-school' models like regression or tree-based family are still around not only because of their robustness, but also due to insights they provide on the feature importance. Neural Networks can boast impressive predictive power, yet they remain mostly a 'black box' (although there are extensive attempts to fix it). Both model-dependent and -independent tools have been developed in order to enhance the our understanding of the internal processes hapenning inside the model. We will use some of these to answer the important questions which variables are important and what is the size and direction of the influence of the variables.

Logit and steps

In many cases you will find yourself using a linear regression to get a quick look at the interactions within data. There are several methods that would allow you to make a judgement about the variable importance, starting with size and p-values of coefficients and ending with automated tools that consider all possible subsets of the pool of explanatory variables and find the model that best fits the data according to some criteria (think Adjusted R2, AIC and BIC). In the process different combinations get scored and you end up with the set of variables deemed optimal for your criteria of choice.

stepAIC function from the package **MASS* makes use of Akaike Information Criterion to compare models. It's universal for regression and classification tasks. If you increase the number of parameters while fitting the model, you will improve the log likelihood (we will refresh what that is in the tutorial) but will run into the danger of overfitting. The AIC penalizes for increasing the number of parameters thus minimizing the AIC selects the model where the improvement in log likelihood is no longer worth the penalty for increasing the number of parameters. In a way, it does similar work with our regularization technics like LASSO and Ridge.

1. Start with building several regressions that use an increasing number variables to predict BAD. Loop over variables 1,2,3,… , incrementally add them to the model and see what happens to your AUC. Plot the results. What is a drawback of such manual looping?
2. Use *stepAIC* function to perform a step-wise regression and see what is the best AIC you can get (refresh your knowledge of AIC if needed). To do it, you will need to build two models first: one logit with only Intercept as an aexplanatory variable and then a full logit. Save the variables selected by stepAIC - we want to compare it with RF results later on.

Note: We will perform the feature selection on train set and predict on validation set. Test set should be used after feature selection.

```r
library(leaps)
library(MASS)
library(caret)
library(hmeasure)
library(randomForest)
library(xgboost)

source("BADS-HelperFunctions.R")
loans <- get.loan.dataset()

# Splitting the data into a test and a training set
# and now an additional woe set
set.seed(123)
idx.test <- createDataPartition(y = loans$BAD, p = 0.3, list = FALSE) # Draw a random, stratified sampl
```

```r
ts <-  loans[idx.test, ] # test set - put it aside

idx.val <- createDataPartition(y = loans$BAD[-idx.test], p = 0.35, list = FALSE)
val <- loans[-idx.test, ][idx.val,]
tr <- loans[-idx.test, ][-idx.val,] # training set

auc <- rep(NA, 14)
#pulling column names
varn <- colnames(loans)[-c(15,16)]

for (i in 1:length(varn)){
  # Add the (arbitrarily) first i variables in the dataset
 Formula <- formula(paste("BAD ~ ", paste(varn[1:i], collapse=" + ")))
 # Train a logit model
 lm <- glm(Formula,tr,family="binomial")
 yhat <- predict(lm, val[,c(1:i,15)],type="response")
 # Calculate the AUC score on the test data
 h <- HMeasure(true.class = as.numeric(val$BAD=="bad"), scores = yhat)
 auc[i]<- h$metrics["AUC"]
}
plot(unlist(auc),type="l")
```

```r
basic <- glm(BAD~1, data=tr,family = "binomial")
basic# this will give us only the intercept
```

```
##
## Call:  glm(formula = BAD ~ 1, family = "binomial", data = tr)
##
## Coefficients:
## (Intercept)
##      -1.033
##
## Degrees of Freedom: 555 Total (i.e. Null);  555 Residual
## Null Deviance:       640.2
## Residual Deviance: 640.2     AIC: 642.2
```

```r
full <- glm(BAD~., data=tr,family = "binomial")
# you can select the direction of selection, you would then look for a model with lowest AIC.
glm_stepwise <- stepAIC(basic, scope = list(lower = basic, upper = full), direction = "both", trace = T
```

```
## Start:  AIC=642.23
## BAD ~ 1
##
##              Df Deviance    AIC
## + dOUTCC      1   622.34 626.34
## + dINC_A      1   623.12 627.12
## + EMPS_A     10   612.57 634.57
## + YOB         1   632.67 636.67
## + RES         4   628.51 638.51
## + dOUTHP      1   635.35 639.35
## + nKIDS       1   636.08 640.08
## + dOUTL       1   637.66 641.66
## + YOB_missing 1   637.69 641.69
## <none>            640.23 642.23
## + dOUTM       1   638.47 642.47
```

```
## + dINC_SP      1   639.31 643.31
## + nDEP         1   639.64 643.64
## + dMBO         1   639.74 643.74
## + dHVAL        1   639.91 643.91
## + PHON         1   639.98 643.98
##
## Step:  AIC=626.34
## BAD ~ dOUTCC
##
##              Df Deviance    AIC
## + dINC_A      1   610.08 616.08
## + YOB         1   614.64 620.64
## + RES         4   610.62 622.62
## + dOUTHP      1   616.86 622.86
## + EMPS_A     10   599.12 623.12
## + YOB_missing  1   618.70 624.70
## + nKIDS       1   619.84 625.84
## <none>            622.34 626.34
## + dOUTL       1   621.27 627.27
## + dOUTM       1   621.79 627.79
## + dMBO        1   622.13 628.13
## + dINC_SP     1   622.14 628.14
## + nDEP        1   622.15 628.15
## + PHON        1   622.27 628.27
## + dHVAL       1   622.34 628.34
## - dOUTCC      1   640.23 642.23
##
## Step:  AIC=616.08
## BAD ~ dOUTCC + dINC_A
##
##              Df Deviance    AIC
## + RES         4   595.41 609.41
## + YOB         1   601.88 609.88
## + dOUTHP      1   606.70 614.70
## + YOB_missing  1   607.91 615.91
## <none>            610.08 616.08
## + dHVAL       1   609.14 617.14
## + nKIDS       1   609.42 617.42
## + dMBO        1   609.73 617.73
## + dOUTL       1   609.88 617.88
## + nDEP        1   609.93 617.93
## + dOUTM       1   609.98 617.98
## + dINC_SP     1   610.01 618.01
## + PHON        1   610.05 618.05
## + EMPS_A     10   595.41 621.41
## - dINC_A      1   622.34 626.34
## - dOUTCC      1   623.12 627.12
##
## Step:  AIC=609.41
## BAD ~ dOUTCC + dINC_A + RES
##
##              Df Deviance    AIC
## + dOUTHP      1   592.08 608.08
## + YOB         1   592.67 608.67
```

```
## + nKIDS        1   592.75 608.75
## <none>             595.41 609.41
## + dINC_SP      1   594.05 610.05
## + YOB_missing  1   594.38 610.38
## + dOUTL        1   594.73 610.73
## + dHVAL        1   594.82 610.82
## + dMBO         1   594.98 610.98
## + PHON         1   594.99 610.99
## + nDEP         1   595.28 611.28
## + dOUTM        1   595.40 611.40
## - RES          4   610.08 616.08
## + EMPS_A      10   586.64 620.64
## - dOUTCC       1   608.85 620.85
## - dINC_A       1   610.62 622.62
##
## Step:  AIC=608.08
## BAD ~ dOUTCC + dINC_A + RES + dOUTHP
##
##               Df Deviance    AIC
## + YOB          1   589.08 607.08
## + nKIDS        1   589.10 607.10
## <none>             592.08 608.08
## + dINC_SP      1   590.70 608.70
## + YOB_missing  1   591.08 609.08
## + dOUTL        1   591.22 609.22
## - dOUTHP       1   595.41 609.41
## + PHON         1   591.59 609.59
## + dMBO         1   591.63 609.63
## + dHVAL        1   591.70 609.70
## + nDEP         1   591.98 609.98
## + dOUTM        1   592.05 610.05
## - RES          4   606.70 614.70
## + EMPS_A      10   583.11 619.11
## - dINC_A       1   605.19 619.19
## - dOUTCC       1   606.31 620.31
##
## Step:  AIC=607.08
## BAD ~ dOUTCC + dINC_A + RES + dOUTHP + YOB
##
##               Df Deviance    AIC
## + nKIDS        1   586.25 606.25
## <none>             589.08 607.08
## + YOB_missing  1   587.47 607.47
## - RES          4   597.63 607.63
## + dINC_SP      1   588.02 608.02
## - YOB          1   592.08 608.08
## + dOUTL        1   588.23 608.23
## - dOUTHP       1   592.67 608.67
## + PHON         1   588.92 608.92
## + dOUTM        1   588.99 608.99
## + dHVAL        1   588.99 608.99
## + nDEP         1   589.01 609.01
## + dMBO         1   589.01 609.01
## - dINC_A       1   600.40 616.40
```

```
## - dOUTCC       1   602.87 618.87
## + EMPS_A      10   582.06 620.06
##
## Step:  AIC=606.25
## BAD ~ dOUTCC + dINC_A + RES + dOUTHP + YOB + nKIDS
##
##               Df Deviance    AIC
## <none>             586.25 606.25
## + YOB_missing  1   584.84 606.84
## - nKIDS        1   589.08 607.08
## - YOB          1   589.10 607.10
## + dINC_SP      1   585.24 607.24
## + dOUTL        1   585.50 607.50
## - RES          4   595.83 607.83
## + PHON         1   585.95 607.95
## + dOUTM        1   586.10 608.10
## + dHVAL        1   586.15 608.15
## - dOUTHP       1   590.18 608.18
## + nDEP         1   586.24 608.24
## + dMBO         1   586.25 608.25
## - dINC_A       1   595.72 613.72
## - dOUTCC       1   599.37 617.37
## + EMPS_A      10   578.90 618.90
```

```r
#glm_stepwise <- stepAIC(full, direction = "backward", trace = TRUE, steps = 100)
stepvar <- glm_stepwise$coefficients # Check and save the selected variables

performance <- HMeasure(true.class=as.numeric(val$BAD=="bad"), scores = cbind(
  "full" = predict(full, newdata = val, type="response"),
  "stepwise" = predict(glm_stepwise, newdata = val, type="response")))

performance$metrics["AUC"]
```

```
##                AUC
## full     0.5992647
## stepwise 0.6414310
```

Variable importance for tree-based models

For both the random forest and the gradient boosting model, we can calculate which variables have the largest influence on the prediction. This *variable importance* is often model-based, i.e. caluclated in a specific way for a certain model. Two measures of variable importance, one for all tree-based models and one specific to random forests, will be discussed in detail in the lecture. For other models or approaches that are not model dependent, see the caret page on [variable importance]{http://topepo.github.io/caret/variable-importance.html} or the recommended literature.
- *Tree-based Gini importance*: The mean squared relative importance of each variable is the sum of squared improvement in the error risk over all internal nodes for which it was chosen as the splitting variable, averaged over all trees.
- *Random forest OOB importance*: The decrease in accuracy when randomly permuting the values of each variable in turn for each tree, averaged over all trees. The test sample for each tree are the observations not contained in the bootstrap training set for that tree a.k.a. out-of-bag observations.

Note: These measures do not capture the effect on prediction in case a variable were not available, because other variables could be used as surrogates.
Note: The importance of highly correlated variables will not be accurate. Expect RF to split the importance

between correlated features and boosting to focus on one of them.

1. Train a random forest model and gradient boosted trees. For random forest, set **importance = TRUE** to calculate the performance on the out-of-bag samples.
2. Calculate the variable importance of the random forest and the gradient boosting model using the package specific importance functions or mlr's **getFeatureImportance()**. How are the respective importance values calculated for the random forest and gradient boosting model?
3. Sort the variable importance for both models and remember the most important variables. Does the importance order of the variables fit your expectation?

```
library(rpart)
Accuracy <- function(prediction, class, threshold = 0.5){
  # Predict class 1 if prob. is higher than threshold
  predClass <-  ifelse(prediction > 0.5, levels(class)[2], levels(class)[1])
  # Accuracy = ratio of predictions equal to actual observations
  acc <- sum(factor(predClass) == class) / length(class)
  return(acc)
}

set.seed(123)
boot <- sample(1:nrow(loans), 1000, replace=TRUE) #Take bootstrap sample(replace = TRUE)
bootstrap <- loans[boot,]
temptest <- loans[-boot,] #put samples not in bootstrap into temporary test set
# Build tree. Remember that random forest builds randomized trees instead
dt <- rpart(BAD~., bootstrap, method="class")
# Predict on temporary test set
yhat <- predict(dt, temptest, type = "prob")[,2]
acctotal <- Accuracy(prediction = yhat, temptest$BAD, threshold = 0.5)

acc <- c()
#bootidx <- sample(1:nrow(loans), 1000, replace=TRUE) #Take bootstrap sample(replace = TRUE)
for (i in 1:14){
temptest_permuted <- temptest
temptest_permuted[,i]<- sample(temptest[,i],replace=FALSE)
yhat <- predict(dt,temptest_permuted,type = "prob")[,2]
acc[i] <- Accuracy(prediction = yhat,temptest$BAD, threshold = 0.5)
}
plot(acc,type="l")
abline(h=acctotal,col="red")
#In case of the RF application, the algorithm would run this loop for every tree and further calculate

dt_var_importance <- acctotal-acc
names(dt_var_importance) <- colnames(loans)[1:14]
```

```
# Explicitly transform factor variables to dummy variables for the xgboost implementation of gradient b
tr.dummy <- mlr::createDummyFeatures(tr, target = "BAD")
val.dummy <- mlr::createDummyFeatures(val, target = "BAD")
ts.dummy <- mlr::createDummyFeatures(ts, target = "BAD")

# Train random forest with the optimal parameters found before

set.seed(123)
rf.randomForest <- randomForest(BAD~., data = tr,
                                method = "rf", ntree = 1000, mtry = 4, sampsize = 200,
                                importance = TRUE)
```

```r
varImpPlot(rf.randomForest,type=1)
# mlr random forest for illustration
library("mlr")
task <- makeClassifTask(data = tr.dummy, target = "BAD", positive = "bad")
rf.mlr <- makeLearner("classif.randomForest", predict.type = "prob",
                      par.vals = list("replace" = TRUE, "importance" = TRUE,
                                      "mtry" = 4, "sampsize" = 200, "ntree" = 1000))
set.seed(123)
rf <- mlr::train(rf.mlr, task = task)

### Random forest variable importance
featureImportance <- list()
# Remember to set importance = TRUE in the call to randomForest() to calculate OOB importance

# To use importance() from the randomForest package on mlr train objects (e.g. rf.mlr) extract rf.mlr$l
# Remember that mlr is only a wrapper function around the underlying packages, so
# mlr will return a "randomForest" object with its other results
class(rf$learner.model)
```

```
## [1] "randomForest.formula" "randomForest"
```

```r
importance(rf$learner.model)
```

```
##                   good         bad MeanDecreaseAccuracy MeanDecreaseGini
## YOB         11.5787959 -1.18531191           12.2630911        8.3500439
## nKIDS       10.3638278 -6.79153338            8.1741279        2.2873953
## nDEP        -1.1138384 -1.25335573           -1.5274792        0.3563888
## PHON         3.3083348 -4.19265916            0.7907029        1.0710568
## dINC_SP     14.1404970 -3.63630979           12.4902988        3.4260521
## dINC_A       3.3837915  8.06457338            8.0757982        8.8690689
## dHVAL        9.2342087 -2.47432768            9.7974445        4.8509790
## dMBO        13.2445356 -5.18843951           12.9169236        4.0355202
## dOUTM        2.4126178 -0.04732355            2.5942919        5.5199921
## dOUTL        1.8668235  3.45624441            3.7690934        3.3850544
## dOUTHP      -0.8357809  3.88586904            1.2233322        1.8872714
## dOUTCC       2.0831276 11.56843286            7.9016575        1.9617549
## YOB_missing  1.9906217  7.34576202            6.6372524        0.3131536
## EMPS_A.B    -3.0917658 -2.47821551           -3.7555091        0.3794149
## EMPS_A.E    -0.1081795  2.06915988            1.0833017        1.0455713
## EMPS_A.M     1.8777429  4.10573971            3.6165594        0.5539091
## EMPS_A.N    -1.6885176 -2.35379476           -2.5324903        0.1438926
## EMPS_A.P     0.6394153  6.53995718            4.6081206        1.4396180
## EMPS_A.R     2.3268148  0.40888924            3.0810005        0.8938455
## EMPS_A.T     3.3044123 -5.63939565            0.6468175        0.5619437
## EMPS_A.U    -1.9019284 -1.93902410           -2.3498966        0.1164475
## EMPS_A.V     3.4123545 -5.18868802            0.5704173        1.0094789
## EMPS_A.W     6.0886247 11.69582161           12.0939742        0.9383283
## EMPS_A.Z     3.1849493  8.27250979            8.2075879        0.3215735
## RES.F       -1.4659437  0.29411206           -1.4046544        0.6693483
## RES.N        4.4663597  6.32670448            8.3163863        1.3366629
## RES.O        8.4588070 -6.61519896            7.9296101        1.0078089
## RES.P        6.1302604 -6.10835033            4.9179191        0.8767425
## RES.U        2.4731578 -2.11806483            1.6448396        0.7940628
```

```r
importance(rf.randomForest)
```

```
##                   good        bad MeanDecreaseAccuracy MeanDecreaseGini
## YOB         11.2552762 -8.3598397            7.7836359       11.5083928
## nKIDS       11.9057838 -8.1235351            9.2314778        2.7414845
## nDEP        -1.5701996 -5.0320673           -3.4647495        0.4398982
## PHON         7.0478703 -5.6501456            3.5824761        1.3010799
## dINC_SP     15.1285259 -4.7161987           12.9405050        3.9842774
## EMPS_A      16.1486625  1.5220985           18.0885842        8.3509540
## dINC_A       7.0835045  5.6725682           10.5976105       11.9160629
## RES         10.5036490 -6.9603696            9.0642206        4.6627733
## dHVAL        8.9649744 -1.3436972            9.5899234        6.2345457
## dMBO        13.6978150 -7.1162341           12.4183865        4.8560302
## dOUTM        0.5928785  0.3583566            0.8917971        7.2816847
## dOUTL        1.4149798  2.2481676            2.7409359        3.9727319
## dOUTHP      -1.5029421  2.4283047           -0.2240645        2.0438575
## dOUTCC       1.5916999 11.8434762            7.1142161        2.0477543
## YOB_missing  0.8602736  2.5192318            2.6536131        0.2632596
```

```r
# Naturally, you can use the mlr wrapper function to calculate the variable importance
# The specific measure for variable importance (e.g. MeanDecreaseGini, OOB descrease in accuracy)
# is selected with argument 'type'
getFeatureImportance(rf, type = 2)
```

```
## FeatureImportance:
## Task: tr.dummy
##
## Learner: classif.randomForest
## Measure: NA
## Contrast: NA
## Aggregation: function (x)  x
## Replace: NA
## Number of Monte-Carlo iterations: NA
## Local: FALSE
##        YOB    nKIDS      nDEP     PHON  dINC_SP   dINC_A    dHVAL    dMBO
## 1 8.350044 2.287395 0.3563888 1.071057 3.426052 8.869069 4.850979 4.03552
##       dOUTM    dOUTL   dOUTHP   dOUTCC YOB_missing  EMPS_A.B EMPS_A.E
## 1 5.519992 3.385054 1.887271 1.961755   0.3131536 0.3794149 1.045571
##     EMPS_A.M  EMPS_A.N EMPS_A.P  EMPS_A.R  EMPS_A.T  EMPS_A.U EMPS_A.V
## 1 0.5539091 0.1438926 1.439618 0.8938455 0.5619437 0.1164475 1.009479
##     EMPS_A.W  EMPS_A.Z     RES.F    RES.N    RES.O     RES.P     RES.U
## 1 0.9383283 0.3215735 0.6693483 1.336663 1.007809 0.8767425 0.7940628
```

```r
featureImportance[["rf"]] <- unlist(getFeatureImportance(rf, type = 2)$res)
# Although the GINI and OOB importance should be somewhat consistent, there is no gurantee
# that they are. That is, one variable may be considered important by one
# measure but less important by another. If in doubt, the OOB-based
# measure (meanDecreaseAccuracy) is considered more reliable than the Gini-based
# measure (meanDecreaseImpurity)
rf.importance <- importance(rf.randomForest)
row.names(rf.importance[order(rf.importance[,"MeanDecreaseAccuracy"], decreasing = TRUE),])
```

```
##  [1] "EMPS_A"      "dINC_SP"     "dMBO"        "dINC_A"      "dHVAL"
##  [6] "nKIDS"       "RES"         "YOB"         "dOUTCC"      "PHON"
## [11] "dOUTL"       "YOB_missing" "dOUTM"       "dOUTHP"      "nDEP"
```

```r
row.names(rf.importance[order(rf.importance[,"MeanDecreaseGini"], decreasing = TRUE),])
```

```
##  [1] "dINC_A"      "YOB"         "EMPS_A"      "dOUTM"       "dHVAL"
##  [6] "dMBO"        "RES"         "dINC_SP"     "dOUTL"       "nKIDS"
## [11] "dOUTCC"      "dOUTHP"      "PHON"        "nDEP"        "YOB_missing"
```

```r
# GINI importance measures the average gain of purity by splits of a given variable. If the variable is
```

```r
### Xgboost variable importance

# Train xgb model
# NOTE: We do not do model selection here but use the optimal parameters we have previously found.
xgb.mlr <- makeLearner("classif.xgboost", predict.type = "prob",
                       par.vals = list("nrounds" = 100, "verbose" = 0, "max_depth" = 4, "eta" = 0.15,
                                       "gamma" = 0, "colsample_bytree" = 0.8, "min_child_weight" = 1, "
xgb <- mlr::train(xgb.mlr, task = task)

# the xgboost package has a function xgb.importance()
# Gain: Gini gain as in RF
# Cover: Relative number of observations split by each feature
xgb.importance(model = xgb$learner.model, feature_names = colnames(task$env$data))
```

```
##      Feature        Gain       Cover   Frequency
##  1:   dINC_A 0.207211682 0.175301694 0.185924370
##  2:      YOB 0.171360557 0.169353491 0.171218487
##  3:    dOUTM 0.102363387 0.101125751 0.125000000
##  4:    dHVAL 0.098315404 0.078135351 0.091386555
##  5:  dINC_SP 0.079882304 0.079829736 0.087184874
##  6:     dMBO 0.068673884 0.049320893 0.069327731
##  7:    dOUTL 0.051752277 0.065901141 0.060924370
##  8:   dOUTHP 0.041466883 0.064898485 0.038865546
##  9:    nKIDS 0.035801932 0.023600330 0.029411765
## 10:   dOUTCC 0.032629649 0.058909617 0.027310924
## 11:    RES.F 0.024059855 0.031225001 0.017857143
## 12: EMPS_A.V 0.012154270 0.016460505 0.008403361
## 13: EMPS_A.N 0.010139478 0.009316289 0.012605042
## 14:    RES.N 0.009254951 0.010831686 0.009453782
## 15: EMPS_A.B 0.008770268 0.008744423 0.008403361
## 16: EMPS_A.U 0.007624563 0.004083048 0.007352941
## 17: EMPS_A.R 0.006649347 0.010618491 0.009453782
## 18:    RES.O 0.006572882 0.008158519 0.010504202
## 19:    RES.P 0.006547482 0.005609072 0.006302521
## 20:     PHON 0.006505380 0.010665165 0.009453782
## 21: EMPS_A.E 0.005781913 0.014010524 0.006302521
## 22: EMPS_A.Z 0.004770981 0.001044117 0.004201681
## 23: EMPS_A.P 0.001710669 0.002856672 0.003151261
##      Feature        Gain       Cover   Frequency
```

```r
getFeatureImportance(xgb)
```

```
## FeatureImportance:
## Task: tr.dummy
##
## Learner: classif.xgboost
## Measure: NA
```

```
## Contrast: NA
## Aggregation: function (x)  x
## Replace: NA
## Number of Monte-Carlo iterations: NA
## Local: FALSE
##          YOB      nKIDS nDEP       PHON    dINC_SP    dINC_A     dHVAL
## 1 0.1713606 0.03580193    0 0.00650538 0.0798823 0.2072117 0.0983154
##          dMBO      dOUTM      dOUTL      dOUTHP     dOUTCC YOB_missing
## 1 0.06867388 0.1023634 0.05175228 0.04146688 0.03262965           0
##   EMPS_A.B    EMPS_A.E    EMPS_A.M EMPS_A.N    EMPS_A.P    EMPS_A.R
## 1        0 0.008770268 0.005781913        0 0.01013948 0.001710669
##      EMPS_A.T EMPS_A.U    EMPS_A.V   EMPS_A.W EMPS_A.Z       RES.F
## 1 0.006649347        0 0.007624563 0.01215427        0 0.004770981
##        RES.N       RES.O       RES.P       RES.U
## 1 0.02405985 0.009254951 0.006572882 0.006547482
```

```
featureImportance[["xgb"]] <- unlist(getFeatureImportance(xgb)$res)
#The measures are based on the number of times a variable is selected for splitting, weighted by the sq
# Plot relative variable importance scaled from 0 to 100
# When interpreting these, it's crucial to consider how the
# importance is calculated
maxMinStandardize <- function(x) ( (x - min(x)) / (max(x) - min(x)) ) * 100
importanceTable <- as.data.frame(sapply(featureImportance, maxMinStandardize, USE.NAMES = TRUE))
importanceTable[order(rowSums(importanceTable), decreasing = TRUE),]
```

```
##                     rf         xgb
## dINC_A      100.0000000 100.0000000
## YOB          94.0700626  82.6983086
## dOUTM        61.7362999  49.4003938
## dHVAL        54.0927256  47.4468443
## dMBO         44.7759874  33.1418979
## dINC_SP      37.8127244  38.5510620
## dOUTL        37.3443198  24.9755595
## nKIDS        24.8034011  17.2779507
## dOUTHP       20.2319259  20.0118461
## dOUTCC       21.0829111  15.7470120
## RES.N        13.9411415  11.6112444
## EMPS_A.P     15.1174196   4.8932945
## EMPS_A.W      9.3901096   5.8656299
## EMPS_A.E     10.6153772   4.2325162
## RES.O        10.1839365   4.4664235
## PHON         10.9065525   3.1394853
## EMPS_A.V     10.2030167   3.6796011
## RES.P         8.6864828   3.1720616
## RES.U         7.7418549   3.1598035
## EMPS_A.R      8.8818871   0.8255657
## RES.F         6.3169736   2.3024672
## EMPS_A.T      5.0898602   3.2089634
## EMPS_A.M      4.9980634   2.7903413
## EMPS_A.B      3.0044410   0.0000000
## nDEP          2.7413643   0.0000000
## EMPS_A.Z      2.3435944   0.0000000
## YOB_missing   2.2473959   0.0000000
## EMPS_A.N      0.3135638   0.0000000
## EMPS_A.U      0.0000000   0.0000000
```

Use the test set to compare the AUC performance for step function, RF and XGboost variable sets.

```
#Let's compare models after feature selection on the test set
AUC <- list()
selectedlr <- glm(glm_stepwise$formula,rbind(tr,val),family = "binomial")
ylr<- HMeasure(true.class=as.numeric(ts$BAD=="bad"), scores = predict(selectedlr, ts, type="response"))
AUC["lr"] <- ylr$metrics["AUC"]

#rf

selectedrf <- row.names(rf.importance[order(rf.importance[,"MeanDecreaseAccuracy"], decreasing = TRUE),]
rf <- randomForest(formula(paste("BAD ~ ", paste(selectedrf[1:10], collapse=" + "))), data = rbind(tr,va

yrf<- HMeasure(true.class=as.numeric(ts.dummy$BAD=="bad"), scores = (predict(rf, ts, type = "prob")[, 2]
AUC["rf"] <- yrf$metrics["AUC"]

#xgboost
selectedxgb <-unlist(getFeatureImportance(xgb)$res)
selectedxgb <- selectedxgb[order(selectedxgb, decreasing = TRUE)][1:10]

set <- loans[colnames(loans)%in%names(selectedxgb)]
set$BAD <- loans$BAD
set_dummy <- mlr::createDummyFeatures(set, target="BAD")

idx<- caret::createDataPartition(y = set_dummy$BAD, p = 0.7, list = FALSE)
trd<-set_dummy[idx, ] # training set
tsd<-  set_dummy[-idx, ]
task <- makeClassifTask(data =trd, target = "BAD", positive = "bad")
xgb <- mlr::train(xgb.mlr, task = task)
yxgb <- predict(xgb, newdata=tsd)

AUC["xgb"]  <- mlr::performance(yxgb,measures = mlr::auc)
AUC
```

```
## $lr
## [1] 0.6178529
##
## $rf
## [1] 0.6443299
##
## $xgb
## [1] 0.6464506
```