Random Forest Algorithm

An application to Insurance data

Random Forest introduction

- An extension of bagging (Boostrap Aggregating), that takes an average of each individual trees prediction,
- Difference is that before each split, the algorithm selects randomly a subset of features as "features candidate" for splitting instead of consideration all the features as candidate.
  ➔ **Consequences:** It de-correlates the trees and help improving the predictive performance.

- The algorithm to build a Random Forest (RF) can be summarized following the steps below:
  - **Step 1:** Create a bootstrapped data set from the original data set. It is done by randomly selecting sample from the original data with replacement. In general, 1/3 of the original data don't end up in this bootstrapped data set. We call it the "Out-of-Bag" data set (a.k.a "OOB"), referring to the entries that have not been selected to compose the original bootstrapped data set.

  - **Step 2:** create a decision tree using the bootstrapped data set, but only considering a random subset of variables at each step. It follows the same logic as a single tree, but this time we select a random number of variables.

  - **Step 3:** repeat steps 1 and 2 a certain number of time. It results in a wide variety of trees which makes the random forest more effective than an individual tree.

Random Forest – distRForest

```r
# Hyperparameter grid search
hyper_grid <- expand.grid(
  ntrees = seq(100,500, by = 100), # number of Trees

  mtry = seq(1, 5, by = 1), # the number of variables to randomly sample as candidates at each split.

  #min.node.size  = seq(3, 6, by = 2), #min.node.size in other packages:
  sample.size = c(.60, .80),
  OOB_RMSE    =  0
)

# total number of combinations
nrow(hyper_grid)
```

```
[1] 50
```

```r
library(distRforest)
set.seed(54321) # set the seed

# Record the computation time.
system.time(

  for(i in 1:nrow(hyper_grid)) {

rf_freq <- rforest( formula = as.formula(paste('cbind(exposure, clm.count) ~', paste(features, collapse = ' + '))),
  data = dta_trn,
  #weights = nclaims,
  method = 'poisson',
  ntrees = hyper_grid$ntrees[i], # T: specify the number of tree in the ensemble.
  ncand = hyper_grid$mtry[i], # m: the number of candidate variable to consider at each node to find the optimal split.
  subsample = hyper_grid$sample.size[i], # delta: each tree in the ensemble is built on randomly sampled data.

  # Parameters that control aspects of the rpart fit.
  control = rpart.control(cp = 0, # cp: complexity parameter.
                          #Any split that does not decrease the overall lack of fit by a factor of cp is not attempted.
                          minbucket = 0.01 * 0.75 * nrow(dta_trn), # kappa * delta in Table 1
                          xval = 0,
                          maxcompete = 0,
                          maxsurrogate = 0),
  red_mem = TRUE, # reduces the memory footprint of individual rpart trees
  track_oob = TRUE)

  hyper_grid$OOB_RMSE[i] <- mean(rf_freq$oob_error)
  }

)
```

Plethora of R packages are available to train a Random Forest, but they often use the mean squared error as the minimization criterion which is not optimal in our case where we focus on a claims count. This issue is solved by the {distRforest} package developed by Roel Henckaerts (https://www.github.com/henckr/distRforest) which is an extension of {rpart} that allows building random forest with both Gamma and Poisson deviance as loss functions.

We can set up a grid search and iterate a through it.

For heavier computation, PySpark or H2O framework are better solutions.

Random Forest – Hyperparameters & Grid Search

```r
# Hyperparameter grid search
hyper_grid <- expand.grid(
  ntrees = seq(100,500, by = 100), # number of Trees

  mtry = seq(1, 5, by = 1), # the number of variables to randomly sample as candidates at each split.

  #min.node.size  = seq(3, 6, by = 2), #min.node.size in other packages:
  sample.size = c(.60, .80),
  OOB_RMSE    =  0
)

# total number of combinations
nrow(hyper_grid)
```

Step 1: set up a grid-search.

```
[1] 50
```

```r
library(distRforest)
set.seed(54321) # set the seed

# Record the computation time.
system.time(

  for(i in 1:nrow(hyper_grid)) {

rf_freq <- rforest( formula = as.formula(paste('cbind(exposure, clm.count) ~', paste(features, collapse = ' + '))),
  data = dta_trn,
  #weights = nclaims,
  method = 'poisson',
  ntrees = hyper_grid$ntrees[i], # T: specify the number of tree in the ensemble.
  ncand = hyper_grid$mtry[i], # m: the number of candidate variable to consider at each node to find the optimal split.
  subsample = hyper_grid$sample.size[i], # delta: each tree in the ensemble is built on randomly sampled data.

  # Parameters that control aspects of the rpart fit.
  control = rpart.control(cp = 0, # cp: complexity parameter.
                          #Any split that does not decrease the overall lack of fit by a factor of cp is not attempted.
                          minbucket = 0.01 * 0.75 * nrow(dta_trn), # kappa * delta in Table 1
                          xval = 0,
                          maxcompete = 0,
                          maxsurrogate = 0),
  red_mem = TRUE, # reduces the memory footprint of individual rpart trees
  track_oob = TRUE)

  hyper_grid$OOB_RMSE[i] <- mean(rf_freq$oob_error)
  }

)
```

Step 2: iterate through it

a. Number of Trees, "ntree":
We want enough trees to stabilize the error rate but using too many trees is unnecessarily inefficient, especially when using large data sets.

b. mtry: represents the number of candidates variables for splitting. With bagging, the aggregation causes some tree correlation limiting the variance reduction effect.
Random Forest helps to reduce the tree correlation by performing a split-variable randomization: each time a split if performed, the search for the split is limited to a random subset (mtry) of the original features (p). Selecting only among a certain number of candidates implies some "randomness" in the process to decorrelate the trees. The direct effect is improving the predictive performance and reduces the variance.

For classification problem, {ranger} takes a default number equals to the square roots of the number of parameters. For regression, it is equal to the number of parameters divided by 3.We note that When mtry = p, the model is the same as bagging. When mtry = 1, the split variable is completely random.

c. Tree complexity parameter (Depth)
The following two parameters control the complexity and the size of the trees:

- **nodesize:** minimum number of samples within the terminal nodes.
- **maxnodes:** is the maximum number of terminal nodes allowed in a tree. Hence, if nodesize is small and maxnodes is large, the tree will grow large, i.e. more complex, which tends increase overfitting.

d. sampsize: the number of samples to train on. The sample size parameter determines how many observations are drawn for the training of each tree. More sample will increase the performance, but adding also more variance.

```r
rf_freq$trees[500]
```

```
[[1]]
n= 19596

node), split, n, deviance, yval
      * denotes terminal node

   1) root 19596 7928.172000 0.160280100
     2) yrs.lic=4,5,6,7,8+ 7428 2389.562000 0.113441100
       4) ncd.level=5,6 4806 1185.561000 0.080305520
         8) region.g1=R1,R7 401   46.911960 0.028924150 *
         9) region.g1=R0,R2,R3,R4,R5,R6,R8 4405 1127.564000 0.085287690
          18) yrs.lic=6,7,8+ 1970  379.549600 0.057791220
            36) drv.age.gr2=38-42,23-27,43-47,53-57,58-62 1151  140.470200 0.032630400
              72) drv.age.gr2=38-42,23-27,53-57 582   57.140620 0.027062770
               144) region.g1=R0,R2,R4,R8 251   10.658310 0.015911020 *
               145) region.g1=R3,R5,R6 331   44.577780 0.039739070 *
              73) drv.age.gr2=43-47,58-62 569   82.500800 0.040968340
               146) region.g1=R2,R3,R4,R5 326   20.087580 0.017892390 *
               147) region.g1=R0,R6,R8 243   55.929330 0.076057730 *
            37) drv.age.gr2=28-32,33-37,48-52,63+ 819  222.961800 0.095059570
              74) drv.age.gr1=44-50,34-38,61-69 500  108.664900 0.075607120 *
              75) drv.age.gr1=18-33,51-60,70+ 319  111.573400 0.127111700 *
          19) yrs.lic=4,5 2435  731.344700 0.107193100
            38) year=2013,2011,2012 2040  586.637500 0.100592300
              76) region.g1=R3,R5,R6,R8 1267  279.024900 0.071495280
               152) drv.age.gr2=38-42,23-27,28-32,33-37,48-52,58-62,63+ 851  152.970600 0.053401550
                 304) drv.age.gr1=18-33,51-60,61-69 363   35.276000 0.027222870 *
                 305) drv.age.gr1=44-50,34-38,39-43,70+ 488  112.239000 0.075019030 *
               153) drv.age.gr2=43-47,53-57 416  120.091400 0.109619500 *
              77) region.g1=R0,R2,R4 773  293.827000 0.146423800
```

Each of the different Trees can be printed and explored.

To estimate the accuracy of our model, we can run the model on the OOB samples and check if they are correctly labelled or not. Hence, we can measure the proportion of OOB samples that are correctly classified by the RF. The proportion of OOB samples incorrectly classified is called the "Out-Of-Bag Error". Once we checked the accuracy of our model, we can adjust our hypothesis by changing how many variables can be used at each step and then chose the one which is the most accurate.

```r
hyper_grid %>%
  dplyr::arrange(OOB_RMSE) %>%
  head(20)
```

Description: df [20 x 4]

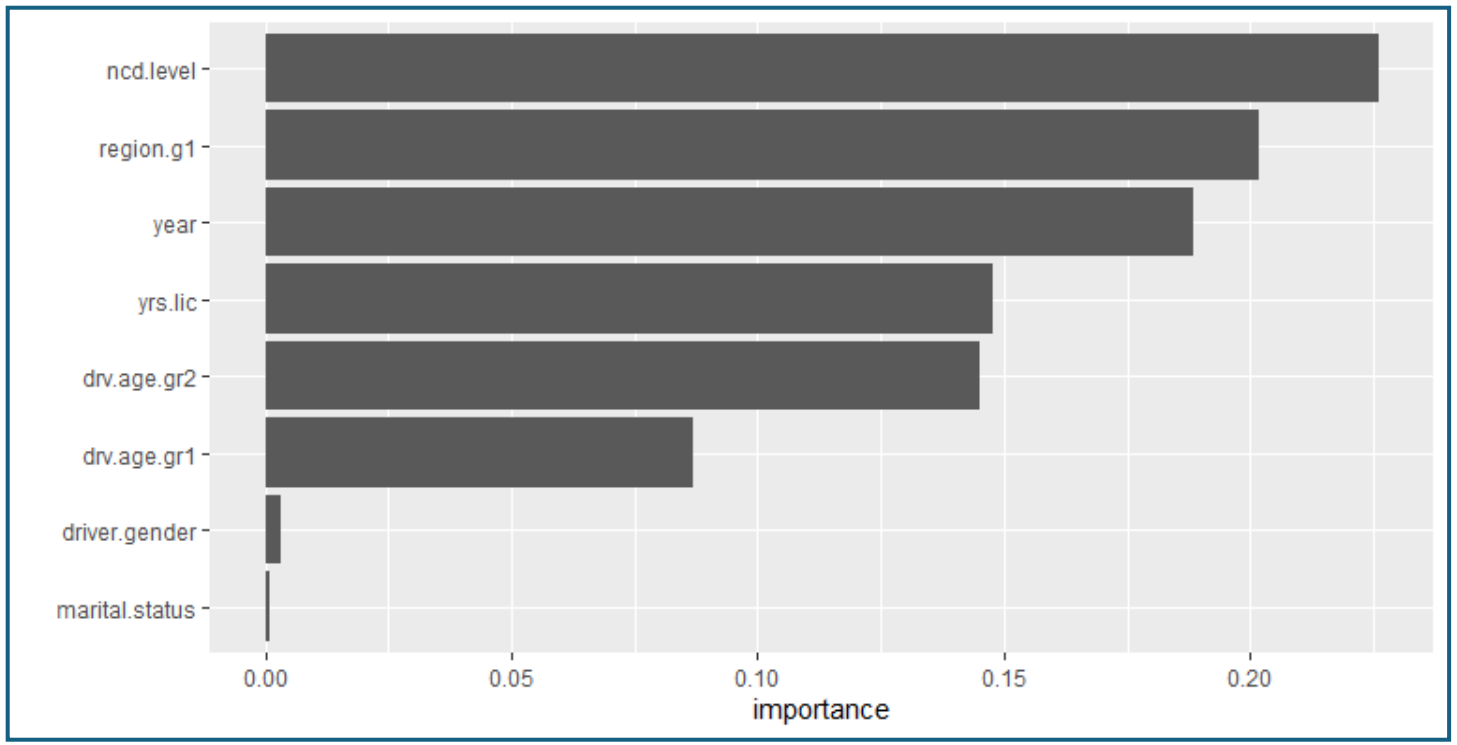
	ntrees <dbl>	mtry <dbl>	sample.size <dbl>	OOB_RMSE <dbl>
1	500	4	0.6	0.3849188
2	500	3	0.8	0.3849542
3	300	3	0.6	0.3849775
4	500	5	0.6	0.3849966
5	500	3	0.6	0.3850185
6	500	5	0.8	0.3850564
7	300	4	0.8	0.3850621
8	500	4	0.8	0.3850691
9	300	5	0.6	0.3850754
10	300	4	0.6	0.3850833

1-10 of 20 rows

Previous 1 2 Next

The run shows that the optimal combination of parameter is 500 trees and 4 candidates at each split.

Interpretation



- As for a single tree, we can plot the Variable Importance bar chart.
- “ncd.level” contributes the most to reduce the loss function.

Conclusion

- Powerful out-of-the-box algorithm that often has great predictive accuracy.
- Add the benefits of decision trees and bagging but greatly reduce instability and between-tree correlation.
- However, suffer from slow computational speed as your data sets get larger.

Sources

- *Predictive Modelling Applications in Actuarial Science, Vol.2* (E. Frees & al.)
- *Hands on ML with R (B. Broecke)* https://bradleyboehmke.github.io/HOML/