Variable importance and selection using random forests

Abderezzak Amimer , Madi Kassymbekov

17/04/2021

1. Presentation of the subject

This project is intended in proposing a tutorial for variable importance and variable selection in random forests using different R resources. The methods used in the following sections are purely designed for prediction setting and not an inference one. Before diving into the tutorial, each element discussed are briefly reviewed.

1.3 Random Forests

Random forests are in the family of ensemble methods. The principle is fairly simple. It combines many regression or classification trees using several bootstrap samples from the training data and randomly selecting a subset of the explanatory variables at each node in order to split the data. Amongst the random subset, one is selected using CART or Inference method to split. Additional details will be provided on this matter and these very powerful algorithms.

1.1 Variable importance

Variable importance can be defined as the contribution of each predictor to the model. It is usually represented as a ranking of the variables based on the effect they have on the generated model. Knowing which variables significantly impacts the quality of the predictions helps analysts weed out those that are not necessary and help , in certain cases, improve the quality of the predictions.

1.2 Variable selection

Variable selection consists of removing certain predictors from a model in order to improve the quality of a prediction. Following Kohavi and John (1997) and Guyon and Elisseff (2003), three groups of methods are distinguishable. First, “wrapper”, which look closely at the prediction performance of a model to make variable selection. Second, “filter”, the score of variable importance is not based on a model design and finally, “embedded” which combines both model estimation and variable selection.

Outline

This project is organized as follows. A description of important literature and the most recent algorithms used in R regarding variable importance and selection using trees. Next, an overview of the methods to be used in the tutorials. To continue, a section will be dedicated to review R resources that will be used on a simulated data set.This last part will be a demonstration of how to perform variable selection and variable importance using the references presented in the appropriate sections.

2. Literature review

2.1 Random forest

Random forests are based on decision trees with bootstrap aggregation called "bagging". It is a method used to generate multiple versions of a predictor to get an average predictor. Randomness is introduced in each predictor by making bootstrap replicates. As low bias is the feature of tree based models, high variance is one of its drawbacks dependent on the number of features that leads to larger trees and potential vulnerability to over fitting. However, compared to other tree models like CART and conditional inference tree, random forest models use bootstrap aggregation technique with random features to do prediction and afterwards uses weighted averaging which lowers the variance of predictions compared to classical decision tree models. The algorithm could be summarized by the following pseudo code Breiman (2001) :

1) For a training set of length N, sample N instances at random with replacement.
2) Grow a tree on the bootstrap training set using a specified number of random features.
3) Repeat step 1 and 2 for the set number of estimators.
4) Average predictions or take majority vote (depending on type of task).

2.2 Variable importance

Sensitivity to n and p All learners are sensitive to the nature of the data and it's quality. Two important components to consider when fitting a model is the number of observations n and the number of explanatory variables p. Genuer et al. (2010) ran an experiment using simulated data to understand how the variable importance index would vary when $n << p$. It was shown that as n decreases and becomes smaller compared to p, the variable importance index of significant variables would decrease and get closer to 0. The variability of the index also increases greatly and that is due to the intrinsic algorithm of random forests.

Sensitivity to mtry and ntree Mtry hyper parameter in the random forest algorithm is defined as the subset of covariates in the data that is used to perform the best split at different nodes. According to Genuer et al. (2010), the increase of the number of variables in the subset leads to an increase in the magnitude of variable importance index of useful predictors. On the other hand, ntree, an other hyper parameter of the random forest algorithm, is defined as the number of trees. It's increase leads to a decrease of the standard deviation of the variable importance index. It is important to note, that although the standard deviation decreases, using a large number of trees can prompt over fitting on the training data.

Sensitivity to correlated covariates Many studies were performed to evaluate the impact of having correlated covariates on the variable importance index. Archer and Kimes (2008) paper indicates that the increase in the number of correlated covariates makes it difficult to pick significant variables due to the dilution of the variable importance score which is confirmed by the experiments of Auret and Aldrich (2011).

2.3 Variable selection

In this project, the focus will be put on the "wrapper" methods which base the variable selection on a score that includes the prediction performance. Usually, the performance measure used is the mean squared error for regression tasks and the misclassification rate for classification tasks.

3. Brief review of methods

In the following section, a brief description of the methods used to calculate variable importance and to perform relevant variable selection will be presented.

3.1 Variable importance calculation methods

Classical approach

First we start with variable importance calculation as it is the main index used to perform variable selection using random forests.

The idea behind this measure is that a covariate X is important if the prediction error increases when the relationship between X and Y is modified. To illustrate how to calculate the variable importance once can simply follow these steps for the classical approach which is presented by Breiman (2001)

1) Fit a random forest on the training data.
2) Select an out of bag sample of observations that the model was not trained on (OOB) and calculate the mean squared error on each tree.
3) Select one covariate in the covariate space and permute the values of each instances in order to break the link between the response variable and the covariate selected.
4) Make new predictions using these permuted values and obtain a new mean squared error for OOB sample.
5) Perform the steps 3 and 4 for each covariate in the covariate space.
6) To calculate the variable importance, the sum of differences of the MSE between the permuted covariate and the non permuted covariate is taken and divided by the total number of OOB.

The following formula displays the results of this algorithm :

$$VIMP(X_j) = \frac{1}{B}\sum_{b=1}^{B}[MSE(\hat{f}_b, OOB_b^j) - MSE(\hat{f}_b, OOB_b)]$$

where j is the covariate for which the value are permuted. b is the b_{th} OOB sample. B is the number of out of bag samples which is equal to the number of trees. $\hat{f}_b$ is the prediction model for the b_{th} sample.

Node assignment variation approach

Isharwan H. (2007) proposed a different way of breaking the link between Y and X. Instead of permuting the values of X_j they proposed to assign the node at random instead of choosing the best variable to split on. The second option, consists of systematically choosing the opposite split each time a split is performed with that variable. This adds noise to the model and the error is then used to calculate the difference in error with the original model.

Holdout approach

Isharwan H. (2007) also proposed a second method to estimate the importance of a variable when it comes to predict the response by holding out a group of variables when growing trees and comparing the OOB errors on the different forests when the group is held out vs when it's not.

3.2 Variable selection methods

Variable selection in random forest is mainly based on the variable importance index. The following section explores and briefly describes the different ways to perform variable selection in order to get a better prediction performance.

Recursive feature elimination (RFE) and non recursive feature elimination (NRFE) RFE's objective is to find the smallest number of variables that help to get the best predictive model. It follows the following steps :

1. Train a random forest.
2. Compute the permutation importance measure.
3. Eliminate the less relevant variable(s).
4. Repeat steps 1 to 3 until no further variables remain.

This method is an improved version of the NRFE, which was proven to be less efficient on correlated data. The update of the ranking based on variable importance has proven to be more effective in presence of correlation withing the covariates. For that reason, NRFE will not be used in the following tutorials (Gregorutti et al (2016)).

Boruta This method was developed to find all relevant variables within a classification framework. Kursa and Rudnicki (2010) described the Boruta algorithm with the following steps:

The Boruta algorithm consists of following steps: 1. Extend the training data by adding copies of all variables (the data is always extended by at least 5 duplicate attributes, even if the number of attributes in the original set is lower than 5).

2. Shuffle the added attributes to remove their correlations with the response.The goal here is to break the link between Y and X.
3. Run a random forest classifier on the extended data set and gather the Z scores computed. The Z score is calculated based on the loss of accuracy in classification task.
4. Find the maximum Z score among duplicate attributes, and then assign a hit to every attribute that scored better than the maximum Z score.
5. For each attribute with undetermined importance perform a two-sided test of equality with the maximum z score.
6. Deem the attributes which have importance significantly lower than the maximum z score as 'unimportant' and permanently remove them from the data set.
7. Deem the attributes which have importance significantly higher than the maximum z score as 'important'.
8. Remove all duplicate attributes.
9. Repeat the procedure until the importance is assigned for all the attributes, or the algorithm has reached the previously set limit of the random forest runs.

Vsurf The Vsurf method is both useful in regression and classification tasks. It is also used for interpretation and prediction. The algorithm in question is performed in two different steps and it is described as follows (Genuer et al. (2015)).

Step 1 : Preliminary ranking and elimination.

a) This step consists of ranking the variables using the classical variable importance index. Genuer et al.(2015) suggests using a typical 50 trees to estimate the variable importance index.

b) Based on a certain threshold, eliminate all variables for which the variable importance index is below the threshold. The threshold in question is estimated by calculating the standard deviation of variable importance. Other strategies could be used to find the best threshold.

Step 2 : Variable selection.

a) For interpretation. The algorithm uses a very simple method to select the most performing model. For $k = 1$ to m, a Random Forest model is built using m important variables following the ranking previously estimated where 1 is the most important variable. Using a different number of runs, typically 25, select the variables that lead to the smallest OOB error.

b) For prediction. In order to improve the prediction performance, start with the ordered sequence of variables selected in the interpretation step. From there, build different random forests by following the order of variable importance from the previous step and use a step wise algorithm to eliminate variables that increase the OOB prediction error by a previously selected threshold.

Jefferie S. Evans et al. method for model selection (Parsimony) The purpose of the authors of the following method was to simplify as much as possible the complexity of the models. They used the permuted variable importance measure to do so. The method starts by fitting a random forest with all variables and estimates variable importance. The values are then ranked and standardized to a certain ratio. Then, it alliteratively subsets variables within a given ratio and fitting a model for each subset. The resulting model is compared to the original model which is kept constant. The performance is calculated on the OOB error which can include a penalty for the number of parameters in order to reduce the complexity of the model.

Altmann Altmann's method is based on an approach that keeps the values and the correlation structure of the covariates intact while computing variable importance under a null hypothesis of no association between predictor and the response variable. Instead of permuting the values of X_j, it permutes the values of the outcome and then multiple random forests are trained and new variable importance are calculated. If the variable importance in the original model without permutation is significantly different than 0, then the variable is kept. Otherwise, it is deemed as non-important and can be discarded from the model. This initial approach is called the permutation approach and is a basic method. Altmann's modifications are implemented with the distribution of the variable importance under the null hypothesis. In other words, in order to allow less training of random forests and to get a conclusive test, it is important to make the right assumptions on the variable importance distribution and the author of the described method allows a parametric approach to estimate the P-values by fitting a specific probability distribution such as a normal, a log-normal etc.

Recurrent relative variable importance This approach is used in case the number of unimportant variables in a data set is large. The selection method is simply based on a ratio of different random forest fits. In other words, first, several random forests are generated based on the data set and parameter values differing only in the seed of the random number generating process. Each random forest is used to compute the variable importance using the classical method and the values obtained are divided by the absolute minimum importance observed in each run. This outputs relative values and each variable having an importance greater or equal to a predetermined factor is kept.

Cross-validated permutation variable importance for variable selection This method randomly splits the data set into k sets of equal size. The method constructs k random forests, where the l-th forest is constructed based on observations that are not part of the l-th set. For each forest the fold-specific permutation variable importance measure is computed using all observations in the l-th data set: For each tree, the prediction error on the l-th data set is recorded. Then the same is done after permuting the values of each predictor variable.

The differences between the two prediction errors are then averaged over all trees. The cross-validated permutation variable importance is the average of all k-fold-specific permutation variable importances. For classification the mean decrease in accuracy over all classes is used and for regression the mean decrease in MSE (Janitza et al.(2015)).

4. Review of R resources

For each method listed above, the following section will provide a detailed overview of R resources allowing to perform the algorithms in question.

4.1 Variable importance packages

To calculate variable importance, two main packages are picked for comparison.

The first one is the **randomForest** initially developed in Fortran by Leo Breiman and Adele Cutler and later on ported to R by Andy Liaw and Matthew Wiener. Two functions are of interest in this resource for variable importance. First, the **importance()** function and second, **varImpPlot()**.

4.1.1 importance() function :

The purpose of this function is to calculate the variable importance values for each variable following the classical method presented earlier. The user can customize the importance ranking estimation by modifying the type of calculation made when performing permutation. The first type uses the decrease in accuracy and the second type uses the mean decrease in node impurity.

A very important detail for this function is to make sure that when fitting the randomForest model, specify in the argument of **randomForest()** , that importance=TRUE in order to take full advantage of the proposed arguments of the importance function itself.

Function call : importance(x, type=NULL, class=NULL, scale=TRUE)

Arguments

x : The fitted randomForest object.

type : Can take two values, 1 if the importance measure is calculated with the mean decrease in accuracy and 2 if it is calculated with the mean decrease in node impurity.

class : For classification tasks, this argument allows to calculate the variable importance for the specific class selected in the argument.

scale : When this argument is equal to true, the variable importance measure is divided by the standard deviation of the variable importance vector.

4.1.2 varImpPlot() function

Using the previous calculated importance values, this function provides a visual representation of the variable importance. With the default settings, it displays two plots based on the two types of importance measures (Type= 1 : mean decrease in accuracy or Type = 2 Node impurity).

Function call :

varImpPlot(x, sort=TRUE, n.var=min(30, nrow(x$importance)), type=NULL, class=NULL, scale=TRUE, main=deparse(substitute(x)))

Arguments

x : The fitted randomForest object.

sort : Sorting the values in decreasing order if True.

n.var : User defined number of variables to show.

type : Can take two values, 1 if the importance measure is calculated with the mean decrease in accuracy and 2 if it is calculated with the mean decrease in node impurity.

class : For classification tasks, this argument allows to calculate the variable importance for the specific class selected in the argument.

scale : When this argument is equal to true, the variable importance measure is divided by the standard deviation of the variable importance vector.

main : User defined plot title.

The second package allowing to calculate variable importance is titled **RandomForestSRC**. The authors justified this package by implementing survival forests and also improving computational performance on certain functions. The main contributors are Ishwaran and Kogalur. The relevant functions for computing importance are **vimp.rfsrc()** and **holdout.vimp()**.

4.1.3 vimp.rfsrc()

Function call :

vimp.rfsrc(object, xvar.names, m.target = NULL, importance = c("permute", "random", "anti"), block.size = 10, joint = FALSE, seed = NULL, do.trace = FALSE)

Arguments

object : The fitted randomForestSRC object.

xvar.names : User defined list of variable for which importance is calculated.

m.target : Label of the response variable for which the importance permutation will be calculated on. If nothing is specified, the code will select the default target variable fit on the random forest object.

importance : Three different types of importance values that can be calculated. If the user selects "permute", the classical approach to calculating variable importance will be used. This approach was explained in the previous sections. If the user specifies "random", to break the link between the target variable and the covariate in question, the nodes will be selected at random and , finally, the last option is to select "anti". For this last approach, the covariate is split by sending systematically the instances in the opposite node.

block.size : If block.size can be set from 1 to ntree. When set to ntree, comparison of OOB error will be made on the ensemble forest. On the other hand, if block.size is set to 1, tho OOB error is calculated by tree. Smaller values lead to better accuracy, but take more computation time. The user can make a compromise by selecting different block sizes.

joint : Is a useful argument when the user defines a set of covariates in xvar.names for which they would like to find the joint importance values when the links between the response and the covariates is perturbed as a group.

seed : Specifying the random number generator seed.

do.trace : To print the estimated time of completion.

4.1.4 holdout.vimp()

Function call :

holdout.vimp(formula, data, ntree = function(p, vtry){1000 * p / vtry}, nsplit = 10, ntime = 50, sampsize = function(x){x * .632}, samptype = "swor", block.size = 10, vtry = 1)

formula : Model formula. The same type of formula to use when the user fits a randomforestsrc().

data : A data frame object containing the response and the covariates.

ntree : Number of trees for growing the forest.

nsplit : Number of splits per node, the default value is 10.

ntime : A constraint of ensemble calculations to a grid of ntime value.

sampsize : Size of subsampled data, it could be a specific number or a function of x.

samptype : Type of bootstrap used. "swor" which means sampling without replacement and "swr" means sampling with replacement.

vtry : number of variables to be held-out when growing a tree, typically 1.

block.size : If block.size can be set from 1 to ntree. When set to ntree, comparison of OOB error will be made on the ensemble forest. On the other hand, if block.size is set to 1, though OOB error is calculated by tree. Smaller values lead to better accuracy, but require more computation time. The user can make a compromise by selecting different block sizes.

Summary

The two previous functions allow to perform calculations to display and indicate which covariates are the most useful in predicting the response variable. The **importance** function in the **randomForest** offers a simple way to get the details and it's usage is very straightforward. Thanks to the function **varImpPlot()** the user can easily display the result of their findings. On the other hand, the **vimp.rfsrc()** function from the **randomforestSRC** package offers much more flexibility and user parametrization. One of the most interesting features is the possibility to select the type of importance measure to compute and the number of trees used to obtain the OOB error comparison. Also, allowing for such customization in the function helps the user to alter the computation speed. For example, the analyst desiring to find out the importance of a group of variables could use the *joint* and specify those variable in the *x.var* argument to accelerate the process. Such flexibility is not offered in the **importance()** function. The last function covered for variable importance is the **holdout.vimp()** from the same package which offers important flexibility as well. The

calculation methods are very different from one another and results for the latter function require a large number of trees in order to lower the variance of the importance estimates.

The following table displays the summary of each function presented above.

Function name	**main purpose**	**approach**	**Package**
importance()	variable importance estimate	Breaking the link between X_j and y by permutation of x_j values for each observation in the data set	randomForest
varImpPlot()	Visualization	2 graphs based on the type of error	randomForest
vimp.rfsrc()	variable importance estimate	Three different approaches to estimate variable importance. Classical permutation of X_j, random node assignment and opposite split	randomForestSRC
holdout.vimp()	variable importance estimate	Holding out certain variables from the growing process and comparing the predictions to the ones made without holding them out	randomForestSRC

4.2 Variable selection packages

For variable selection the following four main R packages will be considered.

The first one is the **Boruta** package developed by Kursa and Rudnicki (2010). The main function for variable selection here is **Boruta** which applies the Boruta algorithm mentioned previously for variable selection.

4.2.1 Boruta() function :

Function call :

Boruta(x,y, pValue = 0.01, mcAdj = TRUE, maxRuns = 100, doTrace = 0, holdHistory = TRUE, getImp = getImpRfZ)

x :

y :

pValue :

mcAdj :

maxRuns :

doTrace :

holdHistory :

getImp :

formula :

data :

4.2.2 VSURF() function :

Function call :

VSURF(x, y, ntree = 2000, mtry = max(floor(ncol(x)/3), 1), nfor.thres = 50, nmin = 1, nfor.interp = 25, nsd = 1, nfor.pred = 25, nmj = 1, RFimplem = "randomForest", parallel = FALSE, ncores = detectCores()-1, clusterType = "PSOCK", verbose = TRUE)

x or formula :

y :

mtry :

nfor.thres :

nmin :

nfor.interp :

nsd :

nfor.pred :

nmj :

RFimplem :

parallel :

ncores :

clusterType :

verbose :

data :

na.action :

4.2.3 PIMP() function ;

Function call :

PIMP(X, y, rForest, S = 100, parallel = FALSE, ncores=0, seed = 123)

X :

y :

rForest :

S :

parallel :

ncores :

seed :

4.2.4 CVPVI() function :

Function call :

CVPVI(X, y, k = 2, mtry= if (!is.null(y) && !is.factor(y)) max(floor(ncol(X)/3), 1) else floor(sqrt(ncol(X))), ntree = 500, nPerm = 1, parallel = FALSE, ncores = 0, seed = 123)

X :

y :

k :

mtry :

ntree :

nPerm :

parallel :

ncores :

seed :

4.2.5 varSelRF() function :

Function call :

varSelRF(xdata, Class, c.sd = 1, mtryFactor = 1, ntree = 5000, ntreeIterat = 2000, vars.drop.num = NULL, vars.drop.frac = 0.2, whole.range = TRUE, recompute.var.imp = FALSE, verbose = FALSE, returnFirstForest = TRUE, fitted.rf = NULL, keep.forest = FALSE)

xdata :

Class :

c.sd :

mtryFactor :

ntree :

ntreeIterat :

vars.drop.num :

vars.drop.frac :

whole.range :

recompute.var.imp :

verbose :

returnFirstForest :

fitted.rf :

keep.forest :

Summary

The following table displays the summary of each function presented above.

Function name	main purpose	approach	Package