How to seep-up VSURF?

Robin Genuer\textsuperscript{a}, Jean-Michel Poggi\textsuperscript{b}, Christine Tuleau-Malot\textsuperscript{c}

\textsuperscript{a}Université de Bordeaux, ISPED, Inserm U1219, Inria
\textsuperscript{b}Université Paris Descartes et Laboratoire de mathématiques d’Orsay
\textsuperscript{c}Laboratoire Jean-Alexandre Dieudonné, Université de Nice

July 12
UseR 2019, Toulouse
Random Forests

- introduced by Breiman (2001)
- very efficient algorithm of statistical learning, for both classification and regression problems
Random Forests

- introduced by Breiman (2001)
- very efficient algorithm of statistical learning, for both classification and regression problems

\[ \mathcal{L}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\} \quad \text{i.i.d. r.v. with the same distribution as } (X, Y). \]

\[ X = (X^1, \ldots, X^p) \in \mathbb{R}^p \quad \text{(input variables)} \]

\[ Y \in \mathcal{Y} \quad \text{(output variable)} \]

- \( \mathcal{Y} = \mathbb{R} \) : regression
- \( \mathcal{Y} = \{1, \ldots, L\} \) : classification

**Goal** : build a predictor \( \hat{h} : \mathbb{R}^p \to \mathcal{Y} \) and select the most relevant variables
**Definition : Random Forests (Breiman 2001)**

\[
\left\{ \hat{h}(., \Theta_\ell), 1 \leq \ell \leq q \right\} \text{ randomized tree-predictor collection,}
\]
\[
(\Theta_\ell)_{1 \leq \ell \leq q} \text{ i.i.d. r.v. independent with } \mathcal{L}_n
\]

RF predictor \( \hat{h}_{RF} \) obtained by **aggregating** the collection of trees

**Agregation :**

- \( \hat{h}_{RF}(x) = \frac{1}{q} \sum_{\ell=1}^{q} \hat{h}(x, \Theta_\ell) \) regression

- \( \hat{h}_{RF}(x) = \arg\max_{1 \leq c \leq L} \sum_{\ell=1}^{q} \mathbf{1}_{\hat{h}(x, \Theta_\ell) = c} \) classification
Tree: piece-wise constant predictor from a recursive binary partitioning of $\mathbb{R}^p$

Splits are parallel to axes

Typically, at each node, the "best" split is seek (heterogeneity criterion based on the Y’s)

Example: CART (maximal tree building + pruning) Breiman et.al. (1984)
Random Forests-Random Inputs (Breiman 2001)

\[ \hat{h}(., \Theta_1, \Theta'_1) \quad \hat{h}(., \Theta_{\ell}, \Theta'_{\ell}) \quad \hat{h}(., \Theta_q, \Theta'_q) \]

Randomness + aggregation \Rightarrow increase of efficiency
**Definition : RI-tree**

An RI-tree is a variant of CART which selects at random, at each node, \( mtry \) variables, and splits only using the selected variables. An RI-tree is also not pruned.

\( mtry \) is THE method parameter and is same for all nodes of all trees in the forest:

- from \( mtry = 1 \) (splitting variable choice completely random)
- to \( mtry = p \) (*Bagging* Breiman 1996)
Main R packages


Other implementations in R:
- `ranger` Wright, Ziegler (since 2015): ”high-dimension”
- `Rborist` Seligman (since 2015): ”big data”

Two main parameters considered here:
- `ntree/num.trees/nTree` : number of trees in the forest (default = 500)
- `mtry/mtry/predFixed` : number of variables randomly selected at each node (default = $\sqrt{p}$ in classification)
Variable importance


**OOB** = Out Of Bag (≈ "Out Of Bootstrap")

**Definition: Variable importance (VI)**

Let $j \in \{1, \ldots, p\}$. For each OOB sample, randomly permute the $j$-th variable values of the data

$\text{VI}(X^j) = \text{mean increase of a tree error after permutation}$

*The more the error increases, the more important is the variable.*
1 Introduction
   - Definition
   - Examples

2 Variable selection
   - Procedure
   - Simulation study
   - Applications
Genuer, Poggi, Tuleau (2010)

We distinguish two different objectives:

1. to select all important variables, even with high redundancy, for interpretation purpose

2. to find a sufficient parsimonious set of important variables for prediction

Our aim is to build an automatic procedure, which fulfills these two objectives.

One earlier work must be cited: Díaz-Uriarte, Alvarez de Andrés (2006).
A simulated dataset \((L = 2, n = 100, p = 200, 6 \text{ true var.})\)
A simulated dataset \((L = 2, n = 100, p = 200, 6 \text{ true var.})\)

50 RF with VI + 31 \times 25 RF + 5 \times 25 RF
**VSURF() with randomForest**

```r
library(VSURF)
data(toys)
toysDef <- VSURF(toys$x, toys$y)
plot(toysDef)
summary(toysDef)
```

##
## VSURF computation time: 1.1 mins
##
## VSURF selected:
## 31 variables at thresholding step (in 45 secs)
## 5 variables at interpretation step (in 21.4 secs)
## 4 variables at prediction step (in 1.5 secs)
VSURF() with ranger

toysRanger <- VSURF(toys$x, toys$y, RFimplem = "ranger")
summary(toysRanger)

##
## VSURF computation time: 1 mins
##
## VSURF selected:
## 25 variables at thresholding step (in 44.5 secs)
## 5 variables at interpretation step (in 15.1 secs)
## 4 variables at prediction step (in 1.6 secs)

(Rborist implementation in progress...)
microbenchmark (25 runs) only on RF runs
High-dimensional \( (L = 2, n = 100, p=2000, 66 \text{ true var.}) \)
Big data ($L = 2$, $n=10000$, $p=10$, 6 true var.)
### VSURF on vac18 data \((L = 4, n = 42, p = 1000)\)

**randomForest**

```r
## VSURF computation time: 1.5 mins
## VSURF selected:
## 97 variables at thresholding step (in 45.7 secs)
## 23 variables at interpretation step (in 35.9 secs)
## 13 variables at prediction step (in 6.2 secs)
## VSURF ran in parallel on a FORK cluster and used 3 cores
```

**ranger**

```r
## VSURF computation time: 2.8 mins
## VSURF selected:
## 97 variables at thresholding step (in 51.7 secs)
## 23 variables at interpretation step (in 1.7 mins)
## 13 variables at prediction step (in 16.1 secs)
## VSURF ran in parallel on a FORK cluster and used 3 cores
```
VSURF on spam data \((L = 2, n = 2300, p = 57)\)

**randomForest**

```r
## VSURF computation time: 11.9 mins
## VSURF selected:
## 97 variables at thresholding step (in 1.2 mins)
## 23 variables at interpretation step (in 2.4 mins)
## 13 variables at prediction step (in 8.3 mins)
## VSURF ran in parallel on a FORK cluster and used 32 cores
```

**ranger**

```r
## VSURF computation time: 25.5 mins
## VSURF selected:
## 97 variables at thresholding step (in 40.9 secs)
## 23 variables at interpretation step (in 2.9 mins)
## 13 variables at prediction step (in 22 mins)
## VSURF ran in parallel on a FORK cluster and used 32 cores
```
Concluding Remarks

- Speed-up VSURF by changing RF implementation: not so simple!
  - ranger improve RF run time, but not always with VI
  - Rborist improve in the big data case, but do not compute VI natively

- ranger and Rborist allow parallel computing (randomForest not)

- Development version include RFimplem parameter:
  
  ```r
  remotes::installgithub("robingenuer/VSURF")
  ```


