Random Uniform Forests : an overview*

Saïp Ciss†

September 22, 2014

Abstract

Random Uniform Forests are a variant of Breiman's *Random Forests* (tm) (Breiman, 2001) and *Extremely randomized trees* (Geurts et al., 2006). Random Uniform Forests are designed for classification, regression and unsupervised learning. They are an *ensemble model* that builds many unpruned and randomized binary decision trees. Unlike Random Forests, they use *random cut-points* to grow each tree. Unlike Extremely randomized trees, they use *bootstrap* and subsampling, since *Out-of-bag (OOB)* modeling plays a key role. Unlike both algorithms, for each node *sampling with replacement* is done to select features. The main purposes of the algorithm are to get low correlated trees, to allow a deep analysis of variable importance and to be natively incremental. *Random uniform decision trees* are the core of the model. We provide an R package (randomUniformForest) that implements almost all functions of the algorithm, including Breiman's bounds and, in this overview, we present main theoretical arguments of the model and fully comprehensive examples with R code.

Keywords : Random Uniform Forests, Random Forests, statistical learning, machine learning, ensemble learning, classification, regression, R package.

*First version, September, 19, 2014. Last update, November, 05, 2014.

†Modal'X. Université Paris Ouest Nanterre La Défense. saip.ciss@wanadoo.fr

Contents

1 Introduction

Random Uniform Forests belong to the family of ensemble models that build many base learners then combine them to achieve tasks like classification or regression. They are first designed to be less computationally expensive than Breiman's Random Forests while keeping all properties of the latter. they also follow the idea of Extremely Randomized Trees (Extra-Trees) but, here, we do not want to loose the OOB (Out-of-bag) evaluation.

Implementations of ensemble learning are widespread, using different manners; however we will be focused on Random Forests and Extra-Trees since they provide efficient algorithms that can be used in R (R core Team, 2014) for real life problems and are close to Random Uniform Forests for their decision rule. Based learners are decision trees, like the CART (Breiman et al., 1984) paradigm, but in Random Uniform Forests they are unpruned and not deterministic.

In Random Uniform Forests, we seek strong randomization and some kind of global optimization. Hence, we want to achieve low correlation between trees (or trees residuals) leaving average variance of trees, eventually, increase but not to much. We are not concerned by bias, since we assume that if trees are enough randomized, ensemble model should have low bias. If not, Random Uniform Forests implement post-processing functions designed to reduce bias. So, conceptually, we build the model using the next steps.

i) For classification, we first draw, for each tree and with replacement, n observations among the n of the training sample (the *bootstrap*). For regression, we use *subsampling* (drawing, without replacement, m points out-of n, $m < n$).

ii) Each tree is grown by sampling uniformly, *with replacement*, a set of variables at each node. It means that, if the dimension of the problem is d, one can choose $\lceil \beta d \rceil$ variables, $\beta \geq 1/d$, at each node to grow the tree.

iii) Each cut-point is generated randomly, *according to the continuous Uniform distribution* on the support of each candidate variable or, more computationally efficient, between two random points of each candidate variable. Hence, we do not use any local optimization to find the best cut-point. Note that each one follows exactly the continuous Uniform distribution and, so, is not drawn among points in the node.

iv) Optimal random node is, then, selected by maximizing *Information Gain* (classification) or minimizing *'L2' (or 'L1') distance* (regression).

Each of the four elements above has a difference with either, Random Forests or Extra-Trees. Main arguments reside in drawing (with replacement) variables, using the continuous Uniform distribution and choosing the optimization criterion. One can note that at the algorithmic level, some others arguments are used (e.g. for categorical variables or for removing useless variables). In both Extra-Trees and Random Forests, there is no replacement when selecting variables. Extra-Trees use random cut-point, which is drawn uniformly between the minimal and the maximal value of each candidate variable for the current node. Like Random Forests, Random Uniform Forests use bootstrap for classification, but in the case of regression, subsampling is usually preferred, giving better results than bootstrap. All three algorithms use a different criterion to find optimal cut-point and variable, for each node, between candidates.

In the next section we focus on theoretical arguments of *random uniform decision trees*, the base learners.

In section 3, we provide the details of Random Uniform Forests.
In section 4, we provide some useful extensions of the model.
In section 5, we give a few fully comprehensive and reproducible examples with real life data sets.

2 Random uniform decision trees

Random uniform decision trees are unpruned and randomized binary decision trees. They use the continuous Uniform distribution to be built (that gives them their name) and, unlike CART, do not seek optimality. They, first, used to be the core of Random Uniform Forests and just growing one random uniform decision tree is useless (in comparison to CART). But understanding Random Uniform Forests needs first to know mechanisms of the former. We suppose than the reader knows what a decision tree is; we are just concerned on how it is grown in this specific case. When growing a (random uniform) decision tree we need, essentially, to know three aspects :
- how to grow and modify a node (region),
- when to stop growing tree,
- how to define and build the decision rule.

To better clarify these steps, we first give a quick definition of trees we propose.

`Definition.` *A random uniform decision tree is a binary decision tree in which nodes are built using random cut-points. For each step of the recursive partitioning,* $\lceil \beta d \rceil$ *variables,* $\beta \geq 1/d$*, are drawn with replacement. Then for each candidate variable, a cut-point* α *is drawn using the continuous Uniform distribution on the support of each candidate or between two random points of the latter. Optimal random node is the one that maximizes information gain (in classification) or that minimizes a* $\boldsymbol{L}^2$ *distance (or another one) in regression. The recursive partitioning is pursued unless a stopping rule is matched. The decision rule is then applied and exists only for terminal nodes.*

2.1 Regions

Random uniform decision trees are close to other types of decision trees and a node has always two or zero children nodes. We can now get the details of the algorithm in a few lines. We first have to define what a region is and for that, we call $\mathcal{P}$, a partition of the data. Following Devroye et al. (1996), we propose this definition.

`Definition.` *A is a region of the partition* $\mathcal{P}$ *if, for any* $B \in \mathcal{P}, A \cap B = \emptyset$ *or* $A \subseteq B$*.*

Hence, we suppose that we have $D_n = \{(X_i, Y_i), 1 \leq i \leq n\}$, corresponding to the observations and responses of the training sample, where (X, Y) is a $\mathbb{R}^d \times \mathcal{Y}$-valued random

pair. A is an *optimal region* of the random uniform decision tree if :

$$for\ any\ A \in \mathcal{P}, \left\{ X_i^{(j^*)} \leq \alpha_{j^*} | D_n \right\},\ 1 \leq j \leq d, 1 \leq i \leq n,$$
$$for\ any\ A^C \in \mathcal{P}, \left\{ X_i^{(j^*)} > \alpha_{j^*} | D_n \right\},\ 1 \leq j \leq d,\ 1 \leq i \leq n,$$

where, for classification :
$\alpha_j \sim \mathcal{U}\left(\texttt{min}(X^{(j)}|D_n), \texttt{max}(X^{(j)}|D_n)\right)$ and $j^* = \underset{j \in \{1,\ldots,d\}}{\texttt{arg max}}\ \texttt{IG}(j, D_n)$,
and for regression :
$\alpha_j \sim \mathcal{U}\left(\texttt{min}(X^{(j)}|D_n), \texttt{max}(X^{(j)}|D_n)\right)$ and $j^* = \underset{j \in \{1,\ldots,d\}}{\texttt{arg min}}\ \texttt{L}_2(j, D_n)$.

2.2 Stopping rules

Once A and A^C, its complementary region, are found, we repeat the recursive partitioning for the two regions (drawing random variables and cut-points and choosing the optimal region) until we met some conditions, which are :
- the minimal number of observations is reached,
- for one region, all the observations have the same label (or value),
- for one region, all the observations are the same,
- for one region, there is no more features to select,
- $\texttt{IG}(j, D_n)$ (or, in regression, $\texttt{L}_2(j, D_n)$) reached a threshold (usually 0).

2.3 Optimization criterion

Now, we just need to define the `IG` function, which is the *information gain*, and a *decision rule* for the tree. For classification, let us suppose that $Y \in \{0, 1\}$. We have :

$$\texttt{IG}(j, D_n) = \texttt{H}(Y|D_n) - \left[\texttt{H}\left(\left(Y|X^{(j)} \leq \alpha_j\right)|D_n\right) + \texttt{H}\left(\left(Y|X^{(j)} > \alpha_j\right)|D_n\right)\right],$$

where H is the Shannon entropy (note that we use it with the natural logarithm), and

$$\texttt{H}(Y|D_n) = -\sum_{c=0}^{1} \left\{ \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}_{\{Y_i = c\}}\ \texttt{log}\left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{I}_{\{Y_i = c\}} \right) \right\},$$

with $0\ \texttt{log}\ 0 = 0$, so that $\texttt{H}(Y) \geq 0$.

Let $n' = \sum_{i=1}^{n} \mathbf{I}_{\left\{X_i^{(j)} \leq \alpha_j\right\}}$, then

$$\texttt{H}\left(\left(Y|X^{(j)} \leq \alpha_j\right)|D_n\right) = -\frac{n'}{n} \sum_{c=0}^{1} \left\{ \frac{1}{n'} \sum_{i=1}^{n} \mathbf{I}_{\{Y_i = c\}} \mathbf{I}_{\left\{X_i^{(j)} \leq \alpha_j\right\}}\ \texttt{log}\left(\frac{1}{n'} \sum_{i=1}^{n} \mathbf{I}_{\{Y_i = c\}} \mathbf{I}_{\left\{X_i^{(j)} \leq \alpha_j\right\}} \right) \right\},$$

and

$$\texttt{H}\left(\left(Y|X^{(j)} > \alpha_j\right)|D_n\right) =$$
$$-\frac{n - n'}{n} \sum_{c=0}^{1} \left\{ \frac{1}{n - n'} \sum_{i=1}^{n} \mathbf{I}_{\{Y_i = c\}} \mathbf{I}_{\left\{X_i^{(j)} > \alpha_j\right\}}\ \texttt{log}\left(\frac{1}{n - n'} \sum_{i=1}^{n} \mathbf{I}_{\{Y_i = c\}} \mathbf{I}_{\left\{X_i^{(j)} > \alpha_j\right\}} \right) \right\}.$$

For regression, we have

$$\mathtt{L}_2(j, D_n) = \sum_{i=1}^{n} \left(Y_i \mathbf{I}_{\left\{X_i^{(j)} \leq \alpha_j\right\}} - \hat{Y}_A \mathbf{I}_{\left\{X_i^{(j)} \leq \alpha_j\right\}} \right)^2 + \sum_{i=1}^{n} \left(Y_i \mathbf{I}_{\left\{X_i^{(j)} > \alpha_j\right\}} - \hat{Y}_{A^C} \mathbf{I}_{\left\{X_i^{(j)} > \alpha_j\right\}} \right)^2,$$

with

$$\hat{Y}_A = \frac{1}{n'} \sum_{i=1}^{n} Y_i \mathbf{I}_{\left\{X_i^{(j)} \leq \alpha_j\right\}} \quad and \quad \hat{Y}_{A^C} = \frac{1}{n - n'} \sum_{i=1}^{n} Y_i \mathbf{I}_{\left\{X_i^{(j)} > \alpha_j\right\}}.$$

2.4 Decision rule

Each time a stopping criterion is met, we define the decision rule, $g_{\mathcal{P}}$, for an optimal (and terminal) node A. We have for classification :

$$g_{\mathcal{P}}(x, A, D_n) = g_{\mathcal{P}}(x) = \begin{cases} 1, \ if \ \sum_{i=1}^{n} \mathbf{I}_{\{X_i \in A, Y_i = 1\}} > \sum_{i=1}^{n} \mathbf{I}_{\{X_i \in A, Y_i = 0\}}, \ x \in A \\ \\ 0, \ otherwise. \end{cases}$$

And for regression :

$$g_{\mathcal{P}}(x, A, D_n) = g_{\mathcal{P}}(x) = \frac{1}{\sum_{i=1}^{n} \mathbf{I}_{\{X_i \in A\}}} \sum_{i=1}^{n} Y_i \mathbf{I}_{\{X_i \in A\}}, \ x \in A.$$

2.5 Algorithm

We can summarize the different tasks mentioned above by the following algorithm :

1- draw with replacement (or subsample, m out-of n, in case of regression) n observations of D_n.
a) draw with replacement $\lceil \beta d \rceil$ variables,
b) for each of the $\lceil \beta d \rceil$ variables, draw α using the continuous Uniform distribution on the support of each candidate variable or between two random points,
2- for the $\lceil \beta d \rceil$ variables, choose the pair (j^*, α^*) that maximizes $\mathtt{IG}(j, D_n)$ for classification, and for regression the one that minimizes $\mathtt{L}_2(j, D_n)$,
3- (j^*, α^*) is the pair that defines the regions A and A^C,
4- If a stopping rule is met, stop the partitioning and build the decision rule $g_{\mathcal{P}}$,
5- If not, pursue step 1 to 5 for A and A^C.

In a random uniform decision tree, partitioning leads to a large and deep tree (if balanced, depth is $\mathtt{log}(n)/\mathtt{log}(2)$) due to the randomness introduced. Since no pruning is done, terminal nodes tend to have a very few values leading to a high variance of the tree. But bias remains low and an interesting property is that more randomness does not hurt bias. Hence, in a random uniform décision tree prediction error is not an achievement, but high variance is. To avoid it increase too much, perturbations on the training sample sent to the optimization criterion are the main argument. The second one is that no local optimization is done since we want to lower the correlation between trees : following

Breiman's ideas, as variance for a single tree is high and hard to reduce, one can let it get high (introducing more diversity) and let ensemble do the reduction using the Law of Large Numbers, if trees are independent. In practice, they are not and one can use correlation to measure level of dependence.

Comparing to CART, one single random uniform decision tree will have an higher variance, but average variance of trees will be lower, in most cases, than CART variance. Then next step is to know how averaging will affect prediction error.

3 Random Uniform Forests

As Random Forests do, we use ensemble of random uniform decision trees to build a Random Uniform Forest. Algorithm is straightforward but one can note that many improvements come from the algorithmic level where a lot of randomness is essential to get improvements over a simple averaging of trees. Another point of view is that Random Uniform Forests use the Bayesian paradigm : *the forest classifier is the result of almost all parameters for a fixed data set.*

3.1 Algorithm

Once the structure of a random uniform decision tree is known, algorithm needs a few lines (but much more in practice) :
1- For each tree, from $b = 1$ to B, grow a random uniform decision tree and build its decision rule
2- For the B trees, apply the rule $\bar{g}_{\mathcal{P}}^{(B)}$ (see section 3.3).

3.2 Key concepts

Before going further, one can note the difference of concept between Random Forests and Random Uniform Forests :
in Random Forests, a tree is grown by choosing, at each step of the recursive partitioning, the optimal region among a few locally optimal, but globally randomized, regions. In Random Uniform Forests, the optimal region is chosen among many, possibly overlapping, random regions. In Extremely Randomized Trees, the optimal region is chosen among a few non-overlapping random regions. In all cases, guarantees are due to the Law of Large Numbers that generates convergence and needs trees to be theoretically independent.

Why not simply use Extra-Trees which are the fastest and have similar performance than Random Forests (and sometimes better, in case of regression)? The first reason is that they lack of OOB evaluation, since they need all training examples for each tree grown. OOB informations are much more important than just being an equivalent of cross-validation. The second reason is linked with optimization criterion which is more complex in Extra-Trees (in both classification and regression). We are looking for (very) simple optimizations in Random Uniform Forests in order to preserve diversity. The third one is that we think that the dimension of the problem matters as the randomization is increased and, especially in the regression case, dimension is one key to the problem.

The main difference with Random Forests appears in *average trees correlation* which is usually higher in the latter (at least for regression) while the *average variance of trees* is smaller. The motivation of pursuing low correlation appears directly in the Law of Large Numbers and theoretical properties of Random Forests, and Random Uniform Forests, that we want to be the closest to the practice. In other words, Random Uniform Forests are an application of Breiman's ideas using (strong) randomization to the point of view of observations rather than to the dimension whose we need, here, more for optimization.

3.3 Decision rule

Let us write $g_{\mathcal{P}}(X) \stackrel{def}{=} g_{\mathcal{P}}(X, \theta)$, where θ is the parameter that translates the randomness introduced in the tree. For the b-th tree, $1 \leq b \leq B$, we have $g_{\mathcal{P}}^{(b)}(X) \stackrel{def}{=} g_{\mathcal{P}}(X, \theta_b)$. For an ensemble of trees, the decision rule, $\bar{g}_{\mathcal{P}}^{(B)}$, is easy to write. We have for binary classification :

$$\bar{g}_{\mathcal{P}}^{(B)}(x) = \begin{cases} 1, \ if \ \sum_{b=1}^{B} \mathbf{I}_{\{g_{\mathcal{P}}^{(b)}(x)=1\}} > \sum_{b=1}^{B} \mathbf{I}_{\{g_{\mathcal{P}}^{(b)}(x)=0\}} \\ \\ 0, \ otherwise. \end{cases}$$

And for regression :

$$\bar{g}_{\mathcal{P}}^{(B)}(x) = \frac{1}{B} \sum_{b=1}^{B} g_{\mathcal{P}}^{(b)}(x).$$

The decision rule is simple and one can ask how to find interesting properties and how to explain good results of ensemble models. Trees in Random (Uniform) Forests are designed to be weakly dependent in order to apply most of their properties. Breiman provides main theoretical properties of Random Forests and Random Uniform Forests simply inherit from them.

3.4 Convergence of prediction error

Let us first look classification, considering mg, the margin function that measures the difference (in frequency) between points correctly classified and points misclassified. We have

$$mg(X, Y) = \left(\frac{1}{B} \sum_{b=1}^{B} \mathbf{I}_{\{g_{\mathcal{P}}^{(b)}(X)=Y\}} \right) - \left(\frac{1}{B} \sum_{b=1}^{B} \mathbf{I}_{\{g_{\mathcal{P}}^{(b)}(X)\neq Y\}} \right).$$

Let us call (following Breiman's notation), PE^* the prediction error. We have

$$PE^* = \mathbf{P}_{\mathbf{X},\mathbf{Y}} \left(mg(X, Y) < 0 \right),$$

and by the Law of Large Numbers, when $B \to \infty$,

$$PE^* \stackrel{p.s.}{\to} PE = \mathbf{P}_{\mathbf{X},\mathbf{Y}} \{ \mathbf{P}_{\theta}(g_{\mathcal{P}}(X, \theta) = Y) - \mathbf{P}_{\theta}(g_{\mathcal{P}}(X, \theta) \neq Y) < \ 0 \}.$$

(Breiman, 2001, theorem 1.2).

For regression, we get same results. We have

$$PE^*(forest) \overset{def}{=} PE^*(\bar{g}_{\mathcal{P}}^{(B)}(X)) = \mathbf{E}_{\mathbf{X},\mathbf{Y}} \left(Y - \frac{1}{B} \sum_{b=1}^{B} g_{\mathcal{P}}^{(b)}(X, \theta_b) \right)^2,$$

and when $B \to \infty$,

$$PE^*(\bar{g}_{\mathcal{P}}^{(B)}(X)) \overset{p.s.}{\to} PE(\bar{g}_{\mathcal{P}}^{(B)}(X)) = \mathbf{E}_{\mathbf{X},\mathbf{Y}} \left(Y - \mathbf{E}_{\theta} g_{\mathcal{P}}(X, \theta) \right)^2.$$

(Breiman, 2001, theorem 11.1).

- As a consequence, Random Uniform Forests do not overfit if trees are independent (in practice, a little dependent). One can note that low correlation between trees is easy to achieve in (binary) classification (usually around 0.1) but much harder in regression (usually around 0.3 or more).
- A second consequence is that one does not need to grow a lot of trees. Convergence toward the true prediction error of the model will quickly happen as one add trees to the forest.
- The third one is that in classification one needs first to lower bias when in regression one needs first to reduce correlation.

But, the main application of convergence is the ability to just focus on ways to reduce prediction error without the need to further work on consistency. Note that, for the latter, first results for regression have been obtained recently (Scornet et al., 2014) and will probably lead to interesting developments.

3.5 The OOB classifier

The Out-of-bag (OOB) informations are the observations that do not participate to the trees building. For each tree, due to bootstrap or sub-sampling, some observations are not chosen and are stored in order to build the OOB classifier whose decision rule is $\bar{g}_{\mathcal{P},oob}^{(B)}(X)$. The OOB classifier exists only for the training sample and use B' trees, $B' < B$, with n observations. Note that the B' trees are not necessary the same for each observation that needs to be evaluated. We have for an observation x and for only D_n

$$\bar{g}_{\mathcal{P},oob}^{(B)}(x) = \begin{cases} 1, \ if \ \sum_{b=1}^{B} \mathbf{I}_{\left\{g_{\mathcal{P}}^{(b)}(x)=1\right\}} \mathbf{I}_{\left\{b \in G^-(x,B)\right\}} > \sum_{b=1}^{B} \mathbf{I}_{\left\{g_{\mathcal{P}}^{(b)}(x)=0\right\}} \mathbf{I}_{\left\{b \in G^-(x,B)\right\}} \\ \\ 0, \ otherwise. \end{cases}$$

And, for regression :

$$\bar{g}_{\mathcal{P},oob}^{(B)}(x) = \frac{1}{\sum_{b=1}^{B} \mathbf{I}_{\left\{b \in G^-(x,B)\right\}}} \sum_{b=1}^{B} g_{\mathcal{P}}^{(b)}(x) \mathbf{I}_{\left\{b \in G^-(x,B)\right\}},$$

where $G^-(x, B)$ is the set of trees, among the B, which have never classified x.

The OOB classifier gives an estimate of prediction error and lead to many improvements, like post-processing, in order to control prediction error or for others purposes. One of the most important is a way to prevent overfitting, using the OOB classifier in conjunction with Breiman's bounds.

3.6 Bounds

As correlation decreases, average variance of trees increases and it begins harder to reduce prediction error. If one wants to control both correlation and variance, one key is to observe and monitor Breiman's bounds. These are the bounds of Random (Uniform) Forests that ensure that prediction error (under the *i.i.d.* assumption) will not increase beyond a limit. Most applications of Breiman's bounds are linked with the OOB classifier (that inherit to the bounds) and show if more work (also depending on n, but not on feature engineering) is needed to improve the modeling or if there is no more room for the algorithm. At first, bounds involve classification and we have two. The first bound is, by the Bienaymé-Tchebychev inequality,

$$PE^* \leq \frac{\mathbf{Var}(mr)}{s^2},$$

where

$$mr(X,Y) = \mathbf{P}_\theta(g_{\mathcal{P}}(X,\theta) = Y) - \mathbf{P}_\theta(g_{\mathcal{P}}(X,\theta) \neq Y),$$

is the limit of mg and $s, s > 0$, the *strength* (or margin), is

$$s = \mathbf{E}_{\mathbf{X},\mathbf{Y}}\{mr(X,Y)\}.$$

This first bound states that prediction error would *always* be under a limit which is explicit but unknown (unless one evaluates it using OOB informations). It is the *upper bound of prediction error* but it can be loose (but useful in case of problems with imbalanced classes or more difficult ones).

The second bound is tighter. We have (Breiman, 2001, theorem 2.3)

$$PE^* \leq \frac{\bar{\rho}(1-s^2)}{s^2},$$

where

$$\bar{\rho} = \frac{\mathbf{E}_{\theta,\theta'}[\rho(\theta,\theta')\sqrt{\mathbf{Var}(\theta)}\sqrt{\mathbf{Var}(\theta')}]}{\mathbf{E}_{\theta,\theta'}[\sqrt{\mathbf{Var}(\theta)}\sqrt{\mathbf{Var}(\theta')}]},$$

with $\rho(\theta,\theta')$ is the correlation of two trees of parameters θ and θ' and $\mathbf{Var}(\theta)$ is the variance of the tree, standing as the variance of the *raw strength*. Let us note the latter rmg. We have

$$rmg(\theta,X,Y) = \mathbf{I}_{\{g_{\mathcal{P}}(X,\theta)=Y\}} - \mathbf{I}_{\{g_{\mathcal{P}}(X,\theta)\neq Y\}},$$

and

$$\mathbf{Var}(\theta) \stackrel{def}{=} \mathbf{Var}_{\mathbf{X},\mathbf{Y}}(rmg(\theta,X,Y)).$$

One can note that $mr(X,Y) = \mathbf{E}_\theta(rmg(\theta,X,Y))$.

Here we are concerned by correlation (easy to reduce) and much more by strength, which is connected with bias. In Random Uniform Forests, we first expect to increase the strength by adding more trees. If we try more optimization, strength may also increase but correlation will increase too. Fortunately, we can lower correlation by using more randomization (up to a limit) without not affecting strength too much. we call Breiman's

second bound the *expected bound of prediction error* and one result we are looking for is that it must not be exceeded, thanks to the OOB error. One can note that the bound could not work in case of imbalanced classes or correlated covariates.

In regression, Breiman also provides a bound and the theoretical prediction error of the forest. The key differences with classification reside in the fact that :
1 - the bound is not an upper bound,
2 - It tells explicitly where prediction error can be pushed.
3 - It has stronger link with (average) variance of trees.

Suppose that, for all θ, $\mathbf{E}(Y) = \mathbf{E}_{\mathbf{X}}(g_{\mathcal{P}}(X, \theta))$. We have, Breiman (2001, theorem 11.2),

$$PE(forest) \leq \bar{\rho} PE(tree),$$

where

$$PE(tree) \stackrel{def}{=} PE(g_{\mathcal{P}}(X, \theta)) = \mathbf{E}_{\theta}\mathbf{E}_{\mathbf{X},\mathbf{Y}}(Y - g_{\mathcal{P}}(X, \theta))^2,$$

and $\bar{\rho}$ is the *average correlation between trees residuals*.
Moreover, we have the *estimator of the theoretical prediction error of the forest* given by

$$\widehat{PE^*}(forest) = \widehat{PE^*}(\bar{g}_{\mathcal{P}}^{(B)}(X)) = \hat{\bar{\rho}} \left(\frac{1}{B} \sum_{b=1}^{B} \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(Y_i - g_{\mathcal{P}}^{(b)}(X_i) \right)^2} \right)^2,$$

with

$$\hat{\bar{\rho}} = \frac{\sum_{1 \leq b < c \leq B} \hat{\rho}_{b,c}(\theta, \theta') \sqrt{\widehat{\mathbf{Var}_{\theta}}(Y - g_{\mathcal{P}}^{(b)}(X, \theta))} \sqrt{\widehat{\mathbf{Var}_{\theta'}}(Y - g_{\mathcal{P}}^{(c)}(X, \theta'))}}{\sum_{1 \leq b < c \leq B} \sqrt{\widehat{\mathbf{Var}_{\theta}}(Y - g_{\mathcal{P}}^{(b)}(X, \theta))} \sqrt{\widehat{\mathbf{Var}_{\theta'}}(Y - g_{\mathcal{P}}^{(c)}(X, \theta'))}}.$$

In regression we are concerned by both correlation and *average prediction error of trees* (and, then, by their variance). Here, each time one wants to lower variance, correlation increases unless one finds a way to do both reduction. Randomization is, yet, the key to reduce correlation. To avoid variance getting high, one strategy is to work more on the dimension, growing large and deep trees (also useful for bias reduction) and, in many cases, to use post-processing. In regression, the main problem is that randomization leads to a lot of combinations that affect variance but not enough reduce correlation. As a consequence, in many experiments, correlation can exceed, 0.3 or 0.4, without reducing variance.

We can note two points :
- Due to this correlation, not enough low, a part of the problem is linked with bootstrap which does not generate enough diversity. That's why it is not used in Random Uniform Forests for regression.
- Due to variance that may remain high, we allow to draw variables, at each node, with replacement in order to increase competition, and thus try to reduce average variance of trees at some cost (expected low) to the correlation.

The good news are that it seems there is enough room to reduce prediction error in Random (Uniform) Forests. One way to assess it is to monitor the theoretical prediction error of the forest using the OOB classifier. If it is lower than the OOB prediction error then improvements may be found at the cost of more computation. If not then one has to take care of possible overfitting.

3.7 Decomposition of prediction error and OOB bounds

We suppose here that the relation of Y and X is unknown. Only first and second order moments are expected (and *i.i.d.* assumption). Then, we can decompose prediction error in order to find where to look for the control of the latter. We have, for binary classification (and $Y \in \{0, 1\}$),

$$\mathbf{P}_{\mathbf{X},\mathbf{Y}}\left(\bar{g}_{\mathcal{P}}^{(B)}(X) \neq Y\right) = \mathbf{P}(Y = 1) + \mathbf{P}(\bar{g}_{\mathcal{P}}^{(B)}(X) = 1) - 2\mathbf{E}\left\{Y\bar{g}_{\mathcal{P}}^{(B)}(X)\right\}, \tag{1}$$

and for regression,

$$\mathbf{E}_{\mathbf{X},\mathbf{Y}}(Y - \bar{g}_{\mathcal{P}}^{(B)}(X))^2 = \mathbf{E}(Y^2) - 2\mathbf{E}_{\mathbf{X},\mathbf{Y}}\left\{Y\bar{g}_{\mathcal{P}}^{(B)}(X)\right\} + \left\{\mathbf{E}_{\mathbf{X}}\left(\bar{g}_{\mathcal{P}}^{(B)}(X)\right)\right\}^2 + \mathbf{Var}_{\mathbf{X}}\left(\bar{g}_{\mathcal{P}}^{(B)}(X)\right), \tag{2}$$

(Ciss, PhD thesis, 2014).

Replacing Breiman's bounds by their OOB counterparts give *OOB bounds*, whose we want to be upper bounds (letting us fully control algorithm and prediction error) of test errors.

A- In the case of classification,
let us, first, write the OOB empirical error and the test error :

$$\overline{PE}_{oob}^{*(B)} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{I}_{\left\{\bar{g}_{\mathcal{P},oob}^{(B)}(X_i)\neq Y_i\right\}},$$

$$\overline{PE}^{*} = \frac{1}{N-n}\sum_{i=n+1}^{N}\mathbf{I}_{\left\{\bar{g}_{\mathcal{P}}^{(B)}(X_i)\neq Y_i\right\}},$$

where $N - n$, $N > n$, is the size of the test sample.

i) Let us suppose that for any randomly chosen training and test samples, the first term of the relation (1) sees its empirical counterpart be approximatively equal in both samples.
ii) Consider the OOB classifier and suppose its correlation, ρ, with Y values of the training sample is approximatively equal with the one of the forest classifier with unknown Y values in the test sample.

If n, the size of the training sample, and B are large enough and if

$$\sqrt{\widehat{\mathbf{Var}_{\mathbf{X}}}\left(\bar{g}_{\mathcal{P}}^{(B)}(X)\right)} - \sqrt{\widehat{\mathbf{Var}_{\mathbf{X}}}\left(\bar{g}_{\mathcal{P},oob}^{(B)}(X)\right)}$$
$$> \frac{\left(1 - \frac{2}{n}\sum_{i=1}^{n}\mathbf{I}_{\{Y_i=1\}}\right)\left(\frac{1}{N-n}\sum_{i=n+1}^{N}\mathbf{I}_{\left\{\bar{g}_{\mathcal{P}}^{(B)}(X_i)=1\right\}} - \frac{1}{n}\sum_{i=1}^{n}\mathbf{I}_{\left\{\bar{g}_{\mathcal{P},oob}^{(B)}(X_i)=1\right\}}\right)}{2\hat{\rho}\sqrt{\widehat{\mathbf{Var}}(Y|D_n)}}$$

then, for any test sample of size, large enough, $N - n$,

$$\overline{PE}^{*} \leq \overline{PE}_{oob}^{*(B)}.$$

(Ciss, 2014, Proposition 6).

The claim means that we can found non-asymptotically conditions where OOB error will be an upper bound of test error. It suffices that we have a large enough train sample (in practice test sample will also be large for real life problems) under the *i.i.d.* assumption. Then the point *i)* will hold. For the point *ii)*, one can note that the OOB classifier is a weaker classifier than the forest classifier (since it uses less trees) and is a part of the latter. Hence, its correlation with Y will usually be lower or close to the correlation of the forest classifier with Y in the test sample.

One can observe here that we bypass the *Bayes classifier*, working directly on the prediction error. We do not know if the forest classifier will be optimal but since it converges our main goal will be to check first if OOB error is below the Breiman's bound (or, in case of enough correlated covariates or imbalanced classes, below the upper bound of prediction error). In a second time we can apply the relation provided above to estimate if overfitting could be a problem. Note that the relation is also a way to look for more optimization. As it holds one can try to optimize the forest.

B- In the case of regression, we have

$$\overline{PE}^{*}(\bar{g}_{\mathcal{P},oob}^{(B)}(X,\theta)) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \bar{g}_{\mathcal{P},oob}^{(B)}(X_i)\right)^2,$$
$$\overline{PE}^{*}(\bar{g}_{\mathcal{P}}^{(B)}(X,\theta)) = \frac{1}{N-n}\sum_{i=n+1}^{N}\left(Y_i - \bar{g}_{\mathcal{P}}^{(B)}(X_i)\right)^2.$$

i) Let us suppose that for any randomly chosen training and test samples, enough large, the empirical counterpart of $\mathbf{E}_{\mathbf{X},\mathbf{Y}}\left\{Y\bar{g}_{\mathcal{P}}^{(B)}(X)\right\}$ *is approximatively equal in the training and test samples.*
ii) Suppose that we get the same result for $\mathbf{E}(Y^2)$*.*

$$if \quad \left|\frac{1}{N-n}\sum_{i=n+1}^{N}\bar{g}_{\mathcal{P}}^{(B)}(X_i)\right| < \left|\frac{1}{n}\sum_{i=1}^{n}\bar{g}_{\mathcal{P},oob}^{(B)}(X_i)\right| \quad and \quad \widehat{\mathbf{Var}}_{\mathbf{X}}(\bar{g}_{\mathcal{P}}^{(B)}(X)) < \widehat{\mathbf{Var}}_{\mathbf{X}}(\bar{g}_{\mathcal{P},oob}^{(B)}(X)),$$

then for any test sample of size, large enough, $N-n$

$$\overline{PE}^{*}(\bar{g}_{\mathcal{P}}^{(B)}(X)) \leq \overline{PE}^{*}(\bar{g}_{\mathcal{P},oob}^{(B)}(X)).$$

In regression we have to focus on mean and variance of the forest (and OOB) classifier, with respect to each sample, and thus there is more work to do (in classification the sample has more effect on the prediction error, in regression it is the model). Here, point *i)* is not so obvious and the hope is, again, on convergence. In practice, one can for example use validation sample to estimate how it holds.

Again, we do not rely on the best possible classifier but have a powerful tool with the theoretical prediction error of the forest that will drive the modeling and that can be connected with the relation above. Note that theoretical prediction error can be misleading especially if correlation is too high and it does not prevent from overfitting.

To achieve first part of the presentation, we will omit some further steps, like *post-processing*, *random combinations of features*, or *extrapolation* and will drive on interpretability, which gives some important keys to the modeling.

3.8 Variable importance

As in Random Forests, we provide a measure for variable importance. In fact, Random Uniform Forests are built to get deeper in the assessment of variable importance.

A - First measure is the *global variable importance*, measured directly used the optimization criterion. if `VI` is the score of importance, we have for the j-th variable, $1 \leq j \leq d$, named j^* if it is optimal,

$$\texttt{VI}(j^*) = \sum_{b=1}^{B}\sum_{l=1}^{k_b}\texttt{IG}_{b,l}(j^*, D_n),$$

and, for regression

$$\texttt{VI}(j^*) = \sum_{b=1}^{B}\sum_{l=1}^{k_b}\texttt{L2}_{b,l}(j^*, D_n),$$

where k_b is the number of regions for the b-th tree, $1 \leq b \leq B$. We, then, measure *relative influence* by computing $\texttt{VI}(j^*)/\sum_{j=1}^{d}\texttt{VI}(j)$. If one variable is never optimal (for the criterion), then it will never be measured and its influence will be null. This global

variable importance is similar to the one (that uses the Gini criterion) in Random Forests. The problem is that it does not tell us anything about how one important variable affect the responses. If we want to do *feature selection*, we might want to know, for example, features that affect the most one class or features that lead to high values of Y, in case of regression. Or we might want to know interactions of variables. To do that and more, we provide many measures that, for the sake of brevity, we will just give a definition and enumerate.

B - First, we define the *local importance*.

`Definition.` *A predictor is locally important at the first order if, for a same observation, and for all trees, it is the one that has the highest frequency of occurrence in a terminal node.*

Let us note $\texttt{LVI}^{(b)}(j,i)$, the local importance score of the i-th observation and the j-th variable of X, for the b-th tree,
$\texttt{LVI}(j,i)$, the score for all trees,
$\texttt{LVI}(j,.)$, the score of the j-th variable of X for all observations,
$R(j,\alpha_j) = \{X|X^{(j)} \leq \alpha_j\}$, a candidate terminal region whose all observations are below α_j, for $X^{(j)}$,
$R^C(j,\alpha_j) = \{X|X^{(j)} > \alpha_j\}$, its complementary region.
For simplicity, we rely on classification and define $\texttt{LVI}^{(b)}(j,i)$ by

$$\texttt{LVI}^{(b)}(j,i)$$
$$= \mathbf{I}_{\{\texttt{IG}_b(j^*,D_n)\geq \texttt{IG}_b(j,D_n)\}} \left(\mathbf{I}_{\left\{g_{\mathcal{P}}^{(b)}(X_i,R(j^*,\alpha_{j^*}))=g_{\mathcal{P}}^{(b)}(X_i,R(j,\alpha_j))\right\}} + \mathbf{I}_{\left\{g_{\mathcal{P}}^{(b)}(X_i,R^C(j^*,\alpha_{j^*}))=g_{\mathcal{P}}^{(b)}(X_i,R^C(j,\alpha_j))\right\}} \right).$$

The above relation states that we count when a variable falls into a terminal node, which tree and which observation are concerned. Knowing that, we can count for all trees, the function $\texttt{LVI}(j,i)$ given by

$$\texttt{LVI}(j,i) = \sum_{b=1}^{B} \texttt{LVI}^{(b)}(j,i),$$

and for all trees and observations and for the j-th variable,

$$\texttt{LVI}(j,.) = \sum_{i=1}^{n} \texttt{LVI}(j,i).$$

Next step is to consider that the j-th variable can appear in second, third, or more, position when counting it, given the definition above. Due to the randomness in trees, one can not expect that a locally important variable will not let space to some others variables. For example, another variable may appear at the second position, very often and never in first position. Note, here, that position and order designate the same term. We, then, define the local importance score of position q, $1 \leq q \leq d$, $\texttt{SLI}_q(j,.)$ for all observations and for the j-th variable. We have

$$\texttt{SLI}_q(j,.) = \sum_{i=1}^{n} \texttt{LVI}_{(d-q+1)}(j,i),$$

where $\texttt{LVI}_{(d-q+1)}$ is the order statistics of the `LVI` function. Thus, at first position, we get the score for the most locally important variable, and so on.

C - At this point, we get the tools to derive some new informations about variable importance. First one is *partial importance.*

`Definition.` *A predictor is partially important at the first order if, for a same observation, and at all the orders, it is the one that has the highest frequency of occurrence in a terminal node.*

Its score is given by

$$\texttt{SLI}(j,.) = \sum_{q=1}^{d} \texttt{SLI}_q(j,.).$$

To understand partial importance one has to take account two points:
i) counting occurrences of a predictor is interesting but does just give frequency,
ii) for each observation met in a terminal node, we record the value estimated by the classifier. That gives us the intensity (or the label in case of classification).

Hence, we have for each observation, the score of all covariates and the classifier estimate. Then, partial importance gives, for each class or for some responses above (or below) a threshold, the predictors which are the most relevant. To be more clear, partial importance tells what are the variables that are explaining the variations of the response (regression) or one class rather than another (classification). This gives us a new information with regard to global variable importance, that just tells us what are the relevant variables.

D - But we may want to know how covariates rely to the problem when we consider them all. For example, some variables could have a low relative influence on the problem but a strong effect on a more relevant covariate. Or this covariate could have many interactions with others, leading the variable to have influence. So, we need a definition for *interactions.*

`Definition.` *A predictor interacts with another one if, for a same observation, and for all trees, both have respectively the first and the second highest frequency of occurrence in a terminal node.*

If `VII` is the score of interactions, we have for $X^{(j)}$ and $X^{(j')}$,

$$\texttt{VII}_{(1,2)}(j,j') = \frac{\texttt{SLI}_1(j,.) + \texttt{SLI}_2(j',.)}{n}.$$

Interactions act as a visualization tool in order to see the influence of all covariates in a unique manner. Hence, the relation above computes the last step of the process.
i) First, we compute and order $\texttt{SLI}_1(j,.)$ for all covariates.
ii) Then, we compute and order $\texttt{SLI}_2(j,.)$ for all covariates.

iii) The last step computes $\texttt{VII}_{(1,2)}(j, j')$ and we create a contingency table where all values of `VII` will stand. That gives us the *interactions visualization tool* of all covariates. Moreover, we can merge covariates that have weak influence to have a granular view.

Interactions are interesting since they give the big picture on how all variables rely each other with respect to the problem. They also lead to the measure of *variable importance based on interactions*. This measure of relative influence is another tool to explain the influence of covariates.

E - The last measures we provide are *partial dependencies*. There are tools that allow to measure how influence of each covariate (or a couple of covariates) is affecting the response values, knowing the values of all others. In other words, a partial dependence plot is the marginal effect of a covariate on the responses values. The idea of partial dependence came from Friedman (2002) who used it in *Gradient Boosting Machines (GBM)*; but it is implemented differently in Random Uniform Forests. Call $\texttt{pD}_q^{(j)}$ the partial dependence at position q for the j-th variable. We have

$$\texttt{pD}_q^{(j)}(Y, X) = \left\{ \left(\left(\bar{g}_{\mathcal{P}}^{(B)}(X_i, R(j, \alpha_j)), X_i^{(j)} \right) | \texttt{SLI}_q(j, i) > 0 \right), 1 \leq i \leq n \right\},$$

where $\bar{g}_{\mathcal{P}}^{(B)}(X_i, R(j, \alpha_j))$ is the forest classifier evaluating each observation, knowing that the j-th variable is locally important for the current evaluated observation. The local importance is, so, designated by the $\texttt{SLI}_q$ function for every point. What makes the difference with a simple look in terminal node to get a prediction is that the covariate must have some influence for each observation. Hence both predicted value and observation will stress the partial dependence plot up to some point where no values will be available to plot. To avoid getting a too weak relation between covariate and target, we define the partial dependence function at all orders. It is given by

$$\texttt{pD}^{(j)}(Y, X) = \left\{ \left(\left(\bar{g}_{\mathcal{P}}^{(B)}(X_i, R(j, \alpha_j)), X_i^{(j)} \right) \middle| \sum_{q=1}^{d} \texttt{SLI}_q(j, i) > 0 \right), 1 \leq i \leq n \right\}.$$

The function provides all the points needed in the range of the covariate and increases the randomness. This has the advantage of getting more consistent and interpretable results. If covariate and target have no relation, plotted points will have an Uniform distribution on the map. Otherwise, one will get the shape of their relation.

Partial dependencies can be extended to a couple of variables, getting a surface, or even on the whole dimension of the problem. But in this latter case, one will need a lot of points and visualization would fail. Hence, one usually works on the most important variables. One of the interesting point with partial dependencies is their ability to allow extrapolation. This latter is known to be a limitation of Random (Uniform) Forests since their estimator can not produce unseen values beyond the range of Y in the training sample. To allow extrapolation one just has to compute a parametric model of the tails in the partial dependence function. Usually these will be linear with the covariate and extrapolation happens by choosing the right threshold beyond which a linear model will be a good approximation. If many covariates have influence, one can eventually compute

one parametric model over a multi-dimensional partial dependence function. The main argument of Random Uniform Forests is that they allow to get more points for such objects like extrapolation. We think it might be the case for others ensemble models too.

We provide in the lines above many ways to assess variable importance and one can visualize most of them in just one screen to get the big picture. It leads to both feature selection and interpretation. This latter point is essential since to our point of view Random (Uniform) Forests are not a black box. We can then summarize main properties of variable importance in Random Uniform Forests.
i) All tools provided, except the global variable importance, work on both training and test sample.
ii) All tools are complementary. It means that when a feature is found to be important, an explanation can be found using each measure separately.
iii) The main purpose of variable importance is to assess what, when, where and how covariates have influence on the problem.
iv) In reference to Random Uniform Forests, we can define the importance of a variable with the following sentence : *importance = contribution + interactions, where contribution is the influence of a variable (relatively to the influence of all) on the prediction error and interactions is, at least, its influence on the others covariates.*

4 Some extensions

In this section we quickly present a few other developments of Random Uniform Forests. We begin by *prediction and confidence intervals*, then we briefly talk about *output perturbation sampling*, which lead to get some insights about Random Forests, and we end with *incremental learning*.

4.1 Prediction and confidence intervals

When going toward real life problems or when industrialization comes to be the standard case, the main problem is not to give accurate prediction on average, but to give strong guarantees that what is announced is what will be happen. First option uses the cross-validation or OOB classifier and errors. For regression, one will usually not want the prediction value, hard to exactly match, but the prediction value with a prediction interval and, for a parameter, a confidence interval. To be more clear if, for example, the problem involves some financial risk or cost, just rely on a prediction value could be dangerous. What is more interesting is, instead, to give some other informations to the man that pushes the button.

Random Uniform Forests provide a new approach to build some predictions and confidence intervals, especially designed for real life cases. We do not discuss to the motivations behind such cases, and give directly the process to build such intervals.

Let us note $\hat{q}_\alpha(g_\mathcal{P}(X_i,\theta))$, the α order empirical quantile of the $g_\mathcal{P}$ distribution for the i-th observation. We usually have this relation : *with a probability* $1-\alpha$*, and for all* $i \in [1,n]$*, the bootstrap prediction interval for every* Y_i *is like that*

$$Y_i \in \left[\hat{q}_{\alpha/2}(g_\mathcal{P}(X_i,\theta)), \hat{q}_{1-\alpha/2}(g_\mathcal{P}(X_i,\theta))\right].$$

This latter works well but is usually too large and sometimes difficult to use in practice.

As an alternative, we consider the following approach, for each Y_i.
i) We draw, with replacement, S_i estimates of $G(X_i,B) = \{g_\mathcal{P}^{(1)}(X_i), g_\mathcal{P}^{(2)}(X_i), ..., g_\mathcal{P}^{(B)}(X_i)\}$, where S is the number of unique values of $G(X_i,B)$. Then, we compute a first estimator of Y_i given by

$$\bar{g}_\mathcal{P}^{(S)}(X_i) = \frac{1}{S}\sum_{s=1}^{S} g_\mathcal{P}^{(s)}(X_i),$$

where the $g_\mathcal{P}^{(s)}(X_i)$ are randomly chosen.
ii) We repeat the operation K times and it leads to a K-sample $\{\bar{g}_\mathcal{P}^{(S_1)}(X_i), \bar{g}_\mathcal{P}^{(S_2)}(X_i), ..., \bar{g}_\mathcal{P}^{(S_K)}(X_i)\}$.
iii) We, then, consider $\hat{q}_\alpha(\bar{g}_\mathcal{P}^{(S)}(X_i,))$, the α order empirical quantile of the $\bar{g}_\mathcal{P}^{(S)}$ distribution for our K-sample. It is not directly used for Y_i, mainly because for some distribution like the exponential one, small or very frequent (and close) values are a hard problem when considering a prediction interval. We suppose, so, that in the neighborhood of a quantile, empirical distribution is Gaussian and, then, define the following prediction interval : *with an approached probability* $1-\alpha$*,*

$$Y_i \in \left[\hat{q}_{\alpha/2}(\bar{g}_\mathcal{P}^{(S)}(X_i)) + z_{\alpha/2}\sqrt{\frac{\widehat{\mathbf{Var}}_{\theta_\mathbf{S}}(g_\mathcal{P}(X_i,\theta_S))}{S}}, \hat{q}_{1-\alpha/2}(\bar{g}_\mathcal{P}^{(S)}(X_i)) + z_{1-\alpha/2}\sqrt{\frac{\widehat{\mathbf{Var}}_{\theta_\mathbf{S}}(g_\mathcal{P}(X_i,\theta_S))}{S}}\right],$$

where $z_{\alpha/2}$ is the $\alpha/2$ order quantile of the $\mathcal{N}(0,1)$ Gaussian distribution,
$\widehat{\mathbf{Var}}_{\theta_\mathbf{S}}(g_\mathcal{P}(X_i,\theta_S))$ is the empirical variance of the decision rule for which values are unique for X_i.

We, then, derive a confidence interval for a parameter, say for example $\bar{Y}$.
Let us have

$$\tilde{q}_{\alpha/2}(g_\mathcal{P}(X_i,\theta)) = \hat{q}_{\alpha/2}(\bar{g}_\mathcal{P}^{(S)}(X_i)) + z_{\alpha/2}\sqrt{\frac{\widehat{\mathbf{Var}}_{\theta_\mathbf{S}}(g_\mathcal{P}(X_i,\theta_S))}{S}},$$

then, with an approached probability $1-\alpha$,

$$\bar{Y} \in \left[\frac{1}{n}\sum_{i=1}^{n}\tilde{q}_{\alpha/2}(g_\mathcal{P}(X_i,\theta)), \frac{1}{n}\sum_{i=1}^{n}\tilde{q}_{1-\alpha/2}(g_\mathcal{P}(X_i,\theta))\right].$$

iv) The last step is to validate intervals using OOB informations or with a validation sample.

4.2 Output perturbation sampling

Output perturbation sampling is one of the most strange application of Random Uniform Forests and we first present it. Let's consider the algorithm and only the regression case : for each tree, subsample m out-of n observations of D_n and proceed to the growth.
Output perturbation sampling stands for the following procedure : for each tree, subsample m out-of n observations of D_n, *and replace all (or a part)* Y_i *values,* $1 \leq i \leq m$, *by random values drawn from a variable* Z, *where* $Z \sim \mathcal{N}(\bar{Y}, c\widehat{\mathbf{Var}}(Y))$, $c \geq 1$. Then, grow the tree.
When using output perturbation sampling, some results are empirically observed :
- overfitting hardly happens,
- correlation is strongly reduced (up to a factor 3), but variance increases too,
- prediction error slightly increases and, in some cases, is better.

The third point is the most surprising, since we found in many cases, if not all, that the prediction error was, in fact, close to the standard case after some (automatic) post-processing. It seems that the model just only needs the parameters. In others words, if one just grows many uncorrelated trees, convergence will happen and prediction error will remain low as long as one knows *only* an estimate of the parameters of the distribution of Y.

4.3 Incremental learning

Incremental learning is a way to do learning for streaming data. If data are very large or come by chunks, then incremental learning is one way to treat them whatever their size is. But, it will usually have a cost, at least a loss of accuracy. Incremental learning proceeds by chunks of data and regression case is usually less easy than classification, for achieving low prediction error (in contrast of a computation of all the data at once).
Incremental learning has two cases :
- the *i.i.d.* assumption holds for all chunks (or, at least, many),
- the (joint) distribution of (X, Y) is shifting (either the distribution itself or its parameters).

Random Uniform Forests are natively incremental, even for a shifting distribution, but some work is required. Output perturbation sampling is a part of the mix for not seeing erratic prediction error. The classification case is more simple and one can let the algorithm change itself how the classification has to work, i.e., let it change the threshold that is giving the result of the majority vote.

We consider the general case of a shifting distribution : the decision rule is the same for all cases, but one has to rely on the OOB classifier (and some others tools) to adapt the forest classifier. Let us call $\bar{g}^{(T)}_{\mathcal{P},inc}$ the incremental forest classifier. We simply have

$$\bar{g}^{(T)}_{\mathcal{P},inc} = \begin{cases} 1, \ if \ \sum_{t=1}^{T}\sum_{b=1}^{B_t} \mathbf{I}_{\left\{g^{(b)}_{\mathcal{P}_t}(x)=1\right\}} > \sum_{t=1}^{T}\sum_{b=1}^{B_t} \mathbf{I}_{\left\{g^{(b)}_{\mathcal{P}_t}(x)=0\right\}} \\ \\ 0, \ otherwise. \end{cases}$$

and for regression,

$$\bar{g}^{(T)}_{\mathcal{P},inc}(x) = \frac{1}{\sum_{t=1}^{T} B_t} \sum_{t=1}^{T} \sum_{b=1}^{B_t} g^{(b)}_{\mathcal{P}_t}(x),$$

where $\mathcal{P}_t$ is a partition of the data for the slice time t,
B_t, is the number of trees for the slice time t,
T, is the number of slices time.

Incremental learning is a subsampling process that we apply on both data and decision rule. Each tree sees only a part of the data and the forest itself sees a (bigger) part of the data. The main argument here is that cut-points are random, so see a part or whole data does not change many things. And even if it does, there is no other option. The problem relies more on the informations retained. Some informations that are important for a slice time, can be obsolete in the next slice time or worst, leading to confuse the forest. The only hope is, then, to adapt the classifier to the new situation without loosing all the informations memorized before. Since a sub-forest is a forest, with just less trees (like the OOB classifier is) we can adapt Random Uniform Forests by assessing $\bar{g}^{(T)}_{\mathcal{P},inc}$, $\bar{g}^{(T-1)}_{\mathcal{P},inc}$ and $\bar{g}^{(B_{T-1})}_{\mathcal{P}}$. One can note that trees are never modified; but they can be recalled, duplicated, removed and stored, or added.

5 Examples

We provide in this section some full examples on how tasks are computed using Random Uniform Forests.

Usage in R

The reference manual of the *randomUniformForest* package provides all the details of how one can use and manage options in the algorithm. In the lines below, we refer on the options used.

To install the package :

```
install.packages("randomUniformForest")
```

To load the package :

```
library(randomUniformForest)
```

Methods

With the *randomUniformForest* package, we use, for comparison,
- package *randomForest* (Liaw, Wiener, 2002), which implements the original Breiman's procedure,
- package *extraTrees* (Simm, de Abril, 2013), which freely implements the Extremely Randomized Trees algorithm,

- package *e1071* (Meyer et al.), which implements *SVM (Support Vector Machines)* (Vapnik, Cortes, 1995),
- package *gbm* (Ridgeway et al.), which implements *Gradient Boosting Machines* (Friedman, 2001, 2002).

Note that others accurate models are available on R, like *BayesTree*, *ipred*, *glmnet*, *rpart*, *kernlab*, *klar*,... View more at *Rdocumentation* or at *CRAN Task View: Machine Learning and Statistical Learning*.

We use for most of comparisons a 10-fold cross validation. For each data set, we use the same seed (equal to 2014), calling before each procedure the R code *set.seed(2014)*, in order to let everyone reproduce the results, since all data are available on the UCI repository : http://archive.ics.uci.edu/ml/index.html

We provide R code, as often as possible, to show how operations are done in a simple way.

Some algorithms like gbm, extraTrees and randomUniformForest are such randomized that using the same seed will produce different results on the same data. To get convergence one has to use enough trees (≥ 200).

5.1 Classification

We evaluate 4 datasets available on the UCI repository using each time one topic to illustrate in a different manner Random Uniform Forests.

5.1.1 Iris data (assessment of variable importance)

Iris data is one of the most famous dataset in machine learning. We report the description: *This famous (Fisher's or Anderson's) iris dataset gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.*

```
# load the algorithm and data
library(randomUniformForest)
data(iris)
XY = iris

# column of the classes
classColumn = 5

# separate data from labels
Y = extractYFromData(XY, whichColForY = classColumn)$Y
X = extractYFromData(XY, whichColForY = classColumn)$X

# run model: default options
ruf = randomUniformForest(X, Y = as.factor(Y))
```

```
# display results...
ruf

# ...gives standard output (below) of a Random Uniform Forest

Call:
randomUniformForest.default(X = X, Y = as.factor(Y))

Type of random uniform forest: Classification

                           paramsObject
ntree                               100
mtry                                  5
nodesize                              1
maxnodes                            Inf
replace                            TRUE
bagging                           FALSE
depth                               Inf
depthcontrol                      FALSE
OOB                                TRUE
importance                         TRUE
subsamplerate                         1
classwt                           FALSE
classcutoff                       FALSE
oversampling                      FALSE
outputperturbationsampling        FALSE
targetclass                          -1
rebalancedsampling                FALSE
randomcombination                 FALSE
randomfeature                     FALSE
categorical variables             FALSE
featureselectionrule            entropy

Out-of-bag (OOB) evaluation
OOB estimate of error rate: 4%
OOB error rate bound (with 1% deviation): 4.88%

OOB confusion matrix:
            Reference
Prediction   setosa versicolor virginica class.error
  setosa         50          0         0        0.00
  versicolor      0         47         3        0.06
  virginica       0          3        47        0.06

OOB geometric mean: 0.9596
OOB geometric mean of the precision: 0.9596
```

```
Breiman's bounds
Prediction error (expected to be lower than): 0.37%
Upper bound of prediction error: 3.84%
Average correlation between trees: 0.0176
Strength (margin): 0.9096
Standard deviation of strength: 0.1782
```

1 - One can note that some options are defined as in the RandomForest package, meaning the same arguments (*ntree* for the number of trees, *mtry* for the number of variables tried at each node, or *nodesize* for the minimal number of observations per node).
2 - The next lines give the OOB evaluation, with OOB errors, confusion matrix and some measures to assess the evaluation.
3- Last group are the Breiman's bounds and their details. One can note here that OOB error is not bounded by any Breiman's bound. It means that overfitting is likely to happen and one should have more data before concluding on prediction error, since pairwise observations are needed to estimate margin and correlation, leading to less cases than attended.

We, then, focus more on how to explain species with the covariates. First step gives the global variable importance, which is embedded (by default) in the model.

```
# call the summary() function gives some details about the forest and
# global variable importance
summary(ruf)
```

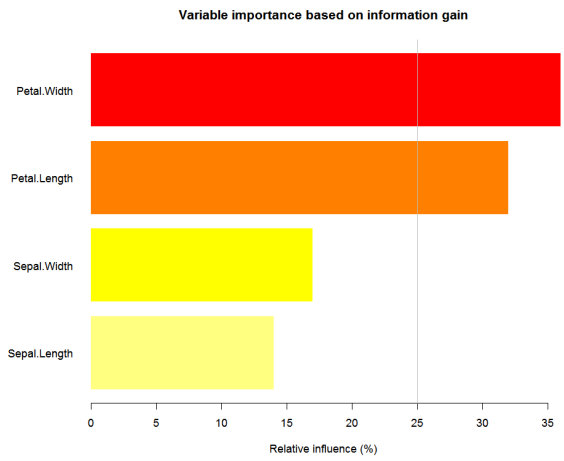

Figure 1: Global Variable importance for the iris dataset

The figure shows relative influence of all variables. the vertical line (in grey) shows where the influence would be if all variables had the same one. But, knowing what variables are important do not tell us what explains one specie or another. We, then, get in the details.

```
# assess the details of variable importance
# we set 'maxInteractions' to its highest level,
# i.e., the dimension of the problem
imp.ruf = importance(ruf, Xtest = X, maxInteractions = 4)

# then, see the big picture (please use vertical tiling in the R menu
# to see all plots)
plot(imp.ruf, Xtest = X)
```

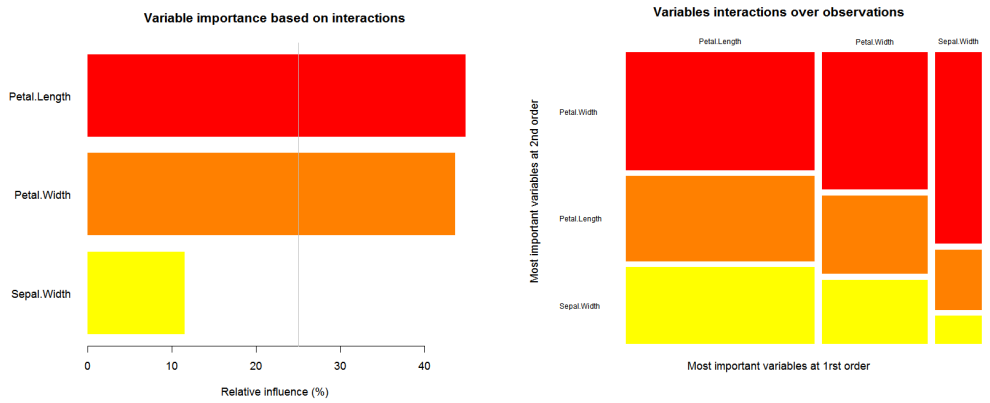

Figure 2: Variable importance based on interactions, and interactions, for the iris dataset

Visualizing the interactions is the first manner to assess variable importance by relying more on both model and data. Here we count variables that reside on terminal nodes in first or second position. We ask more work to the model since most of each tree is necessary before counting a variable, but only half (the terminal nodes) of the tree can be used. Randomness creates constraint, as the interactions creates another one, by requiring that the same variable occurs again in second order (position) to be enough important. Counting occurrences relies more on data since all potential predictions (in all trees) are involved. This scheme gives the *Variables interactions over observations*. One can see that *Petal.width* and *Petal.Length* lead to most of interactions while *Sepal.Length* have no interaction with any other variable.

The *mosaic plot* orders variables by their importance. First order means that the first variable in column is the most important, following by the second variable and so on. Second order means that there is another (unknown) variable more important than the first one in row. This unknown variable can, however, appears in the second order if, for example, it is important in most cases. To be simple, first variables in first and in second order are usually the most important, when taking account interactions, but it is not always the case and not a rule. Here, we do not have many variables. *Petal.Length* has strong influence and consequently, even if we are looking the top variables in the second order view, we get it but not at the most important variable. That shows us that *Petal.Length* has an influence in probably most of the classes, but not the strongest in all.

Aggregating interactions gives the *Variable importance based on interactions*. It is a variable importance that takes account potential variables that can lead to the better separation between classes and are more interpretable than the global variable importance measure.

To show them in another view, one can see the *Variable importance over labels* which gives another picture.

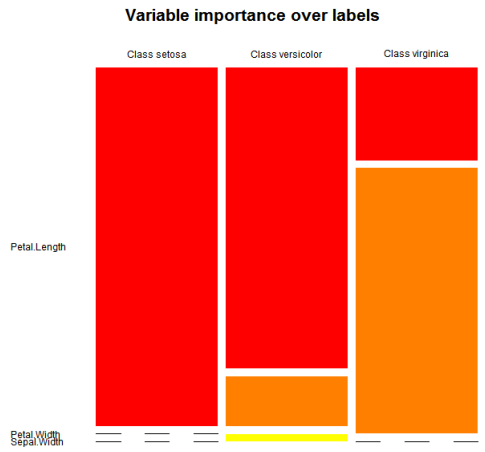

Figure 3: Variable importance over labels for the iris dataset

For each class, we see what variables are important, getting deeper in the analysis. *Petal.Length* explains most two species while *Petal.Width* explains more the last one. *Sepal.Width* remains important only by its interactions and not, here, by its contribution.

But what can we say about *Sepal.Length* ? We might want to know why it does not give informations. In the same case, we also might want to know why the others variables are so important. For that, we get in the *partial dependencies* for all variables (note that the call of the plot above just draws partial dependence for the most important variables letting user to choose, at the prompt, the variable he wants to draw).

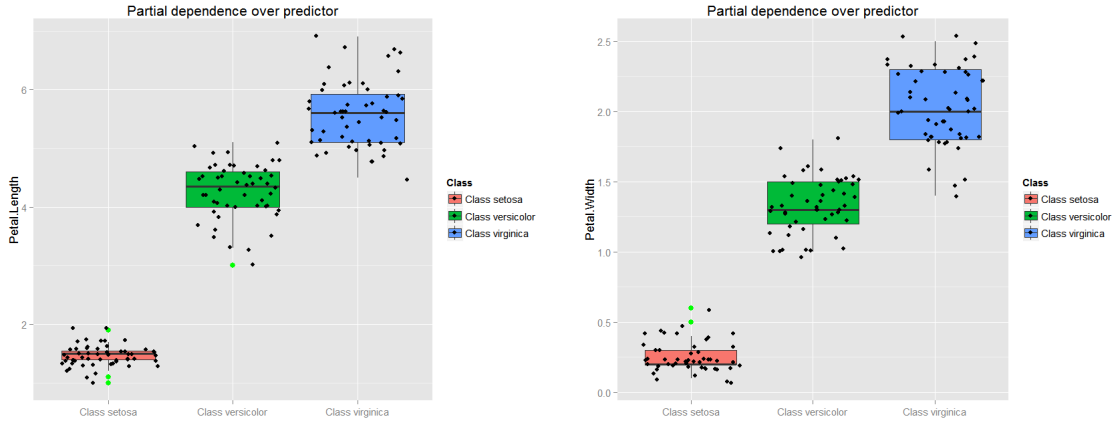

Figure 4: Partial dependencies for Petal Length and Petal Width

The main arguments of partial dependencies are to show what leads to separation between species (the classes), and if it is easy to achieve it or not. For example, here, if *Petal.Length* is below 2 and *Petal.Width* is below 0.5, the specie is, with great probability, *Setosa*. Then, only two variables are enough to identify species. What about the others ? To get more partial dependencies we just call the function with the importance object

```
# for the 'Sepal.Length' variable
pD.Sepal.Length = partialDependenceOverResponses(imp.ruf, Xtest = X,
whichFeature = "Sepal.Length", whichOrder = "all")
```

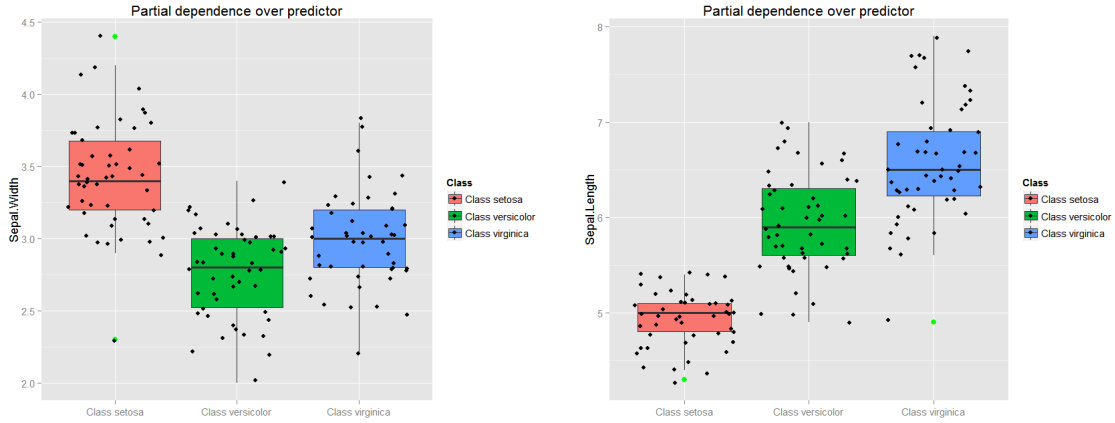

Figure 5: Partial dependencies for Sepal Width and Sepal Length

Sepal.Width and *Sepal.Length* are less discriminant variables and receive, then, less importance. The main difference between the two is that, as we write it above, *Sepal.Width* has more interactions with the others variables.

One may want to take decision about which variables (for example if there are many ones) are needed for each class to achieve enough explanation. We simply let *partial importance* give the information.

```
# for the setosa and virginica species
pImp.setosa = partialImportance(X, imp.ruf,  whichClass = "setosa")
pImp.virginica = partialImportance(X, imp.ruf,  whichClass = "virginica")
```

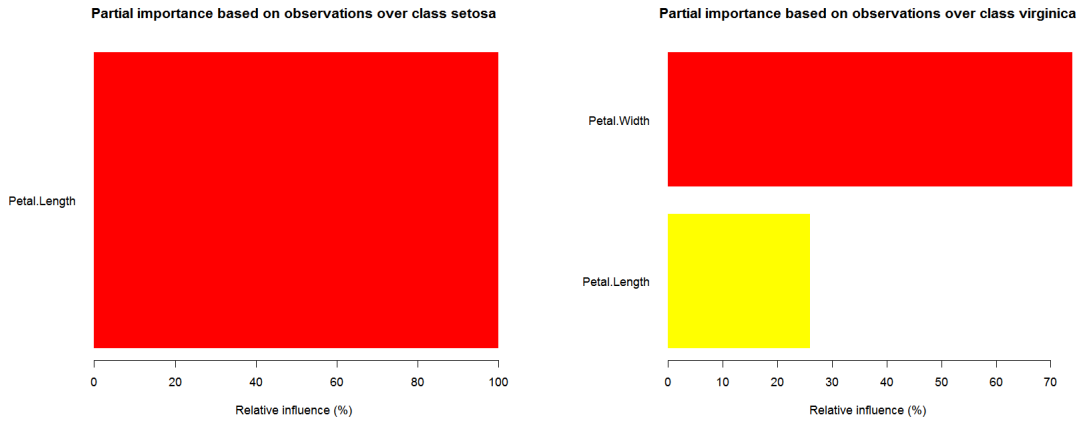

Figure 6: Partial importance for classes setosa and virginica

Partial importance states that *Petal.Length* is enough to know class *setosa* while for class *virginica* we need first *Petal.Width* and, less, *Petal.Length*.

One may want to compare how a single decision tree looks like. One of the main functionnality of this latter is to provide rules that one can easily understand. Since the trees in Random Uniform Forests are strongly randomized (unlike CART, where they are deterministic), visualizing one tree is not suitable for interpretation and no tree is better than another.

```
# get a tree
tree_100 = getTree.randomUniformForest(ruf,100)
# plot a tree in the forest
plotTree(tree_100, xlim = c(1,18), ylim = c(1,11))
```

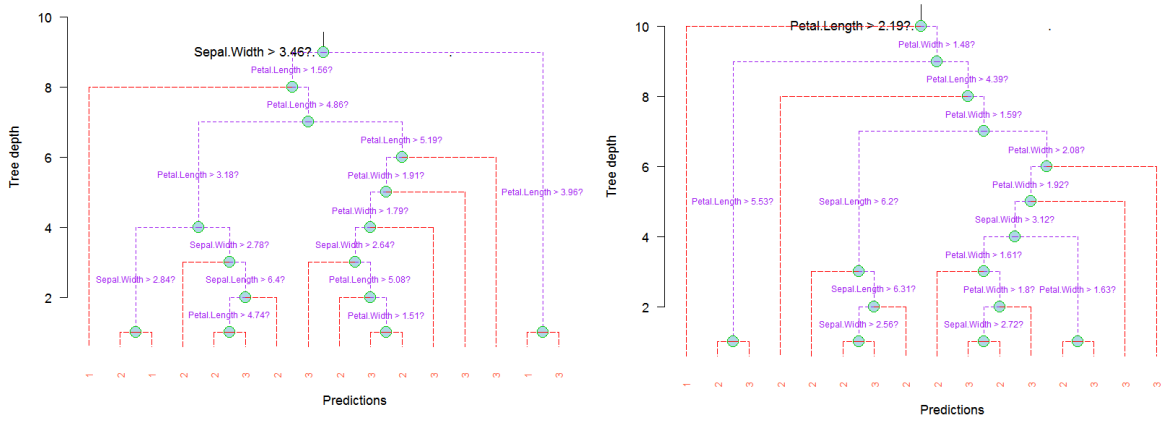

Figure 7: Two trees in the forest. Classes {1,2,3} are {setosa, versicolor, virginica}

One can see that rules are different and trees can be deep and large. Hence in Random Uniform Forests one interest is the bayesian framework. Data are fixed and parameters are random. We build almost all parameters (since convergence happens quickly) using randomness and let the data lie in the space in many different structures. In case of variable importance, the strong randomness ensure that effects we are getting are probably not random otherwise no specific shape would be expected.

5.1.2 Vehicle data

The next dataset we are assessing is also available on the UCI repository or using the *mlbench* R package. Following its description, given by the authors, *The purpose is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles.*

The dataset has 946 observations, 18 attributes and 4 classes.
Here, we simply assess prediction error of Random Uniform Forests, comparing to Random Forests, Extra-Trees and Gradient Boosting Machines.

1- At the first step, we use one train and test sets then see how the model works.

```
data(Vehicle, package = "mlbench")
XY = Vehicle

# column of the classes
classColumn = 19
# separate data from labels
Y = extractYFromData(XY, whichColForY = classColumn)$Y
X = extractYFromData(XY, whichColForY = classColumn)$X

# reproducible code and train sample (50%)
```

```
set.seed(2014)
train_test = init_values(X, Y, sample.size = 1/2)
# training sample
X1 = train_test$xtrain
Y1 = train_test$ytrain
# test sample
X2 = train_test$xtest
Y2 = train_test$ytest

# run model: default options, but setting number of trees to 500
ruf.vehicle  = randomUniformForest(X1, as.factor(Y1),
               xtest = X2, ytest = as.factor(Y2), ntree = 500)
# call
ruf.vehicle

# displays
# ...
Out-of-bag (OOB) evaluation
OOB estimate of error rate: 24.35%
OOB error rate bound (with 1% deviation): 27.53%

OOB confusion matrix:
          Reference
Prediction bus opel saab van class.error
      bus   93    3    4   1      0.0792
      opel   0   51   30   0      0.3704
      saab   1   41   67   4      0.4071
      van    1   10    8 109      0.1484

OOB geometric mean: 0.7271
OOB geometric mean of the precision: 0.7356

Breiman's bounds
Prediction error (expected to be lower than): 21.97%
Upper bound of prediction error: 35%
Average correlation between trees: 0.0782
Strength (margin): 0.5124
Standard deviation of strength: 0.3031
#...
```

```
Test set
Error rate: 26%

Confusion matrix:
          Reference
Prediction bus opel saab van class.error
      bus  119    2    4   0      0.0480
      opel   0   41   29   0      0.4143
      saab   0   61   68   0      0.4729
      van    4    3    7  85      0.1414

Geometric mean: 0.6951
Geometric mean of the precision: 0.7088
```

We observe that overfitting is happening. We can expect it using the Breiman's bound (21.97%), as it must be an upper bound of the OOB error. But the relation just works in one direction. OOB error can be lower than Breiman's bound and not preventing from overfitting.

Let's look how Random Forests do, getting their OOB error.

```
library(randomForest)
rf = randomForest(X1, as.factor(Y1), xtest = X2, ytest = as.factor(Y2))

rf

# displays
Call:
 randomForest(x = X1, y = as.factor(Y1), xtest = X2, ytest = as.factor(Y2))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 24.11%
Confusion matrix:
     bus opel saab van class.error
bus   91    1    1   2  0.04210526
opel   3   54   38  10  0.48571429
saab   5   31   64   9  0.41284404
van    0    1    1 112  0.01754386
                Test set error rate: 26.48%
Confusion matrix:
     bus opel saab van class.error
bus  119    0    0   4  0.03252033
opel   2   44   57   4  0.58878505
saab   4   31   63  10  0.41666667
van    0    0    0  85  0.00000000
```

Overfitting is also happening. Due to many redundancy in data, bootstrap does not work great and is, probably, one of the cause of overfitting.

2- The second step involves 10-fold cross validation. We report the results below. All algorithms run in their default parameters except GBM (*ntree = 500, interaction.depth = 24, n.minobsinnode = 1, shrinkage = 0.05*)

	Random Forests	ExtRa-Trees	GBM (optimized)	Random Uniform Forests
Test error	0.2518 (0.0535)	0.2506 (0.0514)	0.2234 (0.0422)	0.2412 (0.0461)

Table 1: Classification (Vehicle Silhouette). n = 946, p = 18. Cross-validation (10 folds). No tuning except GBM.

GBM works well but have to be optimized. Note that in the test set above, test error is 0.2443. Random Uniform Forests seem to have, for this dataset, well defined default parameters using more the dimension than Random Forests or Extra-Trees. We report standard deviation in parenthesis.

5.1.3 German credit data (cost sensitive learning)

German credit is another well known dataset. As the authors say, *this dataset classifies people described by a set of attributes as good or bad credit risks. Comes in two formats (one all numeric). Also comes with a cost matrix*. We use the numeric format of the data which has 1000 observations, 20 attributes and 2 classes (1: Good, 2: Bad).

In the german credit data, test error is not suitable since there is a cost matrix (5 if the classifier predicts Good when it is actually Bad and 1 if the classifier predicts Bad when it is Good). Therefore we have to minimize the cost. To achieve that, we have to optimize AUC (area under ROC curve) and AUPR (area under precision-recall curve). As in the definition of Fawcett (2006), AUC is the probability that the classifier ranks a randomly positive instance ('Bad' class) higher than a negative one. AUPR is an equivalent of the probability of matching a positive case if it is really one. Hence for credit scoring both measures need to be high.
- AUC gives the level of confidence in matching positive cases.
- AUPR gives the limit in the capacity to match positive cases.

Since a cost matrix is provided, *cost sensitive accuracy* is closer to a real life case. It is given relative to the confusion matrix and the costs associated and for the german credit dataset we have

$$cost\ sensitive\ accuracy = 1 - \frac{n_{2,1} + 5n_{1,2}}{n_{.,1} + 5n_{.,2}},$$

$n_{2,1}$ are the cases classified as 'Bad' (class 2) while they are actually 'Good' (class 1),
$n_{1,2}$ are the cases classified as 'Good' while they are actually 'Bad',
$n_{.,1}$ is the total number of 'Good' cases,
$n_{.,2}$ is the total number of 'Bad' cases.

Note that we omit $n_{1,1}$ and $n_{2,2}$ since their costs are zero. We also used the average cost defined by

$$average\ cost = \frac{n_{2,1} + 5n_{1,2}}{n}.$$

Classes are also imbalanced and without a *class reweighting* method or an algorithm insensitive to imbalanced distribution, one will hardly minimize cost. Random Uniform Forests explicitly implement class reweighting using, with slight modifications, the methodology of Chen et Al. (2004). Since we only find SVM as a method that uses class reweighting in a simple manner (one weight for each class), we keep it for comparison (Random Forests have a class weight option, but it seems to not be enabled).

Here are the parameters we use:
- The *classwt* option in Random Uniform Forests use weights, for each class, that reflects the cost matrix. We do not optimize it and set it according to the class distribution. If 'Y' is the R vector of labels in the sample we have, using the *e1071* R package help :

```
# for SVM
weights_svm = 100/table(Y)

# Random Uniform Forests
distribution = 100/table(Y)
weights_ruf = round(distribution/min(distribution), 4)
```

Others parameters are for SVM :
kernel = "sigmoid"
cost = 0.5.

For Random Uniform Forests, we use some sampling techniques :
- the option for the number of features to try for each node, namely *mtry = 3*,
- we remove the bootstrap and use sub-sampling with its default value,
subsamplerate = 0.7 and *replace = FALSE*,
- we perturb randomly a small part (5%) of the class distribution (before reweighting), setting *outputperturbationsampling = TRUE* and *oversampling = 0.05*,
- we set *ntree = 500*, the number of trees, and *nodesize = 10*, the minimal number of observations per terminal node, to stabilize the forest.
Others options are let to their default values. One can find how parameters are used in the OOB evaluation below.

We load first the data and simply run the cross-validation script (with some modifications to capture measures). For reproducibility, one just have to use the same seed.

```
germanCredit = read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/
statlog/german/german.data-numeric")
XY = germanCredit
# set.seed(2014)
# separate Y and X and call a cross-validation procedure
# ...
```

The results are given below :

	SVM	Random Uniform Forests
Cost sensitive accuracy	0.7492 (0.0553)	0.7651 (0.0425)
Average cost	0.551 (0.1297)	0.517 (0.1)
AUC	0.7333 (0.0550)	0.6979 (0.0459)
AUPR	0.6372 (0.0716)	0.6451 (0.0661)
Test error	0.283 (0.0605)	0.385 (0.0479)

Table 2: Classification (German credit). n = 1000, p = 24, Cross-validation (10 folds). Optimized to maximize cost sensitive accuracy.

Both methods achieve good scores using the class weight option, which is the main argument to get both good cost sensitive accuracy and AUC. While SVM relies more of the latter pushing it in a high level, Random Uniform Forests are more focused on the AUPR (area under precision-recall curve) letting, even, test error increase to get the point. To reach good scores, optimization is needed for both methods. One special issue with the dataset is redundancy in data since many are categorical (with 2 unique values). Thus, Random Uniform Forests do not benefit from dimension and others random techniques are needed to balance accuracy. Typically *output perturbation sampling* is the argument that pushes cost sensitive accuracy beyond the 0.74 limit (and reduce the average cost under 0.56). In both methods, better scores can be probably achieved.

```
OOB evaluation and the OOB classifier
```

We, then, present the OOB evaluation and some tools we may use to assess the dataset.

```
# model and the OOB classifier
ruf.germanCredit = randomUniformForest(X, as.factor(Y), nodesize = 10,
    mtry = 3, outputperturbationsampling = TRUE, oversampling = 0.05,
    subsamplerate = 0.7, replace = FALSE, classwt = weights_ruf, ntree = 500)

ruf.germanCredit
# displays the OOB evaluation
#...
Out-of-bag (OOB) evaluation
OOB estimate of error rate: 37%
OOB error rate bound (with 1% deviation): 40.14%

OOB confusion matrix:
          Reference
Prediction   1   2 class.error
         1 370  40      0.0976
         2 330 260      0.5593

OOB estimate of AUC: 0.6976
OOB estimate of AUPR: 0.6385
```

```
OOB estimate of F1-score: 0.5843
OOB geometric mean: 0.6768
#...
```

OOB cost sensitive accuracy is 0.7590 and *OOB average cost* is 0.53. OOB evaluation is more pessimistic than cross-validation, since it evaluates all the data at once when cross-validation divides the data to have a full evaluation. Hence, OOB analysis is closer to the training (and test) set, since the model will be a generalization of the OOB classifier if a test set is provided. In cross-validation, and for Random (Uniform) Forests, it will not.

In the OOB evaluation above, one can see how it is needed to let the test error increase in order to reduce the average cost. In the german credit dataset the hard task is to reach an high AUPR, while not reducing AUC, that controls both *sensitivity* and *precision* the key ingredients of a low cost.

To see why the task is not so easy one can call visualization of some metrics :

```
# get the ROC curve
roc.curve(ruf.germanCredit, Y, unique(Y))

# get the Precision-Recall curve
roc.curve(ruf.germanCredit, Y, unique(Y), falseDiscoveryRate = TRUE)
```

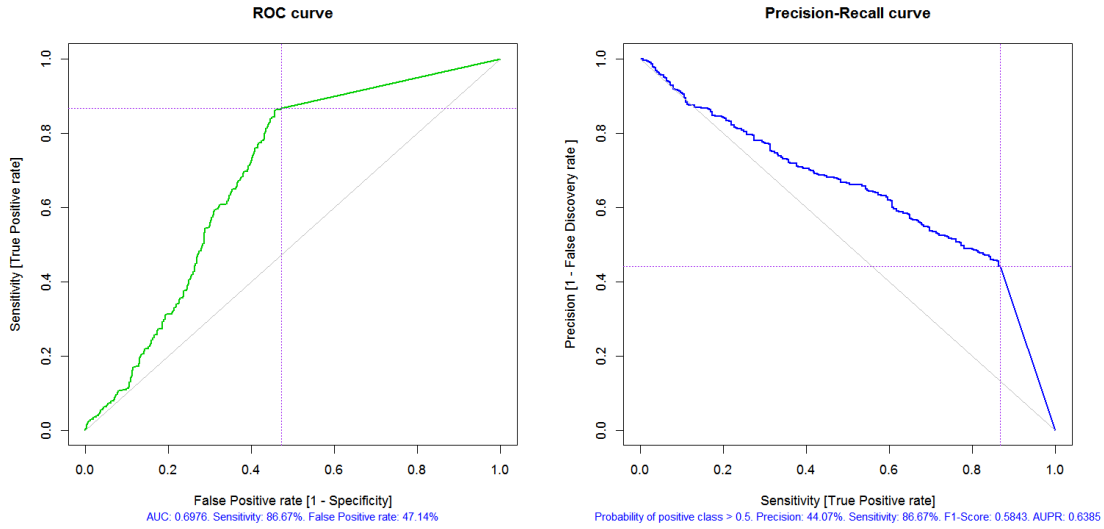

Figure 8: ROC and Precision-Recall curves for the German credit dataset (OOB evaluation).

We can see both plots look like each other (but they are not identical) and the main point is in the beginning of each graph : we observe that for, apparently, some cases (up to 20% of all) it is hard to distinguish if they are from class 'Bad' or not and for the precision-recall it is even more clear; the precision decreases so quickly that weights do not suffice to maintain a discriminant good rule without help of other techniques.

The good sensitivity (recall) compensates but test error can suffer. In fact, even if randomized models can reach good results for this dataset, it appears that it leads to some instability. One can look how the test error is decreasing with the number of trees:

```
plot(ruf.germanCredit, ylim = c(0.25, 0.4))
```

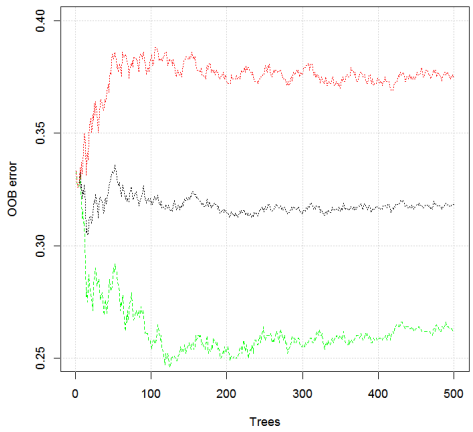

Figure 9: OOB error with the number of trees for the German credit dataset.

The error rate does not decrease (or a little) with the number of trees and is highly unstable (red and green lines indicate where OOB could lie, while the black line estimates its actual value). As a consequence, a single (or a few) tree(s) seem to get results closer to the forest ones. In fact, that holds only for the test error. The forest classifier brings more than stabilization (which is not clear above). It provides more adaptation to the problem. Comparing to a forest of 5 or 10 trees, the ensemble with 500 trees increases AUC by more than 10% and AUPR by more than 30%, while test error is reducing only by 5%.

5.1.4 Breast cancer data (more on Breiman's bound)

We conclude the evaluation for classification by some words about the Breiman's bound. One can note that for the others data sets, it does not work as expected. In fact, it works but, to be clear, Breiman's bound is not designed to get optimal classifier but to get a robust one. For example, in the german credit data, set $mtry = 1$ (getting a *purely random forest*) leads to push OOB error under the Breiman's bound. We do not get the optimal classifier, but probably the more robust. In other words, Breiman's bound can, in most of the cases, be the upper bound but one has to make choice between optimality and robustness. Three elements are needed :
- have enough data,
- balanced classes, but real life problems usually have imbalanced classes,
- low correlation between covariates.

The first condition is necessary, since one of the metrics, the correlation, requires to find pairwise cases between trees. Then, even one has 500 observations, pairwise OOB

cases will not be so numerous and increase the number of trees will not change a lot the situation. Random Uniform Forests are not insensitive to imbalanced classes, then majority class can have influence on the strength (the second parameter needed). Correlated covariates tend to increase the risk of overfitting, then keeping an upper bound is less easy. Hence, changing default parameters of the model is usually necessary to get the Breiman's bound as it should be. We show how it can happen in the lines below.

Breast cancer dataset is available on the UCI repository. Description can be found in the following link :
http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

The dataset has 569 observations, 32 attributes and 2 classes, but the formatted data really have 11 attributes.

```
# load the data
breastCancer = read.table("http://archive.ics.uci.edu/ml/
machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data",
sep = ",")
XY = breastCancer

# remove the first attribute which is a unique ID
XY = XY[,-1]

# get observations and labels (the 10th column)
Y = extractYFromData(XY, whichColForY = 10)$Y
X = extractYFromData(XY, whichColForY = 10)$X
```

Here, we use default parameters for all models and report the 10-fold cross-validation error. Note that some algorithms may need a R matrix, while others need a data frame and test error may be affected, depending on how one is converting data frame to a matrix.

	RF	ET	SVM	Random Uniform Forests
Test error	0.314 (0.0161)	0.03 (0.0124)	0.0371 (0.0153)	0.0314 (0.0175)

Table 3: Classification (Breast Cancer). n = 569, p = 9. Cross-validation (10 folds). No tuning.

Random Uniform Forests stay close to others and achieving an high accuracy is common for many algorithms with this dataset.

Let us look the Breiman's bound. There is no chance we match it using default parameters, due to high correlation in covariates and imbalanced classes. Then, To get the point, we use the *output perturbation sampling* option (changing 20% of the labels with *oversampling* option) which will increase correlation between trees and decrease the strength. Then we increase *mtry* to 64, having more competition between nodes, and set the number of trees to 500 in order to stabilize the forest. All parameters are set only

with the first training sample (50% of the data, with the seed we use since the beginning). Then we generate five random training and test samples and look for OOB error, Breiman's bound and test error.

	OOB error	Breiman's bound	Test error
set 1	0.0315	0.0395	0.0315
set 2	0.0287	0.0303	0.0257
set 3	0.0401	0.0335	0.0257
set 4	0.0372	0.0355	0.0286
set 5	0.0287	0.0339	0.04

Table 4: Classification (Breast Cancer). n = 569, p = 9. Training sample (50%) randomly generated for each set.

Breiman's bound is now much closer to the OOB error, leading to more insights on correlation and strength. The OOB evaluation on the whole data leads to an OOB error of 0.0372 and the Breiman's bound to be 0.0404. One can even control more the Breiman's bound and let it be a strict upper bound. In real life cases, it is the way that Random Uniform Forests are designed to proceed in order to not be so optimistic about the generalization error.

5.2 Regression

In regression, we use Random Uniform Forests with their default parameters (except *mtry* for the first dataset) adding only for some cases the *output perturbation sampling* option to show that convergence still happens. We also use *post processing*.

5.2.1 AutoMPG data

The first dataset we use, *autoMPG*, can be found on the UCI repository. This is its description : *this dataset is a slightly modified version of the dataset provided in the StatLib library. In line with the use by Ross Quinlan (1993) in predicting the attribute "mpg", 8 of the original instances were removed because they had unknown values for the "mpg" attribute. The original dataset is available in the file "auto-mpg.data-original".*
The data have 398 observations and 8 attributes We, first, get the modified data and run a 10-fold cross validation.

```
# load the data,
autoMPG = read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/
auto-mpg/auto-mpg.data")
XY = autoMPG

# remove the last column (car models) and get the mpg attribute (the first one)
XY = XY[,-9]
Y = extractYFromData(XY, whichColForY = 1)$Y
X = extractYFromData(XY, whichColForY = 1)$X
set.seed(2014)
```

We override categorical variables, using data as an R matrix, otherwise Random Forests (RF) algorithm do not want to run due to some misunderstanding with the data imported (data frame) and how the algorithm recognizes categorical variables. One can note that one variable is truly categorical (the last one, 'origin') while the first one ('cylinders') is discrete and can be ordered.
We also use the *tuneRF()* function to find the best *mtry* for Random Forests. The optimal value is 4, and we use it for Extra Trees (ET).
SVM uses a *cost* of 2.5 and the radial *kernel* (assessed with the *tune.svm()* function).
GBM (Gradient Boosting machines) use the parameters we define for others data sets.
rUF stands for Random Uniform Forests and *mtry* is set to 14.

	RF	ET	SVM	GBM	rUF	rUF (post processing)
MSE	7.46 (2.73)	7.1 (2.71)	7.31 (2.49)	7.48 (2.69)	7.15 (2.75)	7.12 (2.76)

Table 5: Regression (AutoMPG). n = 398, p = 7. Cross-validation (10 folds). All algorithms optimized.

Extremely Randomized Trees (ET) work well for regression, so we wonder why increasing the mtry parameter in Random Uniform Forests do not lead to better results (ET use only mtry = 4). Increasing the number of trees from 100 to 200 in Random Uniform Forests do not improve the results. But ET use more observations (the whole sample) while Random Uniform Forests use sub-sampling (with default level 0.7). Using exactly the same parameters than ET lead to not change the MSE (around 7.13). In this dataset, sub-sampling or working more on dimension seem to have no influence.

We assess now the whole dataset (with the OOB evaluation) to see how look like the errors and how we can explain the *miles per gallon (mpg)* variable with others predictors.

```
# run Random Uniform Forests
ruf.autompg = randomUniformForest(X,Y)

ruf.autompg
# displays
#...

Out-of-bag (OOB) evaluation
Mean of squared residuals: 7.1946
Mean squared error bound (experimental): 8.268037
Variance explained: 88.22%

OOB residuals:
      Min        1Q    Median      Mean        3Q       Max
-17.42000  -1.16500   0.27250   0.04214   1.43100  11.60000
Mean of absolute residuals: 1.84719
```

```
Breiman's bounds
Theorical prediction error: 6.930812
Upper bound of prediction error: 7.023337
Mean prediction error of a tree: 15.13134
Average correlation between trees residuals: 0.4642
Expected squared bias (experimental): 0.000442
```

OOB evaluation is close to cross-validation. We can see that the percentage of variance explained is high and lead to have confidence in the modeling. Random Uniform Forests display many others informations that one can use to understand the purpose.

Suppose that we want to know which are the variables that increase the consumption (hence reducing the miles per gallon). We call *importance()* function then *partialImportance()*.

```
# importance
imp.ruf.autompg = importance(ruf.autompg, Xtest = X, maxInteractions = 7)

# which variables leads to consumption more than average
pImp.ruf.autompg = partialImportance(X, imp.ruf.autompg,
     threshold = round(mean(Y),4), thresholdDirection = "low")
```

We get variables 'V5'(the weight of the car) and 'V4' (the horsepower) to be the most important in the increase of consumption. Let us look how 'mpg' is evolving with these two variables.

```
# Dependence between V5 and V4 and influence on the 'miles per gallon'
pDependence.V5V4 = partialDependenceBetweenPredictors(X, imp.ruf.autompg,
            c("V5", "V4"), whichOrder = "all")
```

We get plots (and the level of the variables interaction) that give some insights :

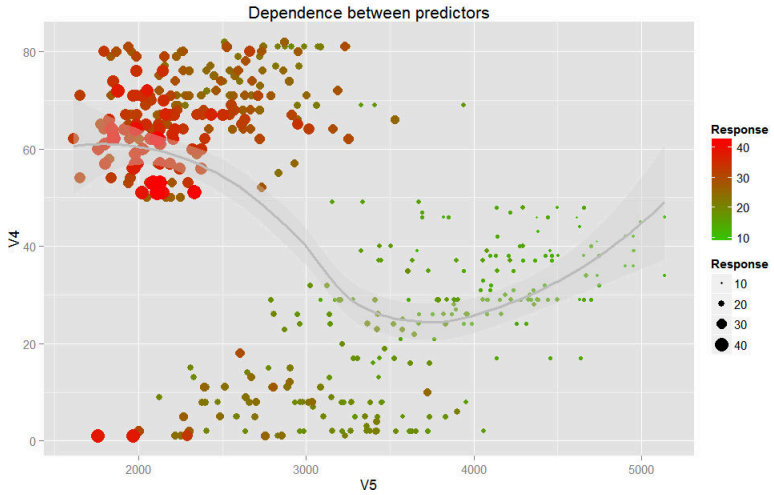

Figure 10: Partial dependencies between 'V5' (weight) and 'V4' (horsepower) and effect on the 'miles per gallon' (Response).

We can see that high values of 'mpg' depend on a low weight and a high horsepower and when a threshold (in the weight) is crossed, 'mpg' decreases no matter what the horsepower is. Dependence between 'weight' and 'horsepower' seems non linear; weight increases while horsepower decreases until a minimum, after then weight increases while horsepower increases too. The interesting point is we know, for the whole relation, what is happening to the miles per gallon.

5.2.2 Abalone, Concrete compressive strength, Boston housing data

We conclude regression by an evaluation with three data sets (also available on UCI). For sake of brevity, we do not use R code here and simply report the results of a 10-fold cross-validation procedure. Random Uniform Forests run with their default parameters.

	RF	ET	SVM	GBM	rUF	rUF*
Abalone	4.59 (0.55)	-	4.52 (0.54)	4.88 (0.7)	4.64 (0.54)	4.74 (0.61)
Concrete	23.05 (5.3)	21.67 (5.38)	34.55 (5.3)	14.94 (3.95)	21.64 (5.28)	22.33 (5.82)
Housing	10.19 (3.87)	9.43 (3.69)	11.23 (6.17)	8.87 (5.73)	9.32 (3.12)	9.57 (4.57)

Table 6: Regression (Abalone, Concrete compressive strength and Boston housing data sets). Reporting mean squared error and standard deviation of the 10-fold cross-validation. All algorithms optimized except Random Uniform Forests.
* : output perturbation sampling (all training responses replaced by random Gaussian values), and post processing enabled.

The most surprisingly fact is linked with the prediction error, for the concrete dataset, of GBM, far below all others. To be sure we ran again the model and change to 2-fold cross-validation. For some data sets GBM is hard to beat. Using the whole sample in Random Uniform Forests reduces the gap, with a MSE of 19.65.

But the point is, when completely changing the values of training responses by random Gaussian ones then doing post-processing, Random Uniform Forests are close to their standard case. Moreover, perturbation of the outputs do not lead to worst results than Random Forests. We see that, in regression, it seems more effective to not choose the optimal cut-points, as both prediction errors in Extremely Randomized Trees and Random Uniform Forests tend to be lower or similar to the ones in Breiman's Random Forests.

5.3 Incremental learning, big data and/or streaming data

Nowadays, it is difficult to not talk about *big data* and how *machine learning* use it to make better predictions.
For our point of view, big data are all the data that can not be treated on a single workstation, in a reasonable time.
Big data come with two main tasks : produce descriptive statistics and produce predictive models. Machine learning is involved in the second one.

When building predictive models with big data, the main problem is how to take account all the information without loosing (too much) accuracy, knowing one will never be able to treat all the data at once. The usual solution is to break data in blocks and to give models the ability to update themselves with incoming data, knowing that each chunk of data will be able to reside in (random access) memory.

Machine learning applied to big data are faced to, at least, two problems which can lead to some severe complications :
- shifting distributions, for which statistical learning has, currently, no established theory,
- model parsimony, which asks the question of the necessity of infinite learning. In other words, since big data do not have end, does a model need to learn continuously the data as they come ?

In fact, in *statistical learning*, the problem is not really the size but whether or not the *i.i.d.* assumption will hold as one is building blocks. If it holds then, no matter what the size is, one will see the limit in prediction error. If it does not hold, then the prediction error will probably have oscillations and one has to make it decrease with the number of blocks, without knowing where the limit will be.

Main purpose of machine learning is to produce algorithms that will be able to generalize (well) their current knowledge, calling either their memory or their learning process. Random Uniform Forests try to build such models, meaning that they are able to combine many different models on different data and try to get the best of all data as if the latter could reside in memory. We note that all data could reside on hard disk or SSD, but then access times would be too slow to allow reasonable time of computation.

Let us consider a practical case on artificial data with the following method :
1 - let us suppose that we have three data sets. The first two come one after the other and the third, the test set, come at the end,
2 - we suppose that the two first data sets do not have necessarily the same joint distri-

bution of the pair (X, Y),
3 - the (joint) distribution of the third dataset depends on the distribution of the first and the second ones and on its own one,
4 - since the hardware is limited, the maximum size a algorithm can compute at the same time is the size of the first data set,
5 - *we can not merge the first two data sets. One can imagine having 100 or more chunks of data. Sampling and merging would be a problem with a shifting distribution.*

With the shifting distribution, the point 5 is the practical case of problem arriving with big or streaming data. For ease of computation, we consider small data sets but the problem does not change as long as only chunks of all the data can be computed at the same time. Then, instead of sample data or update parameters, the point is to update models. For our example, since we only use two training sets, the purpose is to show how incremental learning is basically done.

We provide the full R script[1] for reproducibility and show how Random Uniform Forests work on the problem.

`Protocol`

1 - generate the first training sample and take a part of it to build the first part of the test sample,
2 - generate the second dataset and take a part of it to to build the second part of the test sample. The (random) parameters of the distribution have probably changed.
3 - generate the last part of the test sample and merge it with the others parts kept from point 1 and 2. The parameters of the distribution are no more the same than the ones of the two training samples.

The main argument above is that the *i.i.d.* assumption is probably no longer valid, since each sample generates randomly its own parameters. The only thing that remains constant is the relation between observations and labels, made to match a rule with some random parameters. The algorithm must find its own path to match test labels with the constraint or using a constant-time for learning.

`Data model`

Let us first write the model that generates the data. Let us suppose a pair (X, Y) of random variables, where X is a d-dimensional Gaussian vector, $X \in \mathbb{R}^d$, $d > 6$, and its components are supposed to be independent. $Y \in \{0, 1\}$. We generate samples of (X, Y) using the following procedure :

[1]The procedure has been updated to be as clear as possible, since in the previous one it lead to confusion and was not reproducible just by copying the R code. More tests have been added and an unsupervised case example is provided now.

for each $j \in [1, d]$, consider $Z^{(j)}$, $Z^{(j)} \sim \mathcal{U}_{[-10,10]}$.
For each $j \in [1, d]$, generate $X^{(j)}$, where

$$X^{(j)} \sim \mathcal{N}(z_j, z_j^2),$$

with z_j, one realization of the random variable $Z^{(j)}$.
We, then, define the rule for Y, writing

$$Y = \mathbf{I}_{\left\{R(X,\epsilon) > \mathbb{E}(R(X,\epsilon))\right\}},$$

with $R(X, \epsilon) = 2\left(X^{(1)}X^{(2)} + X^{(3)}X^{(4)}\right) + \epsilon_1 X^{(5)} + \epsilon_2 X^{(6)}$,
$\epsilon_1 \sim \mathcal{U}_{[-1,1]}$, $\epsilon_2 \sim \mathcal{U}_{[-1,1]}$.

`Script to generate data:  Example 1`

The whole script can be just pasted to the R command line interface.

```
library(randomUniformForest)
set.seed(2014)
n = 1000;  p = 50 # one may reduce or increase number of observations or variables

# call the script that generates a p-dimensional randomized Gaussian vector;
# for each variable mean and standard deviation are equal and randomly picked
# between -10 and 10, using the continuous Uniform distribution

X = simulationData(n, p)
X = fillVariablesNames(X)

# generate the rule to define labels

epsilon1 = runif(n,-1,1)
epsilon2 = runif(n,-1,1)
rule = 2*(X[,1]*X[,2] + X[,3]*X[,4]) + epsilon1*X[,5] + epsilon2*X[,6]
Y = as.factor(ifelse(rule  >  mean(rule),  1,0))

# retain some data (30%) to generate first part of the test sample

randomIdx = sample(n, floor(0.30*n))
X.test1 = X[randomIdx,]

# and generate the first training sample

X.train1 = X[-randomIdx,]
Y.train1 = Y[-randomIdx]

# Then, generate the second training sample :
```

```
set.seed(1992)
X = simulationData(n, p)
X = fillVariablesNames(X)
epsilon1 = runif(n,-1,1)
epsilon2 = runif(n,-1,1)
rule = 2*(X[,1]*X[,2] + X[,3]*X[,4]) + epsilon1*X[,5] + epsilon2*X[,6]
Y = as.factor(ifelse(rule > mean(rule), 1, 0))

# retain  some  data  (30%)  to  generate  second  part of the test sample
# and generate the second training sample

randomIdx = sample(n, floor(0.3*n))
X.test2 = X[randomIdx,]

X.train2 = X[-randomIdx,]
Y.train2 = Y[-randomIdx]

# generate the test set :
# one third of the data coming from its own distribution

set.seed(1968)
X.test3 = simulationData(floor(0.3*n),  p)
X.test3 = fillVariablesNames(X.test3)

# and merge it with other parts generated

X.test = rbind(X.test1, X.test2, X.test3)

# test labels have the same rule than training labels,
# using the observations of the test set

epsilon1 = runif(nrow(X.test),-1,1)
epsilon2 = runif(nrow(X.test),-1,1)
rule = 2*(X.test[,1]*X.test[,2] + X.test[,3]*X.test[,4]) +
epsilon1*X.test[,5] + epsilon2*X.test[,6]
Y.test = as.factor(ifelse(rule > mean(rule), 1, 0))

# Finally, we have two train samples 'X.train1' and 'X.train2' with each
# 700 observations and 50 variables, their labels 'Y.train1' and 'Y.train1',
# and a test set 'X.test' with 900 observations and 50 variables
# and its labels 'Y.test'.
```

Random Uniform Forests achieve *incremental learning*, that is, modeling the data as they come (by chunks), update the model and use a constant-time learning process. Hence old data do not need to be processed again when new data arrive. In Random Uniform Forests, the learning process becomes, after some time which can be long, a memory process, meaning that the learning tends to be less and less necessary as the model grows.

We did not find any *online* or incremental learning algorithm within R packages, except the *randomForest* package and the *gbm* one (but less straightforward). Hence we use randomForest for comparison. Learning tasks require three core processes:
1- compute a model for each chunk, adding eventually pre-processing step like scaling,
2- combine models,
3- post-process, eventually, the combined model to adapt the voting mechanism.

All tasks are simple but may require algorithmic and model refinements. Usually, the prediction error is the metric that gives insights on where and how efforts have to be made. We proceed with two steps :

A - the first one calls the *purely random forest*; it is a random forest model in which the classifier does not depend (or very weakly depends) to the data. For each node, each candidate variable and cut-point are chosen randomly. Random Uniform Forests, unlike randomForest algorithm, allow *purely uniformly random forest* in which the randomness is, in all parts of the algorithm, uniform. In the best case, purely uniformly random forests will converge and the main argument, here, is to be independent to the data and to capture (almost) all parameters of the model. Hence the latter states that if convergence happen independently to the data, then the provided OOB error will be the one we can expect, at the best, when going toward the test set since the *i.i.d* does not longer hold. However, empirically, it is also expected that distribution in the latter has some links with the ones in training sets or, at least, that the structure of the relation between observations and labels does not change for any data proceeded. In other words, the distribution may shift, but relation between observations and labels holds, at least on our example. We can, then, compute the *purely uniformly random forest*.

```
Learning models
# we use nodesize = 5 and ntree = 500 for stabilization

pURF.model1 = randomUniformForest(X.train1, Y = as.factor(Y.train1),
              ntree = 500, nodesize = 5, mtry = 1 )
pURF.model2 = randomUniformForest(X.train2, Y = as.factor(Y.train2),
              ntree =500, nodesize = 5,  mtry = 1)

# combine
pURF.combineModels = rUniformForest.combine(pURF.model1, pURF.model2)

# and look for OOB evaluation
pURF.combineModels

# it displays
# ...
Out-of-bag (OOB) evaluation
OOB estimate of error rate: 44.93%
OOB error rate bound (with 1% deviation): 48.15%
```

```
OOB confusion matrix:
    0   1 class.error
0 398 308      0.4363
1 321 373      0.4625

OOB estimate of AUC: 0.5506
OOB estimate of AUPR: 0.5369
OOB estimate of F1-score: 0.5425
OOB (adjusted) geometric mean: 0.5506

Breiman's bounds
Prediction error (expected to be lower than): 14.17%
Upper bound of prediction error: 16.06%
Average correlation between trees: 0.0095
Strength (margin): 0.2701
Standard deviation of strength: 0.1012
```

Since distribution will probably shift, we have also interested by AUC and Breiman's bounds. The latter states that the only hope resides in the very low correlation of trees. Surprisingly it is so low that bound seems too much optimistic. One can note that correlation of trees overrides distribution. More precisely, it states that if trees are very weekly dependent, then prediction error can be reduced even if the shifting distribution hurts the *strength* (second part of the Breiman's bound). But, it is more secure to rely on the OOB classifier that states that one can not expect good prediction error. As a consequence, OOB estimate of AUC is also not good. *In all manners, one can not beat, or even reach, the i.i.d case with all data proceeded at once. Moreover, in the non-i.i.d case, prediction error can be anywhere.* Hence, if convergence happens, the estimates we get are the best results we do expect. The problem is that guarantees disappear as faster as the *non-i.i.d* arises.

B- In order to achieve a compromise, we come back to Random Uniform Forests and build a model that will strongly stress the competition between nodes, leading to a high value of the *mtry* parameter. The link between points A and B relies on the following arguments : since we had a non-optimal highly randomized classifier that gives us its best estimate, if we build a more optimal one, the test error we get should be bounded by the estimate of the non-optimal classifier. Due to the stochastic nature of Random Forests and to the correlation we got between trees, we can expect both properties to work, at least, decently with a shifting distribution.

Back to the R code we get :

```
rUF.model1 = randomUniformForest(X.train1, Y = as.factor(Y.train1),
               ntree = 500, nodesize = 5, mtry = 500, OOB = FALSE)
rUF.model2 = randomUniformForest(X.train2, Y = as.factor(Y.train2),
               ntree = 500, nodesize = 5, mtry = 500, OOB = FALSE)
# combine
rUF.combineModels = rUniformForest.combine(rUF.model1, rUF.model2)
```

```
# predict
predictions = predict(rUF.combineModels, X.test)

# and assess
stats.predictions = model.stats(predictions, Y.test)

# the function above displays
Test set
Error rate: 19.56%

Confusion matrix:
          Reference
Prediction   0   1 class.error
         0 507 124      0.1965
         1  52 217      0.1933

Area Under ROC Curve: 0.7717
Area Under Precision-Recall Curve: 0.6672
F1-score: 0.7115
Geometric mean: 0.7597
```

We get the results which were expected and, more essential, we get stable errors, i.e., running again the algorithm will let errors vary only a little. However, the method has no reason to be specific. We, then, assess Breiman's Random Forests but we can't increase competition between nodes since cut-points are optimized one variable after the other, looking all its observations. The best we can do is to optimize the *mtry* parameter using the *tuneRF()* function for each training set. Here are the results :

```
library(randomForest)
rf1 = randomForest(X.train1, as.factor(Y.train1), mtry = 14, norm.votes = FALSE)
# data in the second model should have the same size than in the first one
rf2 = randomForest(X.train2, as.factor(Y.train2), mtry = 28, norm.votes = FALSE)

rf.combineModels = combine(rf1, rf2)
rf.predictions = predict(rf.combineModels, X.test)
rf.stats.predictions = model.stats(rf.predictions, Y.test)

# displays
Test set
Error rate: 38.11%

Confusion matrix:
          Reference
Prediction   0   1 class.error
         0 378 162      0.3000
         1 181 179      0.5028
```

```
Area Under ROC Curve: 0.6006
Area Under Precision-Recall Curve: 0.472
F1-score: 0.5107
Geometric mean: 0.5958
```

We achieve higher test error with Random Forest and presume that one reason is probably coming from the local optimization of cut-points. Using only the first training sample leads to test error over 45%. We can also evaluate the *i.i.d.* case, relying on the OOB classifier and getting an error around 10%.

`Example 2`

We also wanted to know how algorithms would be sensitive to the shifting in distribution. We repeated the same procedure, changing the seeds (adding the value 31 to each one) and using this time:
a) 50% of the first data set, instead of 30%, with a seed value of 2045
b) 30% of the second data set (as before), with a seed value of 2023
to build first and second part of the test set, having 15% of the data coming from its own distribution (with a seed value of 1999).

We got a test error from 27% to 32% for Random Uniform Forests and around 43% for Random Forests.

`Example 3`

We repeated again, with the new seeds, the task having :
a) 15% of the first data set,
b) 30% of the second data set,
to build first and second part of the test set, having 55% of the data coming from its own distribution.

This time the test error raised to 56% for Random Uniform Forests and a test error from 43% to 52% for Random Forests.

As the distribution is shifting, we see that it becomes more and more difficult to reach a good prediction error. In our example, depending on seeds and on the shifting distribution, trying to learn samples becomes a very hard task and unless ones sees a part of the test set, incremental learning meets its limit. The task is so complicated that even with the two training samples at once, things do not change.

To summarize, if the distribution is shifting slowly we can have some hope. For parametric models, the main problem is in the manner to update parameters and not forget too early the previous distributions. For non-parametric ones, it is the growth of the model. In case of ensemble (and non-parametric) learning, one will usually need to choose the right subset of base models.

If the distribution is shifting faster, as in the last part of our example, *Clustering* and

Dimension Reduction seem to be a very good alternative. For our example, using the unsupervised mode of Random Uniform Forests leads to far better results. In fact, whenever one thinks to be in the *non-i.i.d* case, or to be close, both supervised and unsupervised learning might be needed.

```
Unsupervised Learning as a co-worker of Incremental Learning


# The unsupervised case  with the last example :
# seeds are set respectively to 2045, 2023 and 1999
# a) 15% of the first data set,
# b) 30% of the second data set,
# are used to build first and second part of the test set,
# having 55% of the data coming from its own distribution.

# 1 - compute the model, letting it find the number of clusters

rUF.unsupervised = unsupervised.randomUniformForest(X.test)

# 2- If needed, modify the number of clusters to have two

rUF.unsupervised2 = modifyClusters(rUF.unsupervised, decreaseBy = 1)

# 3- assess test labels (but one should first identify which cluster
# corresponds to the associated label)

clusterLabels = rUF.unsupervised2$unsupervisedModel$cluster

if (names(which.max(clusterLabels[which(Y.test == 0)])) == 2)
{
  clusterLabelsTemp = clusterLabels
  clusterLabelsTemp[clusterLabels == 1] = 2
  clusterLabelsTemp[clusterLabels == 2] = 1
  clusterLabels = clusterLabelsTemp
}
clusterLabels = clusterLabels - 1

# 4- assess the results

table(clusterLabels, Y.test)

# displays, for the run we do
            Y.test
clusterLabels   0   1
            0 444   6
            1 159 391
```

The unsupervised case easily identify the test labels with a test error of 16.5%. The main point is that we no longer have to wonder if the *i.i.d.* case holds or not. The only point is the *separation level* we can get when going toward clustering. We can see that,in Random Uniform Forests, by two ways.
i) call the model :

```
rUF.unsupervised2

# displays

Average variance between clusters (in percent of total variance): 77.98%
Average silhouette: 0.7113
Clusters size:
  1   2
450 550
Clusters centers (in the MDS coordinates):
     [,1]    [,2]
1 -0.0685  0.0030
2  0.0561 -0.0025
```

We get *inter-classes variance in percent of total variance*, i.e. the ratio of average distance between clusters and total variance. The more it gets high, the more we have a high separation level. We also get the *average Silhouette coefficient*, between -1 and 1. It was described by Rousseeuw (1986) and provides a good measure to analyze clusters, especially when one wants to modify or merge clusters.

In the unsupervised case, Random Uniform Forests use a multiple-layer engine, beginning by the Random Uniform Forests algorithm itself, then a proximity matrix, Multidimensional scaling that leads to strongly reduce dimension as low to 2, and ending by either kmeans or hierarchical clustering. Providing the number of clusters is not required.

ii) In almost all cases, one can easily see the clusters, by a simple plot.

```
plot(rUF.unsupervised2)
# which displays
```

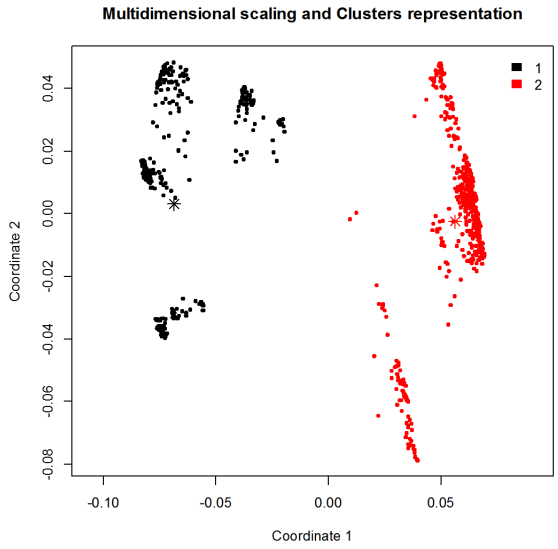

Figure 11: Clusters representation for the test set of Example 3

Since the rule to define test labels is simple, one may ask why no simply use kmeans or others clustering algorithms, which are faster. Hence, we tried *kmeans*, *clara* (R package *cluster* and *Mclust* (R package *mclust*) which uses the EM algorithm.

```
# K-Means
kmeansOnXtest = kmeans(X.test, 2)

# assess test labels: check first if clusters are correctly assigned to test labels

clusterLabels = kmeansOnXtest$cluster
table(clusterLabels[which(Y.test == 0)])
table(clusterLabels[which(Y.test == 1)])

# cluster labels need to be inverted :
clusterLabelsTemp = clusterLabels
clusterLabelsTemp[clusterLabels == 1] = 2
clusterLabelsTemp[clusterLabels == 2] = 1
clusterLabels = clusterLabelsTemp
clusterLabels = clusterLabels - 1

table(clusterLabels, Y.test)
# displays
             Y.test
clusterLabels   0   1
            0 294   6
            1 309 391
```

```
# clara:  Clustering Large Applications
# first install package, if not

# install.packages("cluster")
library(cluster)
claraOnXtest = clara(X.test, 2)

# one can easily plot clara object, getting also statistics
# plot(claraOnXtest)

# assess test labels
clusterLabels = claraOnXtest$clustering

table(clusterLabels[which(Y.test == 0)])
table(clusterLabels[which(Y.test == 1)])

# cluster labels need to be inverted :

clusterLabelsTemp = clusterLabels
clusterLabelsTemp[clusterLabels == 1] = 2
clusterLabelsTemp[clusterLabels == 2] = 1
clusterLabels = clusterLabelsTemp
clusterLabels = clusterLabels - 1

table(clusterLabels, Y.test)

# displays
             Y.test
clusterLabels   0   1
            0 294   6
            1 309 391

# Mclust:  Model-Based Clustering
# first install package, if not

mclustOnXtest = Mclust(X.test, 2)
clusterLabels = mclustOnXtest$classification

# here there is no need to assign again cluster labels

clusterLabels = clusterLabels - 1
table(clusterLabels, Y.test)
            Y.test
clusterLabels   0   1
            0 444   6
            1 159 391
```

Only, *Mclust* reaches the expected results. But in all cases, unsupervised learning, using clustering, outperforms supervised learning whenever the *non-i.i.d.* case is rising in the data, in our examples. The supervised case fails because of the shifting distribution, even for a simple rule to define test labels. If one can have unsupervised methods that can find complex patterns, learning the test sample itself, it might bring more efficiency to supervised learning.

We only have discussed about he classification case. The regression one is more challenging and, in Random Uniform Forests, involves others mechanisms.

6 Conclusion

In this paper, we have tried to show main arguments and functions of Random Uniform Forests. From the theoretical side, we describe the key differences with Random Forests, which are the use of more randomness (*the continuous Uniform distribution*) for the choice of cut-points at each node and the use of sampling, *with replacement*, the subset of candidate variables at each node. The *optimization criterion* is also different. Hence the choice of optimal node is no longer the same as in Breiman's procedure, leading to more uncorrelated trees in Random Uniform Forests (with, however, an average variance of trees that increases). Lowering correlation is essential since it is one of the key for convergence and for some other results. Since Random Uniform Forests inherit of all theoretical properties of Breiman's Random Forests, they achieve similar results at a lower cost.

From the practical side, Random Uniform Forests can be understood as an extension of the original method, designed to achieve a deeper assessment of variable importance, to be natively incremental and to allow easy distributed computing. We showed many tools to assess variable importance, providing ways to see and understand how influential variables matter. We did not talk about dome other tools like imputation of missing values or extrapolation. But we provided full working examples of cost sensitive learning, showed how comprehensive model can be built using visualization (ROC or Precision-Recall curves) and a full case of incremental learning. Following Breiman's ideas, Random Uniform Forest are aimed to be a highly versatile tool for classification, regression and unsupervised learning.

References

Biau, G., 2012. Analysis of a Random Forests Model. *The Journal of Machine Learning Research* 13, 1063-1095.

Biau, G., Devroye, L., Lugosi, G., 2008. Consistency of random forests and other averaging classifiers. *The Journal of Machine Learning Research* 9, 2015-2033.

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C., 1984. *Classification and Regression Trees*. New York: Chapman and Hall.

Breiman, L., 1996. Bagging predictors. *Machine learning* 24, 123-140.

Breiman, L., 1996. Heuristics of instability and stabilization in model selection. *The annals of statistics* 24, 2350-2383

Breiman, L., 1999. Pasting Small Votes for Classification in Large Databases and On-Line. *Machine Learning* 36, 85-103.

Breiman, L., 2001. Random forests. *Machine learning* 45, 5-32.

Breiman, L., 2001. Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statistical Science* 16, 199-231

Chen, C., Liaw, A., Breiman, L., 2004. *Using Random Forest to Learn Imbalanced Data*, Statistics Technical Reports. Statistics Department, University of California, Berkeley, University of California at Berkeley, Berkeley, California.

Ciss, S., 2014. *randomUniformForest: random Uniform Forests for Classification and Regression*. R package version 1.0.8, http://CRAN.R-project.org/package=randomUniformForest.

Ciss, S., 2014. *Forêts uniformément aléatoires et détection des irrégularités aux cotisations sociales* (in French). PhD thesis, 2014. Université Paris Ouest Nanterre.

Devroye, L., Györfi, L., Lugosi, G., 1996. *A probabilistic theory of pattern recognition*. New York: Springer.

Fawcett, T., 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27, 861-874.

Friedman, J.H., 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 29, 1189-1232.

Friedman, J.H., 2002. Stochastic gradient boosting. *Computational Statistics and Data Analysis* 38, 367-378.

Geurts, P., Ernst, D., Wehenkel, L., 2006. Extremely randomized trees. *Machine Learning* 63, 3-42.

Grömping, U., 2009. Variable Importance Assessment in Regression: Linear Regression versus Random Forest. *The American Statistician* 63, 308-319.

Hastie, T., Tibshirani, R., Friedman, J.J.H., 2001. *The elements of statistical learning*. New York: Springer.

Ho, T.K., 1998. The random subspace method for constructing decision forests.*IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 832-844.

Liaw, A., Wiener, M., 2002. Classification and Regression by randomForest. *R News* 2(3), 18–22.

Lin, Y., Jeon, Y., 2002. Random Forests and Adaptive Nearest Neighbors. *Journal of the American Statistical Association* 101-474.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., Leisch, F., 2014. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-3. http://CRAN.R-project.org/package=e1071

Oza, N.C., 2005. *Online bagging and boosting*, in: 2005 IEEE International Conference on Systems, Man and Cybernetics, pp. 2340-2345 Vol. 3.

R Core Team (2014). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Ridgeway, G., with contributions from others, 2013. *gbm: Generalized Boosted Regression Models.* R package version 2.1. http://CRAN.R-project.org/package=gbm

Rousseeuw, P. J., 1987. Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Computational and Applied Mathematics* 20, 5365.

Scornet, E., Biau, G., Vert, J. P., 2014. Consistency of Random Forests. *arXiv* preprint arXiv:1405.2881.

Shannon, C.E., 1949. *The Mathematical Theory of Communication.* University of Illinois Press.

Simm, J., de Abril, I.M., 2013. *extraTrees: ExtraTrees method.* R package version 0.4-5. http://CRAN.R-project.org/package=extraTrees

Vapnik, V.N., 1995. *The nature of statistical learning theory.* Springer-Verlag New York.