Appendices

.1 Complexity

Let us consider p predictor variables $(x_1, \cdots, x_p) \in D \subset \mathbb{R}^p$ of a response variable y. The goal of this subsection is to determine the time complexity of the Random Forest model, given N samples data.
Let $T(N)$ denote the time for building one decision tree.

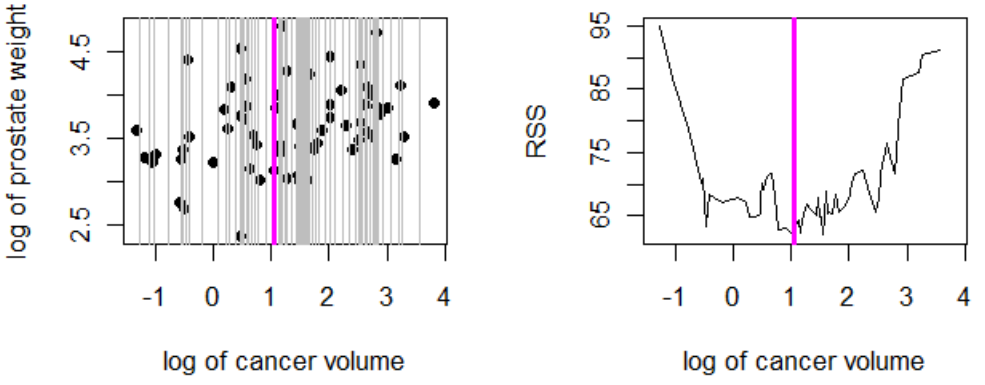

Figure 1: Choosing the best vertical split using RSS measure

$T(N)$ can be approximated recursively. Indeed, we can grow a decision tree in three steps : find the best splitting, grow a left child tree, and grow a right child tree.
Let us denote the time complexity of these steps by $s(N)$, $T_{left}(N)$ and $T_{right}(N)$ respectively. Hence : $T(N) = s(N) + T_{left}(N) + T_{right}(N)$. We will present both best case and worst case complexity for every step, and we will assume the time complexity to lay in between.

splitting and partitioning The partitioning step requires at least an iteration over the N samples, so we can already set a linear lower bound on the time complexity within a node. Finding a split can be done using Figure 1 (example from [**?**]) method, i.e computing a Residual Sum of Squares (RSS) and minimize it. This operation requires to sort the N samples for every predictor $x_i, i \in \{1, \cdots, p\}$. Time complexity to sort a list with size N is at worst $\mathcal{O}(N)$, and at best $\mathcal{O}(\log N)$. Combining both splitting and partitioning, time complexity is at worst $\mathcal{O}(pN^2)$ and at best $\mathcal{O}(pN \log N)$. Random Forest can limit the search over $K \leq p$ features, which reduces this complexity between $\mathcal{O}(KN \log N)$ and $\mathcal{O}(KN^2)$.

Child left and right trees The best case for growing these two child trees is intuitively $\frac{N}{2}$. The worst case would be having a tree with one pure leaf and $N - 1$ samples in the other tree.

Hence,

$$s(N) + T_{left}(\frac{N}{2}) + T_{right}(\frac{N}{2}) \leq T(N) \leq s(N) + T_{left}(1) + T_{right}(N-1)$$

Or

$$s(N) + 2.T(\frac{N}{2}) \leq T(N) \leq s(N) + T(1) + T(N-1)$$

Having a recurrence equation, let us make use of the following theorem.

Theorem .1. *Master Theorem*

$$T(n) = \begin{cases} c & \text{if } n < d, \\ aT(n/b) + f(n) & \text{if } n \geq d, \end{cases}$$

where $a \geq 1, b > 1$, and d are integers and c is a positive constant. Let $\nu = \log_b a$.

Case (i) $f(n)$ **is "definitely smaller" than** n^ν**:** *If there is a small contant $\epsilon > 0$, such that $f(n) \preceq n^{\nu-\epsilon}$, that is, $f(n) \prec n^\nu$, then $T(n) \sim n^\nu$.*

Case (ii) $f(n)$ **is "similar in size" to** n^ν**:** *If there is a constant $k \geq 0$, such that $f(n) \sim n^\nu (\log n)^k$, then $T(n) \sim n^\nu (\log n)^{k+1}$.*

Case (iii) $f(n)$ **is "definitely larger" than** n^ν**:** *If there are small constants $\epsilon > 0$ and $\delta < 1$, such that $f(n) \succeq n^{\nu+\epsilon}$ and $af(n/b) \leq \delta f(n)$, for $n \geq d$, then $T(n) \sim f(n)$.*

Lemma .2. *For $\frac{s(N)}{K} = \mathcal{O}(N \log N)$ and $T(N) = 2.T(\frac{N}{2}) + s(N)$, time complexity for building a decision tree is $T(N) \sim K.N. \log^2 N$.*

Proof. Apply Theorem .1 with $a = b = 2$, $d = 1$, $k = 1$ and $T(N) \equiv \frac{T(N)}{K}$ □

Lemma .3. *For $\frac{s(N)}{K} = \mathcal{O}(N \log N)$ and $T(N) = s(N) + T(1) + T(N-1)$, time complexity for building a decision tree is $T(N) \sim K.N. \log^2 N$ (upper bound) and $T(N)$.*

Proof. Let us rewrite in a first time the expression of $T(N)$ in a better form to make use of Theorem .1.

$$\frac{s(N)}{K} = \mathcal{O}(N \log N)$$

Let us assume there exists C_1, C_2 such that $C_1 N \log N \leq \frac{s(N)}{K} \leq C_2 N \log N$. Hence, using the right side,

$$\begin{aligned} \frac{T(N)}{K} &= \frac{s(N)}{K} + \frac{T(1)}{K} + \frac{T(N-1)}{K} \\ &\leq C_2 N \log N + \frac{T(1)}{K} + \frac{T(N-1)}{K} \end{aligned}$$

Let $t(N) = \frac{T(N)}{K}$.

$$
\begin{aligned}
& t(N) \leq C_2 N \log N + t(1) + t(N-1) \\
\iff & t(N) - t(N-1) \leq C_2 N \log N + t(1) \\
\Rightarrow & \sum_{j=2}^{N} t(j) - t(j-1) \leq (N-1)t(1) + C_2 \sum_{j=2}^{N} j \log j \\
\Rightarrow & t(N) \leq Nt(1) + C_2 \log N \sum_{j=1}^{N} j \\
\Rightarrow & t(N) \leq Nt(1) + C_2 \log N \frac{N(N+1)}{2} \\
\Rightarrow & t(N) = \mathcal{O}(N^2 \log N) \\
\Rightarrow & T(N) = \mathcal{O}(KN^2 \log N)
\end{aligned}
$$

Similarly, using the left side gives a lower bound

$$
\begin{aligned}
& t(N) \geq C_1 N \log N + t(1) + t(N-1) \\
\iff & t(N) - t(N-1) \geq C_1 N \log N + t(1) \\
\Rightarrow & \sum_{j=2}^{N} t(j) - t(j-1) \geq (N-1)t(1) + C_1 \sum_{j=2}^{N} j \log j \\
\Rightarrow & t(N) \geq Nt(1) + C_1 \sum_{j=2}^{N} j \\
\Rightarrow & t(N) \geq Nt(1) + C_1 (\frac{N(N+1)}{2} - 1) \\
\Rightarrow & t(N) \succeq N^2 \\
\Rightarrow & T(N) \succeq N^2
\end{aligned}
$$

□

Finally, growing n_{trees} random trees has in the best case a time complexity $\Theta(n_{trees} K N \log^2 N)$ and in the worst case $\mathcal{O}(n_{trees} K N^2 \log N)$.

Prediction time complexity Iterating over every tree generated by the Random Forest, predicting depends on the depth of the tree. Similarly to previous analysis, we can lower bound and upper bound the depth of one tree. Indeed, let $D(N)$ denote the depth of a tree generated in the Forest.

- Best case is again a $\frac{N_{at\_node\_i}}{2}$ split at every node i, which gives the following recurrence equation for prediction:

$$
\begin{cases}
D(1) = 1 \\
D(N) = 1 + D(\frac{N}{2}) + D(\frac{N}{2}) = 1 + 2.D(\frac{N}{2})
\end{cases} \tag{1}
$$

A quick application of Theorem .1 with $a = b = 2$ and $k = 0$ gives $D(N) \sim \log N$

- Worst case is a split with one simple leaf and a tree with $N - 1$ samples, which gives the following recurrence equation :

$$\begin{cases} D(1) = 1 \\ D(N) = 1 + D(1) + D(N-1) \end{cases}$$

$$\begin{aligned} D(N) - D(N-1) &= 1 + D(1) \\ D(N) &= (N+1)D(1) \\ D(N) &= \mathcal{O}(N) \end{aligned}$$