Chapter 4

INFFORESTS

The INFFOREST variable importance is a method of permuted variable importance not unlike that of conditionally permuted variable importance (Algorthim 3). Permuted variable importance is calculated at the tree level, using the partitions on X_j from a tree created to predict the model $X_j \sim X_1, ..., X_{j-1}, X_{j+1}, ..., X_p$. This auxiliary tree is fit by considering all $p-1$ predictors at each split and so may be quite large or quite small depending on the richness of the correlation structure around X_j. The auxiliary tree is also fit using the OOB sample for the tree at question. If the auxiliary tree results in a single leaf, i.e. there are no splits, then X_j is permuted blindly, without partitions. If the auxiliary tree results in two leaves, there will be two partitions on X_j to permute X_j within, and so on. After permuting X_j within these partitions, the RSS is calculated for that tree using the OOB sample of predictors, including the now-permuted X_j. The absolute difference of the RSS after permutation and the RSS with the untouched OOB sample is INFFOREST variable importance for that tree. Note that for this reason, the INFFOREST variable importance is always greater than or equal to zero, and is standardized by the max INFFOREST variable importance value given by that tree. As the variable importance values are calculated for each tree for each variable, once the method is completed there is a distribution of potential variable importance values for X_j, one for each tree. These distributions may or may not be normal, depending on the multicollinearity of the predictors. The INFFOREST variable importance algorithm works as follows:

Algorithm 1 INFForests, $VI_{inf}(R)$

1: Fit a random forest, R on the dataset D fitting the model $Y \sim X_1, ..., X_p$.
2: **for** each $X_i \in X_1, ..., X_p$ **do**
3: **for** each $t \in R$ **do**
4: Calculate: $\Xi_o = \frac{1}{\nu_t} RSS(t, \bar{B}^t)$
5: Calculate a tree T_i that predicts $X_i \sim X_1, ..., X_{i-1}, X_{i+1}, ...X_p$ using the subset of the observations used to fit t
6: Permute the subset of X_i contained in $\bar{B}_t$ with respect to the set of partitions P_{xi} from T_i.
7: Now find $\Xi^* = \frac{1}{\nu_t} RSS(t, \bar{B}_t^*)$
8: The difference between these values, $\Xi^* - \Xi_o$, is the variable importance for X_i on t
9: **end for**
10: Test the null hypothesis that 0 is the likely value of $\frac{1}{\nu_t} RSS(t, \bar{B}_t^*)$ using the distribution of values of Ξ^* gathered from each tree in R
11: **end for**

INFFOREST variable importance operates under the null hypothesis that Y is independent of X_J given the correlation structure of X_j and the other $p-1$ predictors, or that the true INFFOREST variable importance for X_j is 0. The alternative hypothesis is that Y and X_j are not independent given the correlation structure of X_j and the other predictors or that the INFFOREST variable importance for X_j is greater than zero. After INFFOREST values have been computed for the entire forest, they are treated as samples from the population of possible INFFOREST values for X_j given the random forest R_f, and a significance test can be under the null hypothesis.

Implementation In `INFTREES` and Results

Notes on the Implemetation

Implementing the `INFFOREST` and therefor the `INFTREES` algorithms, required creating a suite of functions to create trees and random forests. The trees are fit following the standard two-part CART-like algorithm. [^1] The function chooses a variable to split on with linear correlation with respect to Y, but instead of looking for correlations above a certain threshold which is common, it chooses the variable with the highest

correlation when compared to its peers. This alleviates the situation where a variable with a non-linear relationship would be passed over again and again. The splitting is done via minimization of the following function with respect to i:

$$RSS_{node}(i, X, Y) = RSS_{leaf}(Y|X < i) + RSS_{leaf}(Y|X \geq i)$$
$$RSS_{leaf} = \sum (y - \hat{y})^2$$
$$\hat{Y} : \hat{y} \in \hat{Y} : \hat{y} = E(Y),\ where\ |\hat{Y}| = |Y|$$

This function considers the regression case only, and only numeric predictors. Leafs are created when the resultant split would be unsatisfactory, i.e. at least one daughter node would have five members or less. This generates very large trees - a quality that is not an issue in random forests but may be problematic in a stand-alone setting. At this time, there is also no function to prune the trees.

Table 2: A Home-Grown Tree on $Y\ X_1 + X_2 + X_3 + X_4$

var	n	dev	ypred	split.cutleft
X2	50	5958.56398616138	-3.32099458459633	1.89262782336418
leaf	14	1683.79172385909	15.2166924465285	0
X1	36	1771.9972397028	-10.5300950967004	-0.758420438327835
X1	27	1061.15524312941	-5.71617332730625	-0.251307488862014
leaf	17	720.083908357861	-2.51280971153014	0
leaf	10	341.071334771552	-11.1618914741256	0
leaf	9	239.837381494473	-24.971860404883	0

The tree output is read in the following way: each row corresponds to a node of the tree which considers `n` observations. The mean of the Y values included in the node are `ypred`. If there is an optimal and allowable split, [^2] then the chosen variable, `var`, and the RSS_{node}, `dev`, are recorded.[1] The value of the variable in question that acts as the split point is recorded as `split.cutleft`. If there is no split on the node in question, then `var` will be recorded as `<leaf>` and the `dev` value will be the value of RSS_{leaf} at this node.

The tree output is read roughly from top to bottom, with a coda in the middle. The first row corresponds to the first node, or the node that includes the entire dataset. The second row is the beginning of the right subtree or the right daughter of first node. This pattern continues, favoring the right daughter, until a leaf is reached. The left daughter of the first node is found after all of the splits off of the right daughter have finished but is easily identified as the row with a value of `n` that is exactly the difference between the `n` values of the first two rows. In the case where the right daughter contained many more observations of the original dataset, there may be a node within the right subtree that contains the same number of observations as the left daughter of the first node. In this case, the left daughter is simply the second row with this property. The pattern of following the right daughter until a leaf is reached continues with the left subtree.

The INFTREE function follows the algorithm referenced earlier. The partitions on X_j are generated by fitting a tree, T, to the model $X_j \sim X_1, ..., X_{j-1}, X_{j+1}, ..X_p$ and calculating the predictions $T(X_1, ..., X_{j-1}, X_{j+1}, ..X_p)$. Then permuting X_j with respect to the partitions on X_j given by those predictions. For example, if $x_j \in X_j$ and the value of $T(x_1, ..., x_{j-1}, x_{j+1}, ..x_p)$ corresponding to x_j is α, x_j is permuted along with the other values of X_j that also have $T(x_1, ..., x_{j-1}, x_{j+1}, ..x_p)$ corresponding to α.

The values of $INFFOREST(X_j)$ are scaled in the following way: since the INFFOREST function computes the INFFTREES, (or the difference in post and pre permutation RSS), values in a tree-wise manner, each tree's values are divided by the maximum value. This ensures that the values are between zero and one, and that in each tree one variable is clearly deemed the *most important*.

[1] Recall that we only allow splits to take place that split the data into two groups, each with more than five members.

Results

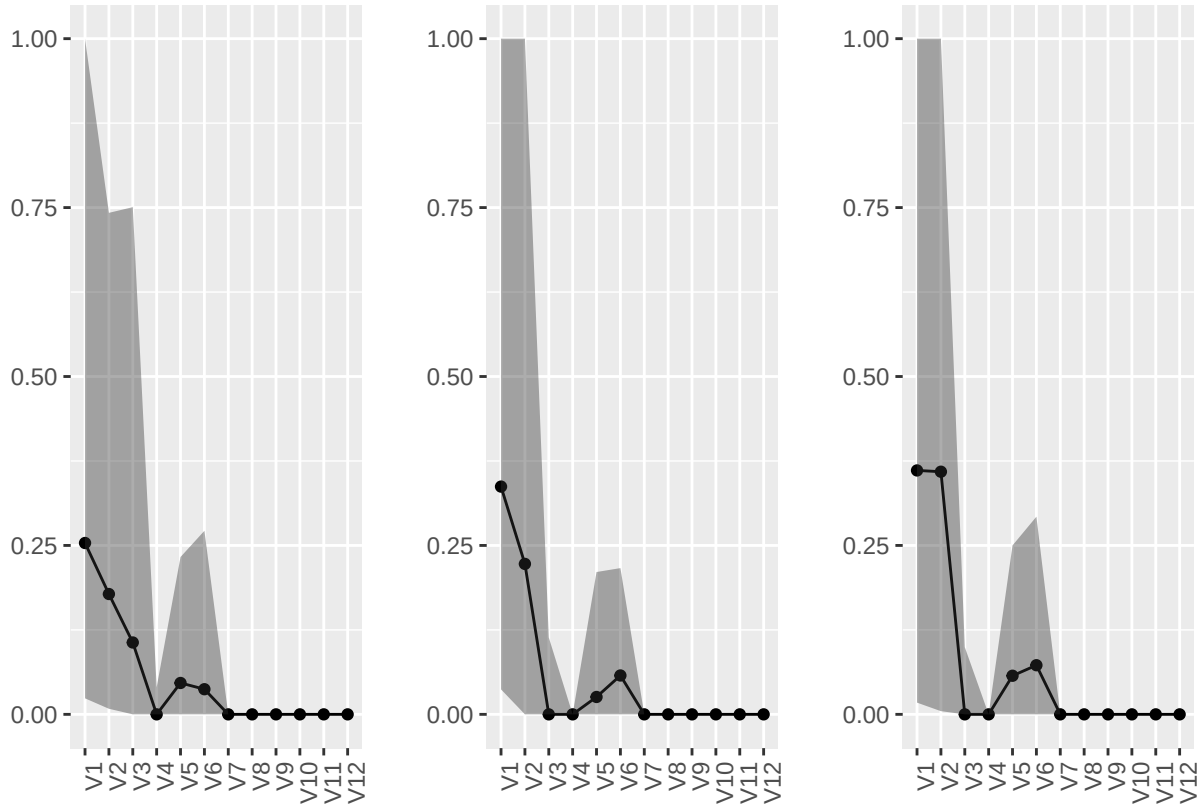

Figure 1: Median Values of INFFOREST Variable Importance for mtry = 4,6 and 8

As in (ref Strobl et al 2008), the median INFFOREST variable importance scores are reported here for the dataset D_1. [2]

As noted in several publications (Strobl et al, Breiman, Intro to Stat Learning), random forests structure is dependent on the value of `mtry`.[3] INFFOREST variable importance remains fairly consistent as `mtry` fluctuates.

[2]It's the convention to call the RSS_{node} the deviance at a node N, but, of course, this only makes sense when the node is a leaf.

[3]A great deal of effort was undertaken by the author to find the definitive, authentic CART algorithm. This implementation follows the rough strokes set out in the 1984 text *Classification and Regression Trees* to the best of the author's ability and may not be exactly the algorithm found in R packages like 'tree()'