Globally Induced Forest: A Prepruning Compression Scheme

Abstract

TODO: Abstract

1. Introduction

Decision forests, such as Random Forest (Breiman, 2001) and Extremely Randomized Trees (Geurts et al., 2006), are popular methods in the machine learning community. This popularity is due to their overall good accuracy, relative ease-of-use, short learning/prediction time and interpretability. The good accuracies are a consequence of the variance reduction through the ensemble averaging. The ease-of-use is due to the small number of well-understood hyper-parameters. In particular, the accuracy is a monotonic increasing function of the number of trees. The computational efficiency in the learning phase originates in the greedy building mechanism whereas it is a consequence of the partitioning tree structure during the prediction phase. Finally, the model is interpretable in the sense that it yields, as a by product, a measure of variable importances with respect to the goal.

Over the past decade, datasets have become bigger and bigger. The number of instances N has increased and the community has turned to very high-dimensional learning problems. The former has led to bigger trees, as the number of nodes in a tree is $O(N)$. The latter, on the other hand, tends to steer toward larger forests. Indeed, the variance of individual trees tends to increase with the dimensionality P of the problem (Joly et al., 2012). Therefore, the adequate number of trees T increases with the dimensionality. Overall, this change of focus might render tree-based ensemble techniques impractical memory-wise, as the total footprint is $O(N \times T(P))$.

Aside from big data, other areas of machine learning suffer from the high memory demand of tree-based methods. For instance, low-memory devices, such as mobile phones and embedded systems, require lightweight models. In addition, bigger is not always better when it comes to tree-based ensembles. If it true that the more trees, the better the model, this is not the case for deeper individual trees. Smaller models also implies faster predictions, dear to real-time application.

All in all, tree-based models might benefit from lighter memory footprint in many different ways.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Related work

Memory constraints of tree-based ensemble methods is not a new topic and has been tackled from various perspectives, which can be partitioned into tree-agnostic and tree-aware methods.

The former set of techniques are general purpose methods which can deal with any ensembles. We can distinguish further between re-learning algorithms (*e.g.* Domingos (1997), Menke & Martinez (2009)), which try to come up with a smaller, equivalent models, and ensemble pruning methods. Those try to eliminate some of the redundant base learners constituting the ensemble (for a review of those methods, please consult Tsoumakas et al. (2008), Rokach (2016)).

Tree-aware methods strive to build smaller trees by limiting the total number of nodes within the forest. They either work with a subsample of the training set (*e.g.* Breiman (1999)), or on the whole learning set. The latter can be partitioned into pre- and post-pruning families. Pre-pruning methods aims at stopping the development of uninteresting branches in the top down induction procedure. On the other hand, the post-pruning methods's goal is to discard *a posteriori* branches which do not provide significant accuracy improvements.

Originally, the idea was introduced to control the model complexity and avoid overfitting. The advent of ensemble methods somewhat cast aside those techniques as the averaging mechanism became responsible for reducing the learning algorithm's variance. Nonetheless, a few ensemble-wise, post-pruning methods have recently emerged with a focus on memory minimization. In both Meinshausen et al. (2009) and Joly et al. (2012), the compression is formulated as a slightly different global constrained optimization problem. In opposition, compression is undertook by a sequential optimization problem in Ren et al. (2015). In De Vleeschouwer et al. (2015), the authors alleviate the leaves memory requirements by clustering their conditional distributions. After computing a

wavelet coefficient for each node, the authors of Elisha & Dekel (2016) discard all the node's which are not on the path to a node of sufficient coefficient. All these methods are able to retain almost the full forest accuracies while offering a significant memory improvement, leaving their requirement of building the whole forest first, and consequently the high memory and computational costs, as their only major drawbacks.

Contribution and outline In this paper, we propose the Globally Induced Forest (GIF), an algorithm which, under a node budget constraint, iteratively and greedily deepens the trees by optimizing globally the sequence of nodes to develop and its associated weight, while still choosing locally, based on the standard score criterion, the splitting variables and cut points at all tree nodes.

This mix of global, local and greedy optimization results in a fast pre-pruning approach to build lightweight, yet accurate forests, learned on the whole training set. Contrary to post-pruning approaches, GIFs alleviate the need to build the whole forest first, resulting in a fast learning algorithm and discarding the need for a large temporary storage.

We show that TODO

TODO outline

3. Globally Induced Forest

GIFs rely on the view of the forest as a linear model in the "forest space", a binary M-dimensional space, where M is the total number of nodes in the whole forest (Joly et al., 2012):

$$\hat{y}(x) = \frac{1}{T} \sum_{j=1}^{M} w_j z_j(x), \tag{1}$$

where the indicator function $z_j(x)$ is 1 if x reaches node j and 0 otherwise, and w_j is the prediction at a node j if j is a leaf and 0 otherwise. In regression, $w_j \in \mathbb{R}$ would be the average value of the subset of outputs reaching node j. In classification, $w_j \in \mathbb{R}^K$ is a vector of dimension K, where $w_j^{(k)}$ $(k = 1, \dots, K)$ is the probability associated to class k.

GIF's learning phase is described in Algorithm 1. Starting from a constant model (step 5), it builds an additive model in the form of Equation (1) by incrementally adding new node indicator functions in a stagewise fashion in order to grow the forest. At each step, a subset of candidate nodes C_t is built uniformly at random from the total candidate list C (step 8). The node j^* among those of C_t which contributes the most to a decrease of the global loss is selected (step 9) and introduced in the model via its indicator function z_{j^*} and its optimal weight w_j^* tempered by some learning rate λ (step 10). This node is then split locally according to the reference tree growing strategy $\mathcal{A}$ (step 11) and replaced by its two children in the candidate list (step 12). The process is stopped when the node budget B is reached.

Algorithm 1 Globally Induced Forest

1: **Input:** $D = (x_i, y_i)_{i=1}^N$, the learning set; $\mathcal{A}$, the tree learning algorithm; L, the loss function; B, the node budget; T, the number of trees; CW, the candidate window size; λ, the learning rate.
2: **Output:** An ensemble S of B tree nodes with their corresponding weights.
3: **Algorithm:**
4: $S = \emptyset$; $C = \emptyset$; $t = 1$
5: $\hat{y}^{(0)}(.) = \arg\min_{y \in \mathbb{R}^K} \sum_{i=1}^N L(y_i, y)$
6: Grow T stumps with $\mathcal{A}$ on D and add the left and right successors of all stumps to C.
7: **repeat**
8: C_t is a subset of size $\min\{CW, |C|\}$ of C chosen uniformly at random.
9: Compute:
$$(j^*, w_j^*) = \underset{j \in C_t, w \in \mathbb{R}^K}{\arg\min} \sum_{i=1}^{N} L\left(y_i, \hat{y}^{(t-1)}(x_i) + w z_j(x_i)\right)$$
10: $S = S \cup \{(j^*, w_j^*)\}$; $C = C \setminus \{j^*\}$;
$y^{(t)}(.) = y^{(t-1)}(.) + \lambda w_{j^*}^* z_{j^*}(.)$
11: Split j^* using $\mathcal{A}$ to obtain children j_l and j_r
12: $C = C \cup \{j_l, j_r\}$; $t++$
13: **until** budget B is met

The predictions follows from Equation (1) with the slight difference that internal nodes now have a weight as well. This could trivially be gotten rid of by pushing all the weights to the leaves.

Note that GIF is very similar to a least-square boosting algorithm (Friedman et al., 2001), where the set of base learners would be composed of node indicator functions and would be expanded at each iteration.

Node selection and weight optimization Step 9 of Algorithm 1 can be decomposed into two parts. Firstly, the optimal weight for a given candidate node is computed. Then, the optimal node — the one which reduces the loss the most — is selected with exhaustive search. Also note that computing the error can be done efficiently by going only over the instances reaching node j^*:

$$w_j^{(t)} = \arg\min_{w \in \mathbb{R}^K} \sum_{i=1}^{N} L\left(y_i, \hat{y}^{(t-1)}(x_i) + w z_j(x_i)\right) \quad (2)$$

$$\text{err}_j^{(t)} = \sum_{i=1}^{N} L\left(y_i, \hat{y}^{(t-1)}(x_i) + w_j^{(t)} z_j(x_i)\right) \quad (3)$$

$$j_t^* = \arg\min_{j \in C_t} \text{err}_j^{(t)} = \arg\max_{j \in C_t} \text{err}^{(t-1)} - \text{err}_j^{(t)} \quad (4)$$

$$\text{gap}_j^{(t)} = \text{err}^{(t-1)} - \text{err}_j^{(t)} \quad (5)$$

$$= \sum_{i=1}^{N} \left[L\left(y_i, \hat{y}^{(t-1)}(x_i)\right) - L\left(y_i, \hat{y}^{(t)}(x_i)\right)\right] \quad (6)$$

$$= \sum_{i \in Z_j} \left[L\left(y_i, \hat{y}^{(t-1)}(x_i)\right) - L\left(y_i, \hat{y}^{(t)}(x_i)\right)\right] \quad (7)$$

since $\hat{y}^{(t-1)}(x_i) \neq \hat{y}^{(t)}(x_i)$ only for the instances reaching node j: $i \in Z_j$. Due to the partitioning property of the tree, it means that, at each iteration, computing the optimal weights for all the nodes of a given tree is at most $O(N)$, assuming a single optimization runs in linear time in the number of instances.

Tree learning algorithm The tree learning algorithm is responsible for splitting the data reaching a node into two partitions. This choice is made locally, meaning that the algorithm only considers the node it is processing and disregard the other trees. In order to encourage diversity among candidates, we have chosen to use the Extremely randomized trees's splitting rule ((Geurts et al., 2006)): m out of p features are selected uniformly at random and for each feature, a cut point is chosen uniformly at random between the current minimum and maximum value of this feature; the final decision function is the one which reduces the impurity score—variance in regression, gini index in classification—the most.

Node budget The node budget accounts for the total number of nodes in the resulting forest. That is, both internal (splitting) and external (decision) nodes. The root nodes are only accounted for when one of its children is taken into the model.

Forest's shape Three parameters interact to influence the forest's shape: the number of trees T, the candidate window size CW and the learning rate λ.

On the one hand, $CW = 1$ means that the forest's shape is predetermined and solely governed by the number of trees. Few trees imposes the development in depth, while many trees encourages in breadth development.

On the other hand, $CW = +\infty$ means that the algorithm takes the time to optimize completely the node it chooses, giving it full rein to adapt the forest's shape to the problem at hand. In that case, the learning rate plays an important role (Figure 2). If it is low, the node will not be well optimized and the learning algorithm will look for similar nodes. In opposition, if the learning rate is high, the node will be well optimized and the algorithm will turn to different nodes. As similar nodes tend to be located roughly at the same place in trees, low (resp. high) learning rate will encourage in breadth (resp. in depth) development.

Regularization The learning rate is the most explicit regularization parameter. However, many other factors act as implicit regularization in GIF, allowing at the same time for faster learning. Namely, the local splitting done by the tree learning algorithm $\mathcal{A}$, the greedy, stagewise weight optimization and the candidate subsampling.

3.1. Regression

Under the $L2$-norm, the optimization (2) becomes:

$$w_j^{(t)} = \arg\min_{w \in \mathbb{R}} \sum_{i \in Z_j} \left(r_i^{(t-1)} - w\right)^2 \quad (8)$$

where $r_i^{(t-1)} = y_i - \hat{y}^{(t-1)}(x_i)$ is the residual at time $t-1$. The optimal weight is the average residual:

$$w_j^{(t)} = \frac{1}{|Z_j|} \sum_{i \in Z_j} r_i^{(t-1)} \quad (9)$$

In the case of a unit learning rate and a single tree, the model's prediction coincide with the ones the underlying tree would provide (see Appendix A).

Extending to the multi-output case is straightforward: one only needs to fit a weight independently for each output. The loss becomes the sum of the individual losses over each output.

3.2. Classification

Binary classification can either be tackled with the square loss, recasting the classes as $\{-1, +1\}$, or by employing a more suited loss function. Indeed, the former has the disadvantage that it will penalize correct classification if the prediction overshoots the real value.

In multiclass classification, one has several options. A first possibility is to build several binary classification models using a binary loss function. Interestingly, this can be

done in a single learning phase by attributing one output per model. In contrast with a pure one-versus-one or one-versus-rest technique, the individual models would not be independent as they share the same forest structure.

A second approach is to employ a custom multiclass loss. An example of such a loss function is the multiclass exponential loss discussed in Zhu et al. (2009). Firstly, we must encode the class into a K-dimensional vector so that

$$y_i^{(k)} = \begin{cases} 1, & \text{if the class of } y_i \text{ is } k \\ -\frac{1}{K-1}, & \text{otherwise} \end{cases} \tag{10}$$

This representation agrees with the binary case and is less demanding than a one-versus-rest approach: the negative classes weigh the same as the correct one; $\sum_{k=1}^{K} y_i^{(k)} = 0$.

With this representation, the optimization problem (2) becomes:

$$w_j^{(t)} = \underset{w \in \mathbb{R}^K}{\arg\min} \sum_{i \in Z_j} \exp\left(\frac{-1}{K} y_i^T \left(\hat{y}^{(t-1)}(x_i) + w\right)\right) \tag{11}$$

This program is convex and verifies the following constraint at the optimum:

$$\alpha_j^{(t-1,k)} \phi^{(k)}(w) = \alpha_j^{(t-1,l)} \phi^{(l)}(w) \quad i \leq k, l \leq K \tag{12}$$

$$\alpha_j^{(t-1,k)} = \sum_{i \in Z_j^{(k)}} \exp\left(-\mu_i^{(t-1)}\right) \tag{13}$$

$$\mu_i^{(t-1)} = \frac{1}{K} \sum_{k=1}^{K} y_i \hat{y}^{(t-1,k)}(x_i) \tag{14}$$

$$\phi^{(k)}(w) = \exp\left(-\frac{1}{K} \psi^{(k)}(w)\right) \tag{15}$$

$$\psi^{(k)}(w) = -w^{(k)} + \frac{1}{K-1} \sum_{l=1, l \neq k}^{K} w^{(l)} \tag{16}$$

where $Z_j^{(k)} = \{1 \leq i \leq N | z_{i,j} = 1 \wedge y_i^{(k)} = 1\}$ is the subset of learning instances of class k reaching node j. In words, $\mu_i^{(t-1)}$ is the hyper-margin of instance i at time $t-1$ and $\alpha_j^{(t-1,k)}$ is the class error of label k for node j at time $t-1$.

In keeping with the representation (Equation 10), we can impose a zero-sum constraint on the prediction to get a unique solution for the kth component for node j at time t. If it is imposed at each stage, it means that

$$\sum_{k=1}^{K} \hat{y}^{(t-1,k)} = \sum_{k=1}^{K} \hat{y}^{(t,k)} = 0 = \sum_{k=1}^{K} w^{(k)} \tag{17}$$

and this is not impacted by the learning rate. The solution becomes

$$\phi^{(k)}(w) = \exp\left(-\frac{1}{K-1} w^{(k)}\right) \tag{18}$$

$$\alpha_j^{(t-1,k)} = \sum_{i \in Z_j^{(k)}} \exp\left(-\frac{1}{K-1} \hat{y}^{(t-1,k)}(x_i)\right) \tag{19}$$

$$w_j^{(t,k)} = \frac{K-1}{K} \log \prod_{l=1}^{K} \frac{\alpha_j^{(t-1,k)}}{\alpha_j^{(t-1,l)}} \tag{20}$$

Probabilities Posterior probabilities of an example x belonging to class k can be derived by running the additive model through a softmax:

$$P^{(t)}(k|x) = \frac{\exp\left(\frac{1}{K-1} \hat{y}^{(t,k)}(x)\right)}{\sum_{l=1}^{K} \exp\left(\frac{1}{K-1} \hat{y}^{(t,l)}(x)\right)} \tag{21}$$

In the case of a unit learning rate and a single tree, the probabilities thus derived coincide with the ones the underlying tree would provide (see Appendix A).

Trimmed exponential loss Equation (20) glosses over a crucial detail: what happens when some classes are not represented, that is the class error $\alpha_j^{(t-1,k)}$ is zero? To circumvent this problem, we propose to approximate the optimal weight in the following fashion:

$$w_j^{(t,k)} = \frac{K}{K-1} \sum_{l=1}^{K} \tau_\theta\left(\alpha_j^{(t-1,k)}, \alpha_j^{(t-1,l)}\right) \tag{22}$$

$$\tau_\theta(x_1, x_2) = \begin{cases} \theta, & \text{if } x_2 = 0 \text{ or } \frac{x_1}{x_2} > e^\theta \\ -\theta, & \text{if } x_1 = 0 \text{ or } \frac{x_2}{x_1} > e^\theta \\ \log \frac{x_1}{x_2}, & \text{otherwise} \end{cases} \tag{23}$$

The thresholding function τ_θ acts as an implicit regularization mechanism: it prevents some class errors from weighing too much in the final solution by imposing a maximum order of magnitude between the class errors. In contrast with simpler fix, the value of the θ parameter is easily apprehended and/or do not depend in some intricate way on t.

Table 1. Datasets

DATASET	LS SIZE	DIMENSIONALITY	# CLASSES
FRIEDMAN1	300	10	-
ABALONE	2506	10	-
CT SLICE	2000	385	-
HWANG F5	2000	2	-
CADATA	12384	8	-
RINGNORM	300	20	2
TWONORM	300	10	2
HASTIE	2000	10	2
MUSK2	2000	166	2
MADELON	2200	500	2
MNIST8VS9	11800	784	2
WAVEFORM	3500	40	3
VOWEL	495	10	11
MNIST	50000	784	10
LETTER	16000	8	26

4. Empirical analysis

Datasets Table 1 sums up the main characteristics of the datasets we used. The noise parameter of the Friedman1 dataset has been set to 1. Cadata stands for California data housing. Out of the 500 features of Madelon, 20 are informative and 50 are redundant; the others are noise. Mnist8vs9 is the Mnist dataset of which only the 8 and 9 digits have been kept. A binary version of the Mnist, Letter and Vowel datasets have been created as well by grouping the first half and second half classes together.

4.1. Default hyper-parameters

Our first experiment was to test the GIF against the Extremely randomized trees (ET). For this, we first computed several forests of a 1000 fully-developed ET to extract an expected node budget per dataset. We then examined how GIF compared to ET for 1% and 10% of the original budget. For GIF, we started with $T = 1000$ stumps and a learning rate of $\lambda = 10^{-1.5}$. The underlying tree building algorithm is ET. In both the classic ET and GIF, no restriction is imposed regarding the depth and $\sqrt{p}$ features are examined for each split, where p is the initial number of features. For regression, we used the square loss and a candidate window size $CW = 1$. For classification, we tested two methods. The first one is a one-vs-rest approach by allocating one output per class with the square loss. In that case, CW is also equal to 1. The second method was to use the trimmed exponential loss with a saturation $\theta = 3$. This means that the class errors imbalance is not allowed to count for more than $e^3 \approx 20$. Since there is an additional regularization at play, we allowed for a bit more optimization and set $CW = 10$. In all cases, the experiment were repeated ten times and are reported on Tables 2 and 3.

Regression As we can see from Table 2, this default set of parameters performs quite well under heavy memory constraint (*i.e.* a budget of 1%). $\text{GIF}_{1\%}$ outperforms significantly $\text{ET}_{1\%}$ four times out of five. Moreover, on those four datasets, $\text{GIF}_{1\%}$ is able to beat the original forest with only 1% of its node budget. The mild constraint case (*i.e.* a budget of 10%) is more contrasted. On Friedman1, California data housing and CT Slice, $\text{GIF}_{10\%}$ outperforms $\text{ET}_{10\%}$. For both Abalone and Hwang, $\text{GIF}_{10\%}$ overfits; in both cases the errors of $\text{GIF}_{1\%}$ were better than at 10% and, as mentioned, better than $\text{ET}_{100\%}$.

Classification Table 3 draws an interesting conclusion: the number of classes should guide the choice of loss. In the binary case, the trimmed exponential seems to work best. At 1%, it loses on Twonorm and Musk2 and the binarized version of Vowel and Letter. In all those cases, it is much closer to the winner than to the loser though. Moreover, the winner is not the same in both cases. Overall, the trimmed exponential seems to be the safest, and usually the best choice. This is even clearer at 10%, where the trimmed exponential beats the other methods on all but the binary Letter dataset.

When it comes to multiclassification, however, the trimmed exponential seems to suffer. It wins only one time: at 10% on Waveform, a three-classes dataset. In the other settings, its performances range from bad to catastrophic. The multi-output square loss version is sometimes able to outperform the ET version. This is the case of both Waveform and Mnist at 1% and of Mnist at 10%.

The binary version of Vowel, Mnist and letter indicates that GIF struggle much more with the number of classes than with the the dimensionality of the problem and/or the learning sample size.

Interestingly, GIF's performance on Madelon with both losses are better than the base ET version. This suggests that GIF is capable of handling irrelevant features quite well.

Needless to say that this default parameter setting, although performing well on average, is not optimal for all datasets. For instance, on CT slice at 1%, we can reach 20.54 ± 0.76 by enlarging the candidate window size to 10. For the trimmed exponential, on Twonorm at 1%, we can reach 3.74 ± 0.31 with $CW = 1, \lambda = 10^{-1}$ and 3.54 ± 0.3 on Musk2 at 1% for $\lambda = 10^{-1}$.

4.2. Influence of the hyper-parameters

Learning rate Figure 1 depicts a typical evolution of the error with the budget in the case of Friedman1 (the budget of 59900 corresponds to 10%). A unit learning rate will usually decrease the test set error rapidly but will then ei-

Table 2. Mean square error.

DATASET	$ET_{100\%}$	$ET_{10\%}$	$GIF_{10\%}$	$ET_{1\%}$	$GIF_{1\%}$
FRIEDMAN1	4.89 ± 0.23	5.02 ± 0.22	2.37 ± 0.24	5.87 ± 0.27	3.26 ± 0.29
ABALONE	4.83 ± 0.21	4.87 ± 0.21	5.20 ± 0.21	5.29 ± 0.27	4.74 ± 0.23
CT SLICE	19.32 ± 1.69	19.62 ± 1.69	19.31 ± 0.61	23.84 ± 1.85	36.48 ± 1.32
HWANG F5 $\times 10^{-2}$	8.20 ± 0.11	8.25 ± 0.11	8.58 ± 0.10	8.67 ± 0.12	6.91 ± 0.04
CADATA $\times 10^{-2}$	25.45 ± 0.65	25.71 ± 0.62	21.76 ± 0.66	28.39 ± 0.97	24.08 ± 0.65

Table 3. Misclassification rate (in percent). SQ relates to the classification with square loss and one output per class. TE relates to the trimmed exponential loss.

DATASET	$ET_{100\%}$	$ET_{10\%}$	$SQ_{10\%}$	$TE_{10\%}$	$ET_{1\%}$	$SQ_{1\%}$	$TE_{1\%}$
RINGNORM	2.91 ± 0.40	3.28 ± 0.41	4.05 ± 0.45	3.21 ± 0.26	7.43 ± 0.55	5.35 ± 0.65	4.31 ± 0.67
TWONORM	3.13 ± 0.13	3.54 ± 0.18	3.50 ± 0.24	3.45 ± 0.27	8.00 ± 0.57	3.91 ± 0.39	4.12 ± 0.38
HASTIE	10.30 ± 0.46	11.78 ± 0.56	10.33 ± 0.41	7.47 ± 0.30	20.38 ± 0.56	7.64 ± 0.50	6.97 ± 0.39
MUSK2	3.65 ± 0.40	3.70 ± 0.37	3.41 ± 0.34	3.21 ± 0.36	4.22 ± 0.37	7.40 ± 0.38	4.64 ± 0.31
MADELON	9.75 ± 0.75	12.43 ± 0.77	9.18 ± 0.83	8.57 ± 0.63	23.91 ± 1.17	12.55 ± 0.83	10.30 ± 0.93
MNIST8VS9	0.99 ± 0.23	1.06 ± 0.23	0.86 ± 0.24	0.70 ± 0.15	1.58 ± 0.31	2.10 ± 0.35	1.02 ± 0.16
BIN. VOWEL	1.96 ± 1.04	2.28 ± 1.20	2.81 ± 1.17	2.10 ± 0.98	4.18 ± 1.70	12.28 ± 2.00	6.57 ± 1.28
BIN. MNIST	1.92 ± 0.16	2.04 ± 0.21	1.76 ± 0.15	1.58 ± 0.16	3.37 ± 0.17	3.24 ± 0.20	2.13 ± 0.16
BIN. LETTER	1.80 ± 0.20	2.00 ± 0.17	2.44 ± 0.25	2.25 ± 0.24	3.59 ± 0.35	7.57 ± 0.38	4.14 ± 0.20
WAVEFORM	13.95 ± 0.58	14.47 ± 0.93	14.17 ± 0.62	14.05 ± 0.82	19.11 ± 0.57	13.26 ± 0.56	14.69 ± 0.74
VOWEL	5.92 ± 1.29	6.08 ± 1.13	7.31 ± 1.18	10.87 ± 1.61	11.74 ± 1.71	22.91 ± 2.03	31.76 ± 2.87
MNIST	2.63 ± 0.18	2.87 ± 0.19	2.26 ± 0.17	3.66 ± 0.31	4.94 ± 0.21	3.92 ± 0.25	5.16 ± 0.29
LETTER	2.53 ± 0.16	2.75 ± 0.17	2.82 ± 0.19	5.88 ± 0.32	5.34 ± 0.27	8.10 ± 0.55	9.90 ± 0.53

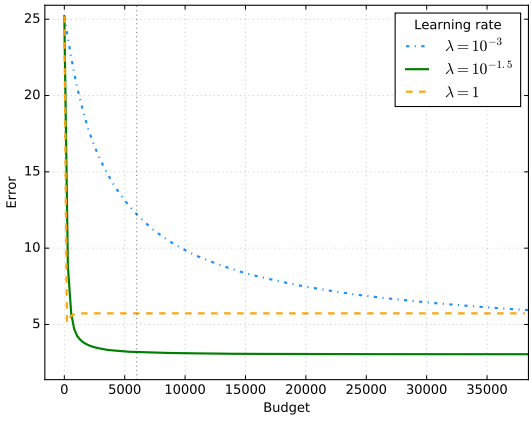

Figure 1. Friedman1: Testing set mean square error with respect to the budget ($CW = \infty$, m=$\sqrt{10}$, $T = 1000$, budget=59900 (10%)).

ther saturates or overfits. Too small a learning rate (*e.g.* 10^{-3}) will prevent the model from reaching its minimum in the alloted budget. The learning rate also influences the forest's shape, provided the candidate window size is large enough. On Figure 2, we can see that, for the smallest learning rate, the 80% of the smallest trees account for 43% of nodes. At that stage, only 17% and 13% of the nodes are covered for the average and biggest learning rates, respectively.

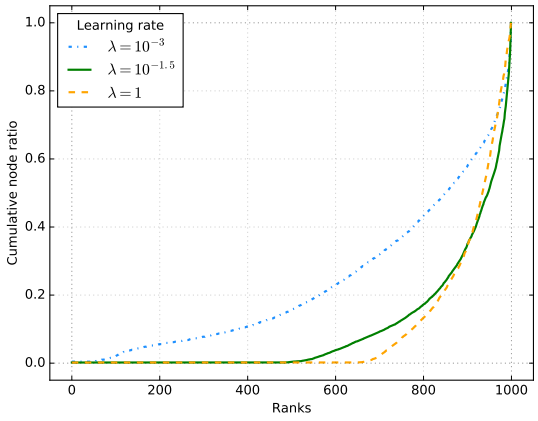

Figure 2. Friedman1: node distribution across tree ($CW = \infty$, m=$\sqrt{10}$, $T = 1000$, budget=59900 (10%).

Number of features

Candidate window size CW=1, CW=10, CW=inf Performance + time

Number of trees With CW=inf, With CW=1

4.3. Comparison with Boosting

A. Equivalence of GIF and the underlying tree

In the case of a single tree and a unit learning rate, both the square loss in regression and the multiexponential loss in classification produce the same prediction as the underlying tree. This is due to the fact that, when examining the weight to give to node j at time t, the prediction of time $t-1$ relates to the parent π_j of j. It is thus independent of t and is also the same for all instance reaching that node. Consequently, we will adopt the following slight change in notation:

$$\hat{y}_j = \hat{y}_{(\pi_j)} + w_j \tag{24}$$

Meaning that the prediction associated to any object reaching node j is the weight of j plus the prediction associated to its parent π_j. With $\hat{y}_{(\pi_1)} = 0$, the prediction of the root's pseudo-parent.

A.1. Regression

In regression, the tree prediction Tr_j of any leaf j is the average of the learning set's outputs reaching that node: $Tr_j = \frac{1}{|Z_j|}\sum_{i \in Z_j} y_i$. We need to show that the GIF prediction is:

$$\hat{y}_j = \frac{1}{|Z_j|}\sum_{i \in Z_j} y_i \tag{25}$$

The prediction of node j is

$$\begin{aligned}
\hat{y}_j &= \hat{y}_{\pi_j} + w_j && (26)\\
&= \hat{y}_{\pi_j} + \frac{1}{|Z_j|}\sum_{i \in Z_j}\left(y_i - \hat{y}_{\pi_j}\right) && (27)\\
&= \hat{y}_{\pi_j} + \frac{1}{|Z_j|}\sum_{i \in Z_j}(y_i) - \hat{y}_{\pi_j} && (28)\\
&= \frac{1}{|Z_j|}\sum_{i \in Z_j} y_i && (29)
\end{aligned}$$

The first step is how the additive model is built. The second is the optimal weight value of node j derived in Equation 9, the third step is due to the fact that the prediction at π_j is constant since there is only one tree.

A.2. Classification

In order to have the same prediction as the underlying tree, we must demonstrate that the probability of being in class l associated to node j will be $\frac{Z_j^{(l)}}{|Z_j|}$. Under the zero-sum constraint, we have

$$\begin{aligned}
\exp\left(\frac{1}{K-1}w_j^{(l)}\right) &= \frac{1}{c_j}\alpha_{\pi_i}^{(l)} && (30)\\
&= \frac{1}{c_j}\sum_{i \in Z_j^{(l)}} \exp\left(-\frac{1}{K-1}\hat{y}_{\pi_i}^{(l)}\right) && (31)\\
&= |Z_j^{(l)}| \exp\left(-\frac{1}{K-1}\hat{y}_j^{(l)}\right) && (32)
\end{aligned}$$

$$\exp\left(\frac{1}{K-1}\hat{y}_j^{(l)}\right) = \exp\left(\frac{1}{K-1}\hat{y}_{\pi_j}^{(l)}\right)\exp\left(\frac{1}{K-1}w_j^{(l)}\right) \tag{33}$$

$$\begin{aligned}
&= \frac{1}{c_j}|Z_j^{(l)}| && (34)\\
P_j(l) &= \frac{\exp\left(\frac{1}{K-1}\hat{y}_j^{(l)}\right)}{\sum_{k=1}^{K}\exp\left(\frac{1}{K-1}\hat{y}_j^{(k)}\right)} = \frac{|Z_j^{(l)}|}{|Z_j|} && (35)
\end{aligned}$$

where $c_j = \left(\prod_{k=1}^{K}\alpha_j^{(k)}\right)^{\frac{1}{K}}$ is a constant. The first equality is a consequence of the value of $w_j^{(l)}$ (Equation 20). The second is a due to the definition of $\alpha_j^{(l)}$ (Equation 19). The third is a consequence of having a single tree: the prediction of the parent is the same for all instances.

Notice that, in both regression and classification, the equivalence also holds for an internal node: the prediction is the one the tree would have yielded if that node had been a leaf.

References

Breiman, Leo. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1-2):85–103, 1999.

Breiman, Leo. Random forests. *Machine learning*, 45(1): 5–32, 2001.

De Vleeschouwer, Christophe, Legrand, Anthony, Jacques, Laurent, and Hebert, Martial. Mitigating memory requirements for random trees/ferns. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pp. 227–231. IEEE, 2015.

Domingos, Pedro. Knowledge acquisition from examples via multiple models. In *Machine learning-international workshop then conference*, pp. 98–106. Morgan Kaufmann publishers, INC., 1997.

Elisha, Oren and Dekel, Shai. Wavelet decompositions of random forests-smoothness analysis, sparse approximation and applications. *Journal of Machine Learning Research*, 17(198):1–38, 2016.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

Geurts, Pierre, Ernst, Damien, and Wehenkel, Louis. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Joly, Arnaud, Schnitzler, François, Geurts, Pierre, and Wehenkel, Louis. L1-based compression of random forest models. In *20th European Symposium on Artificial Neural Networks*, 2012.

Meinshausen, Nicolai et al. Forest garrote. *Electronic Journal of Statistics*, 3:1288–1304, 2009.

Menke, Joshua E and Martinez, Tony R. Artificial neural network reduction through oracle learning. *Intelligent Data Analysis*, 13(1):135–149, 2009.

Ren, Shaoqing, Cao, Xudong, Wei, Yichen, and Sun, Jian. Global refinement of random forest. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 723–730, 2015.

Rokach, Lior. Decision forest: Twenty years of research. *Information Fusion*, 27:111–125, 2016.

Tsoumakas, Grigorios, Partalas, Ioannis, and Vlahavas, Ioannis. A taxonomy and short review of ensemble selection. In *ECAI 2008, workshop on supervised and unsupervised ensemble methods and their applications*, pp. 41–46, 2008.

Zhu, Ji, Zou, Hui, Rosset, Saharon, and Hastie, Trevor. Multi-class adaboost. *Statistics and its Interface*, 2(3): 349–360, 2009.