Globally Induced Forest: A Prepruning Compression Scheme

Jean-Michel Begon [1] **Arnaud Joly** [1] **Pierre Geurts** [1]

Abstract

Tree-based ensemble models are heavy memory-wise. An undesired state of affairs considering nowadays datasets, memory-constrained environment and fitting/prediction times. In this paper, we propose the Globally Induced Forest (GIF) to remedy this problem. GIF is a fast prepruning approach to build lightweight ensembles by iteratively deepening the current forest. It mixes local and global optimizations to produce accurate predictions under memory constraints in reasonable time. We show that the proposed method is more than competitive with standard tree-based ensembles under corresponding constraints, and can sometimes even surpass much larger models.

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. However, datasets have become bigger and bigger over the past decade. 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. Smaller models also implies faster predictions, which is crucial for real-time applications. All in all, tree-based models might benefit from lighter memory footprint in many different ways.

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

As a pre-pruning approach, GIFs circumvent the need to build the whole forest first, thus discarding the need for a large temporary storage. This, in addition to the mix of global, local and greedy optimization, results in a fast method, able to produce lightweight, yet accurate forests learned on the whole training set.

After a discussion of the related work in Section 2, Section 3 introduces the GIF algorithm and how it can be applied for both regression (Section 3.1) and classification (Section 3.2). In Section 4, we show that our proposed algorithm, with its default setting, performs well on many datasets, sometimes even surpassing much larger models. We then conduct an extensive analysis of its hyper-parameters (Section 4.2). Since GIF shares some resemblance to Boosting, the two approaches are compared in Section 4.3, before concluding and outlining future works in Section 5.

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. These latter methods try to eliminate

[1]Department of Electrical Engineering and Computer Science University of Liège, Liège, Belgium. Correspondence to: Jean-Michel Begon <jm.begon@ulg.ac.be>, Pierre Geurts <p.geurts@ulg.ac.be>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

some of the redundant base models constituting the ensemble but do not attempt to reduce the complexity of the individual models (Tsoumakas et al., 2008; Rokach, 2016).

Tree-aware methods strive to build smaller trees by limiting the total number of nodes within the forest. Several families have been proposed. For instance, Breiman (1999) learns the forest with a subsample of the training data. Some authors have proposed to relax the trees into DAGs *a posteriori* at first (e.g., Peterson & Martinez, 2009) and more recently *a priori* (Shotton et al., 2013). Similarly, techniques working on the whole dataset and yielding ensemble of trees can be partitioned into pre- and post-pruning methods. Pre-pruning methods aim at stopping the development of uninteresting branches in the top down induction procedure. On the other hand, the goal of post-pruning methods is to discard *a posteriori* subtrees which do not provide significant accuracy improvements.

Originally, pruning methods were 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 variance and rendered pruning mostly unnecessary from the point of view of accuracy. 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 (Ren et al., 2015), compression is undertook with a sequential optimization approach by removing iteratively the least interesting leaves. 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, Elisha & Dekel (2016) discard all the nodes 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 for building the whole forest first, and consequently the high temporary memory and computational costs, as their only major drawbacks.

Although our aim is to pre-prune random forests, the GIF algorithm shares similarity with Boosting methods (Friedman, 2001), which fit additive tree ensembles based on a global criterion and are also able to build accurate yet small models. Whereas most Boosting methods only explore ensembles of fixed-size trees, GIF does not put any prior complexity constraint on the individual trees but instead adapts their shape greedily. It shares this property with Johnson & Zhang (2014)'s regularized greedy forests (RGF), a method proposed to overcome several limitations of standard gradient boosting. The link between GIF and these methods will be discussed further in Section 3.3.

Algorithm 1 Globally Induced Forest

1: **Input:** $D = (x_i, y_i)_{i=1}^N$, the learning set with $x_i \in \mathbb{R}^P$ and $y_i \in \mathbb{R}^K$; $\mathcal{A}$, 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: $\hat{y}^{(0)}(.) = \arg\min_{y \in \mathbb{R}^K} \sum_{i=1}^N L(y_i, 0)$
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: $C = C \cup \{j_l, j_r\}$; $t = t + 1$
13: **until** budget B is met

3. Globally Induced Forest

GIFs rely on the view of a T trees 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; Vens & Costa, 2011):

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

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

Algorithm 1 describes the GIF training algorithm. A visual illustration is given in Figure 1 of the supplementary materials. Starting from a constant model (step 5), it builds an additive model in the form (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 drawn uniformly at random from the total candidate list C (step 8). For each of those nodes, the weight is optimized globally according to some loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ expressing the degree to which the model predictions disagree with the ground truth (such as the $L2$ norm, for instance). The node j^* among those of C_t which contributes the most to a decrease of the 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. The node budget B accounts for the total number of nodes in the resulting forest, i.e., both internal (splitting) and external (decision) nodes. The root nodes are only accounted for when one of its children is taken into the model.

Contrary to Equation 1, each node has a non-zero weight, since it was optimized. Note however that, as soon as both its children are inserted, the parent node weight can be removed from the sum by pushing its weight to its successors.

Node selection and weight optimization. Step 9 of Algorithm 1 can be decomposed into two parts. First, the optimal weight for a given candidate node is computed using:

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

Closed-form formulas for optimal weights are derived in Sections 3.1 and 3.2 for two losses. Second, the optimal node—the one which reduces the loss the most—is selected with exhaustive search. Computing the loss gain associated to a candidate node j can be done efficiently as it requires to go only over the instances reaching that node j. Indeed, finding the optimal node j_t^* at step t requires to compute:

$$j_t^* = \arg\min_{j \in C_t} \sum_{i=1}^{N} \text{err}_{j,i}^{(t)} = \arg\max_{j \in C_t} \sum_{i=1}^{N} (\text{err}_i^{(t-1)} - \text{err}_{j,i}^{(t)}), \quad (3)$$

where $\text{err}_{j,i}^{(t)} \triangleq L(y_i, \hat{y}^{(t-1)}(x_i) + w_j^* z_j(x_i))$ and $\text{err}_i^{(t-1)} \triangleq L(y_i, \hat{y}^{(t-1)}(x_i))$. Given that $z_j(x_i) \neq 0$ only for the instances reaching node j, Equation (3) can be simplified into:

$$j_t^* = \arg\max_{j \in C_t} \sum_{i \in Z_j} (\text{err}_i^{(t-1)} - \text{err}_{j,i}^{(t)}) \quad (4)$$

where $Z_j = \{1 \leq i \leq N | z_j(x_i) = 1\}$. Due to the partitioning induced by the tree, at each iteration, computing the optimal weights for all the nodes of a given tree is at most $O(N)$, assuming a single weight optimization runs in linear time in the number of instances reaching that node. Consequently, the asymptotic complexity of the induction algorithm is the same as the classical forest.

Note that, since the optimization is global, the candidate node weights must be recomputed at each iteration as the addition of the chosen node impacts the optimal weights of all the candidates it is sharing learning instances with. Arguably, the minimization of a global loss prevent from building the trees in parallel. The search for the best candidate could, however, be run in parallel, as could the search for the best split.

Tree learning algorithm. The tree learning algorithm is responsible for splitting the data reaching a node. This choice is made locally, meaning that it disregards the current global predictions of the model. As a consequence, the tree nodes that are selected by GIF are exactly a subset of the nodes that would be obtained using algorithm $\mathcal{A}$ to build a full ensemble. The motivation for not optimizing these splits globally is threefold: (i) our algorithm can be framed as a pre-pruning technique for any forest training algorithm, (ii) it introduces some natural regularization, and (iii) it leads to a very efficient algorithm as the splits in the candidate list do not have to be re-optimized at each iteration. Although any tree learning method can be used, in our experiments, we will 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.

3.1. Regression

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

$$w_j^{(t)} = \arg\min_{w \in \mathbb{R}} \sum_{i \in Z_j} \left(r_i^{(t-1)} - w\right)^2 \quad (5)$$

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

$$w_j^{(t)} = \frac{1}{|Z_j|} \sum_{i \in Z_j} r_i^{(t-1)} \quad (6)$$

In the case of a unit learning rate ($\lambda = 1$) and a single tree ($T = 1$), the model predictions coincide with the ones the underlying tree would provide (see Supplementary material).

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 appropriate 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

$$y_i^{(k)} = \begin{cases} 1, & \text{if the class of } y_i \text{ is } k \\ -\frac{1}{K-1}, & \text{otherwise} \end{cases} \tag{7}$$

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:

$$w_j^{(t)} = \underset{w \in \mathbb{R}^K}{\arg\min} \sum_{i=1}^{N} \exp\left(\frac{-1}{K} y_i^T \left(\hat{y}^{(t-1)}(x_i) + w z_j(x_i)\right)\right) \tag{8}$$

whose solution is not unique. In keeping with the output representation (Equation 7), we can impose a zero-sum constraint on the prediction to get a unique solution for each component $w_j^{(t,k)}$ $(1 \leq k \leq K)$ of $w_j^{(t)}$. If it is imposed at each stage, it means that

$$\sum_{k=1}^{K} \hat{y}^{(t-1,k)} = \sum_{k=1}^{K} \hat{y}^{(t,k)} = 0 = \sum_{k=1}^{K} w^{(k)} \tag{9}$$

and this is not impacted by the learning rate. The corresponding analytical solution (see Supplementary material for more details) is

$$w_j^{(t,k)} = \frac{K-1}{K} \sum_{l=1}^{K} \log \frac{\alpha_j^{(t-1,k)}}{\alpha_j^{(t-1,l)}}, \tag{10}$$

where

$$\alpha_j^{(t-1,k)} \triangleq \sum_{i \in Z_j | y_i = k} \exp\left(-\frac{1}{K-1} \hat{y}^{(t-1,k)}(x_i)\right) \tag{11}$$

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

$$P^{(t)}(k|x) = \frac{\exp\left(\frac{1}{K-1} \hat{y}^{(t,k)}(x)\right)}{\sum_{l=1}^{K} \exp\left(\frac{1}{K-1} \hat{y}^{(t,l)}(x)\right)} \tag{12}$$

In the case of a unit learning rate ($\lambda = 1$) and a single tree ($T = 1$), the probabilities thus derived coincide with the ones the underlying tree would provide (see Supplementary material).

Trimmed exponential loss. Equation (10) 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 for some k? To circumvent this problem, we propose to approximate the optimal weight (Equation 10) in the following fashion:

$$w_j^{(t,k)} = \frac{K-1}{K} \sum_{l=1}^{K} \tau_\theta\left(\alpha_j^{(t-1,k)}, \alpha_j^{(t-1,l)}\right) \tag{13}$$

$$\tau_\theta(x_1, x_2) \triangleq \begin{cases} \theta, & \text{if } x_2 = 0 \text{ or } \frac{x_1}{x_2} > e^\theta \\ -\theta, & \text{if } x_1 = 0 \text{ or } \frac{x_2}{x_1} > e^\theta \\ \log \frac{x_1}{x_2}, & \text{otherwise} \end{cases} \tag{14}$$

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, through the parameter θ, a maximum order of magnitude between the class errors. For instance, a saturation $\theta = 3$ means that the class errors imbalance is not allowed to count for more than $e^3 \approx 20$.

3.3. Discussion

GIF versus Boosting. From a conceptual point of view, GIF is very similar to gradient Boosting (Friedman, 2001), where, however, the set of base learners would be composed of node indicator functions and would be expanded at each iteration, while gradient boosting usually exploits depth-constrained decision trees. Also, GIF weights can be multidimensional to accommodate for multiclass or multi-output problems, whereas they are usually scalar in Boosting (with potentially multioutput base models). GIF's forest development mechanism makes it noticeably close to Johnson & Zhang (2014)'s RGF method that can also, in principle, build a forest greedily by choosing at each iteration the leaf to split based on a global objective function (although, to reduce computing times, only the last tree added in the forest can be further expanded in practice). As an important difference, however, splits in RGF are globally optimized based on the current forest predictions, while splits in GIF are optimized locally and only the nodes and their weights are chosen globally. This local optimization, together with the learning rate and candidate subsampling, acts as the main regularizer for GIF, while RGF uses explicit regularization through the objective function.

Forest shape. Three parameters interact to influence the shape of the (pruned) forest: 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 shape is predetermined and solely governed by the number of trees. Few trees impose a development in depth of the forest, while many trees encourage in-breadth growth. Since the selection is uniform over the candidates, it also implies

that well-developed trees are more likely to get developed further, as choosing a node means replacing it in the candidate list by its two children (unless it is a leaf). This aggregation effect should somewhat be slowed down when increasing the number of trees (in-breadth development). Note that subsampling the candidates (*i.e.* small value of CW) also acts as a regularization mechanism and reduces the computing time.

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 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 fully exploited and the algorithm will look for similar nodes at subsequent steps. In contrast, if the learning rate is high, the node will be fully exploited and the algorithm will turn to different nodes. As similar nodes tend to be located roughly at the same level in trees, low (resp. high) learning rate will encourage in breadth (resp. in depth) development.

4. Empirical analysis

All the results presented in this section are averaged over ten folds with different learning sample/testing sample splits. See the Supplementary material for detailed information on the datasets.

4.1. Default hyper-parameters

Our first experiment was to test the GIF against the Extremely randomized trees (ET). To get an estimate of the average number of nodes per tree, we first computed ten forests of 1000 fully-developed ET. We then examined how GIF compared to ET for 1% and 10% of the original budget. For GIF, these values were directly used as budget constraints. For ET, we built forests of 10 ($\text{ET}_{1\%}$) and 100 ($\text{ET}_{10\%}$) trees. The supplementary materials include further comparisons with three other local pre-pruning baselines, focusing more on the top of the trees. As these baselines tend to perform poorly, we focus our comparison below to the $\text{ET}_{1\%}$ and $\text{ET}_{10\%}$ baselines.

The extremely randomized trees were computed with version 0.18 of Scikit-Learn (Pedregosa et al., 2011) with the default parameters proposed in (Geurts et al., 2006). In particular, the trees are fully-developed and the number of features examined at each split is $\sqrt{p}$ in classification and p in regression, where p is the initial number of features. For GIF, we started with $T = 1000$ stumps, a learning rate of $\lambda = 10^{-1.5}$ and $CW = 1$. The underlying tree building algorithm is ET with no restriction regarding the depth and $\sqrt{p}$ features are examined for each split, in both classification and regression. We will refer to this parameter setting as the default one. Note that GIF is implemented on top of the Scikit-Learn library[1].

Regression was handled with the square loss. 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. The second method was to use the trimmed exponential loss with a saturation $\theta = 3$. The results are reported in Tables 1 and 2.

Regression. As we can see from Table 1, 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 2 draws an interesting conclusion: the number of classes should guide the choice of loss. In the binary case, the trimmed exponential works well. At 1%, it loses on Musk2, and the binarized version of Vowel and Letter to $\text{ET}_{1\%}$. At 10%, it only loses on binary Vowel, where it closes the gap somewhat.

When it comes to multiclassification, however, the trimmed exponential seems to suffer. 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 versions of Vowel, and Mnist indicate that GIF at 10% struggles 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 well capable of handling irrelevant features.

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 loss, with $\lambda = 10^{-1}$ at 1%, we can reach 3.74 ± 0.31 on Twonorm and 3.54 ± 0.3 on Musk2.

4.2. Influence of the hyper-parameters

Learning rate. Figure 1 depicts a typical evolution of the error with the budget for different learning rates in the case

[1]The code is readily available at https://github.com/jm-begon/globally-induced-forest

Table 1. Average mean square error at 1% and 10% budgets ($m = \sqrt{p}$, $\lambda = 10^{-1.5}$, $T = 1000$, $CW = 1$).

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 2. Error rate (%) at 1% and 10% budgets ($m = \sqrt{p}$, $\lambda = 10^{-1.5}$, $T = 1000$, $CW = 1$). $GIF_{SQ,\cdot}$ relates to the multi-output square loss. $GIF_{TE,\cdot}$ relates to the trimmed exponential loss with $\theta = 3$. The six firts datasets are binary classification. The last three are multiclass. The three in the middle are their binary versions.

DATASET	$ET_{100\%}$	$ET_{10\%}$	$GIF_{SQ,10\%}$	$GIF_{TE,10\%}$	$ET_{1\%}$	$GIF_{SQ,1\%}$	$GIF_{TE,1\%}$
RINGNORM	2.91 ± 0.40	3.28 ± 0.41	4.05 ± 0.45	3.17 ± 0.34	7.43 ± 0.55	5.35 ± 0.65	4.30 ± 0.51
TWONORM	3.13 ± 0.13	3.54 ± 0.18	3.50 ± 0.24	3.35 ± 0.22	8.00 ± 0.57	3.91 ± 0.39	3.92 ± 0.31
HASTIE	10.30 ± 0.46	11.78 ± 0.56	10.33 ± 0.41	7.38 ± 0.29	20.38 ± 0.56	7.64 ± 0.50	6.76 ± 0.42
MUSK2	3.65 ± 0.40	3.70 ± 0.37	3.41 ± 0.34	3.14 ± 0.34	4.22 ± 0.37	7.40 ± 0.38	6.65 ± 0.28
MADELON	9.75 ± 0.75	12.43 ± 0.77	9.18 ± 0.83	8.03 ± 0.60	23.91 ± 1.17	12.55 ± 0.83	12.40 ± 0.76
MNIST8VS9	0.99 ± 0.23	1.06 ± 0.23	0.86 ± 0.24	0.76 ± 0.16	1.58 ± 0.31	2.10 ± 0.35	1.53 ± 0.31
BIN. VOWEL	1.96 ± 1.04	2.28 ± 1.20	2.81 ± 1.17	2.24 ± 1.19	4.18 ± 1.70	12.28 ± 2.00	11.92 ± 2.03
BIN. MNIST	1.92 ± 0.16	2.04 ± 0.21	1.76 ± 0.15	1.59 ± 0.15	3.37 ± 0.17	3.24 ± 0.20	2.76 ± 0.18
BIN. LETTER	1.80 ± 0.20	2.00 ± 0.17	2.44 ± 0.25	2.28 ± 0.19	3.59 ± 0.35	7.57 ± 0.38	6.65 ± 0.24
WAVEFORM	13.95 ± 0.58	14.47 ± 0.93	14.17 ± 0.62	14.51 ± 0.67	19.11 ± 0.57	13.26 ± 0.56	14.78 ± 0.81
VOWEL	5.92 ± 1.29	6.08 ± 1.13	7.31 ± 1.18	15.90 ± 1.35	11.74 ± 1.71	22.91 ± 2.03	36.30 ± 2.62
MNIST	2.63 ± 0.18	2.87 ± 0.19	2.26 ± 0.17	4.05 ± 0.25	4.94 ± 0.21	3.92 ± 0.25	5.68 ± 0.31
LETTER	2.53 ± 0.16	2.75 ± 0.17	2.82 ± 0.19	9.07 ± 0.53	5.34 ± 0.27	8.10 ± 0.55	19.87 ± 0.77

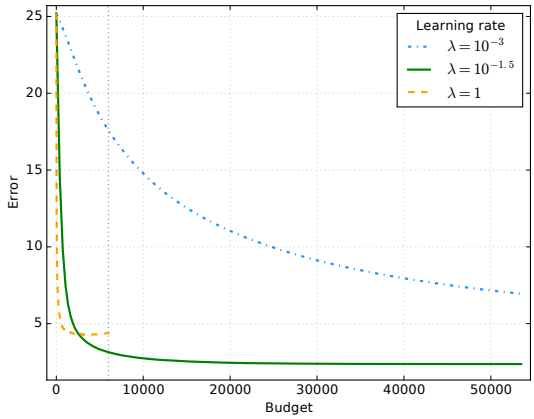

Figure 1. Friedman1: average test set error with respect to the budget B ($CW = 1$, m=$\sqrt{10}$, $T = 1000$).

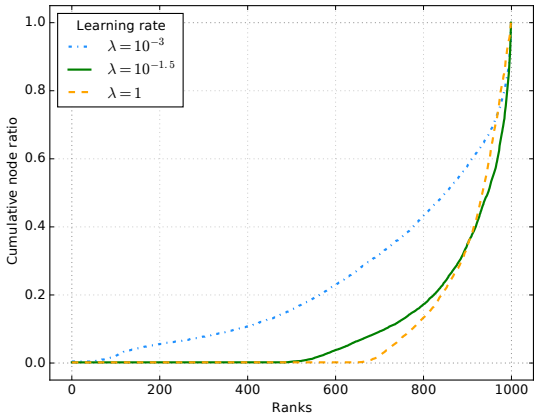

Figure 2. Friedman1: cumulative node distribution with respect to the size-ranks ($CW = \infty$, m=$\sqrt{10}$, $T = 1000$, $B = 10\%$).

of Friedman1 (the budget maxes out at 59900 nodes, corresponding to 10%). A unit learning rate will usually decrease the test set error rapidly but will then either saturate or overfit. 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 shape, provided the candidate window size is large enough. Figure 2 portrays the cumulative node distribution with respect to the size-ranks of the trees for $CW = \infty$, meaning that f(x) is the ratio of nodes of the x/T smallest trees. We can see that, for the smallest learning rate, 80% of the smallest trees account for approximately 43% of the nodes. At the same stage, only 17% and 13% of the nodes are covered for the average and biggest learning rates, respectively.

Number of features. Table 3 shows how the error varies at 10% for $CW = 1$ with respect to both the learning rate λ and m, the number of features examined for a split, in the case of CT slice and Musk2, two datasets with many features. Interestingly, the error tends to vary continuously over those two parameters. On both datasets, it appears that the choice of learning rate (global parameter) is more critical than the number of features (local parameter). The optimal number of features remains problem-dependent, though.

Candidate window size. Figure 3 illustrates the influence of the candidate window size on both the error and the fitting time for several datasets with $\lambda = 10^{-1.5}$, $m = \sqrt{p}$, $T = 1000$ and a budget=10%. Firstly, the linear dependence of the window size on the building time

Table 3. Average test set error with respect to m and λ ($CW = 1$, $T = 1000$, $B = 10\%$; $\theta = 3$). In bold is $m = \sqrt{p}$.

CT slice: mean square error

m \ λ	$10^{-2.5}$	10^{-2}	$10^{-1.5}$	10^{-1}	$10^{-0.5}$
19	27.28	20.34	19.31	21.97	29.82
38	25.78	19.51	18.63	20.88	27.62
96	25.53	19.74	18.79	20.68	26.64
192	26.55	20.96	19.92	21.62	26.87
288	28.20	22.43	20.91	22.31	27.64
385	31.42	25.04	23.11	24.17	29.56

Musk2: error rate (%)

m \ λ	$10^{-2.5}$	10^{-2}	$10^{-1.5}$	10^{-1}	$10^{-0.5}$
12	5.13	3.74	3.14	2.90	2.86
16	5.00	3.67	3.11	2.91	2.85
41	4.50	3.39	3.00	2.93	2.93
83	4.24	3.26	2.92	2.88	2.90
124	4.11	3.20	2.89	2.79	2.75
166	4.11	3.19	2.94	2.84	2.86

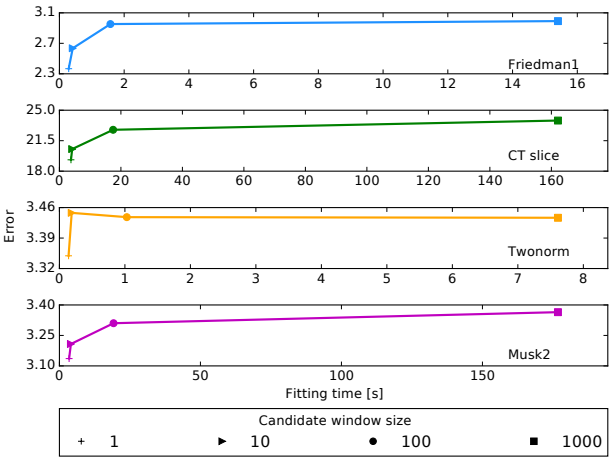

Figure 3. Average test set error (MSE for Friedman1 and CT slice, error rate (%) for Twonorm and Musk2) and fitting time with respect to CW ($\lambda = 10^{-1.5}$, $m = \sqrt{p}$, $T = 1000$, $B = 10\%$; $\theta = 3$).

is clearly visible. More interestingly, the smaller window size ($CW = 1$) performs best on all four datasets. All in all, this seems to be a good regularization mechanism, allowing for a dramatic decrease of computing times while ensuring better predictions.

Although this is representative of the regression and binary classification problems, this is not exactly the case of multi-classification, where increasing CW over 1 might improve performance slightly (see Table 4).

Number of trees. The initial number of trees is an intricate parameter, as it impacts model predictions, the fitting time and the shape of the forest.

Table 5 focuses on the errors with $m = \sqrt{p}$ and $\lambda = 10^{-1.5}$. Unsurprisingly, the models perform badly when it has only 10 trees at its disposal; this leaves only little

Table 4. Error rate (%) for the trimmed exponential loss ($\theta = 3$, $\lambda = 10^{-1.5}$, $m = \sqrt{10}$, $T = 1000$, $B = 10\%$)

DATASET	CW=1	CW=10
WAVEFORM	14.51 ± 0.67	14.05 ± 0.82
VOWEL	15.90 ± 1.35	10.87 ± 1.61
MNIST	4.05 ± 0.25	3.66 ± 0.31
LETTER	9.07 ± 0.53	5.88 ± 0.32

Table 5. Test set error with respect to the initial number of trees T ($m = \sqrt{p}$, $\lambda = 10^{-1.5}$, same budget $B = 10\%$; $\theta = 3$).

Friedman1: mean square error

T	CW=1	CW=∞
10	7.88 ± 0.64	7.62 ± 0.71
100	3.31 ± 0.41	3.60 ± 0.35
1000	2.37 ± 0.24	3.05 ± 0.29
10000	2.26 ± 0.20	3.18 ± 0.28

Twonorm: misclassification rate (%)

T	CW=1	CW=∞
10	7.47 ± 0.73	7.05 ± 0.29
100	3.44 ± 0.16	3.52 ± 0.13
1000	3.35 ± 0.22	3.43 ± 0.23
10000	3.53 ± 0.25	3.87 ± 0.32

room for the learning algorithm to optimize globally. The more trees is not always better, however. When the candidate window is infinitely large, this might be due to overfitting: there are so many candidates to choose from that over-optimization hurts the model. When the window size is 1, this is more directly linked to the forest shape.

Table 6 holds the normalized entropy of the node distribution across trees for Friedman1. By "normalized", we mean that the entropy was divided by its maximal possible value $\log_2 T$ and then multiplied by 100. Only one value is reported for the case $CW = 1$ as the forest has always the same shape, whatever the learning rate λ. The evolution of the entropy for a fix number of trees when $CW = \infty$ has already been commented on (see Figure 2). It is rendered more obvious when the initial number of trees is larger, however, meaning that GIF is able to exploit the greater freedom offered by the additional trees. When $CW = 1$, the distribution is much closer to being uniform (entropy close to 100) than when the learning algorithm can adapt the forest shape. If this shape does not agree with the data, the model might perform less well. Nevertheless, as we saw, $CW = 1$ yields better result on all but the multiclass problems, and $T = 1000$ seems to be adequate in average.

The number of trees also impacts the learning time, as depicted by Table 7. The linear increase in computing time in the case of $CW = \infty$ is due to the global optimization of the chosen node that must run through all the candidates. In the case of $CW = 1$, the computing time is almost not burdened by the number of trees. The slight increase is actually related to the forest shape: since the distribution of node tends to be more uniform, the algorithm must run through more examples while optimizing the weights (higher part of the trees).

Table 6. Friedman1: average normalized node distribution entropy with respect to T and λ ($m = \sqrt{p}$, same budget $B = 10\%$).

	CW=1	CW=∞		
T \ λ	*	10^{-3}	$10^{-1.5}$	1
100	99.89	99.84	99.24	98.48
1000	98.15	94.49	87.32	83.72
10000	97.20	89.12	76.23	68.99

Table 7. Friedman1: fitting time (seconds) with respect to T and λ ($m = \sqrt{p}$, same budget $B = 10\%$).

	CW=1	CW=∞	
T \ λ	*	10^{-3}	1
100	0.34 ± 0.07	0.35 ± 0.07	0.32 ± 0.07
1000	0.59 ± 0.12	3.84 ± 0.18	2.78 ± 0.54
10000	1.55 ± 0.02	25.95 ± 1.05	20.69 ± 2.92

4.3. A preliminary comparison with Boosting

In this section, we carry out a first comparison of GIF with Boosting. To submit Boosting to the budget constraint, we have used stumps as base learners and have made as many trees as were necessary to meet the constraint. We have used the same learning rate as for GIF in Table 1. Regression has been tackled with least square Boosting (Friedman et al., 2001) and classification with Adaboost (Freund & Schapire, 1995), so that the same losses are used for GIF and Boosting. Scikit-Learn was used as Boosting implementation.

Table 8 holds the errors for Boosting at 1% and 10%. In the default setting, GIF beats Boosting on all regression datasets except Abalone where it performs slightly less well. Interestingly, Boosting also overfits on Abalone and Hwang. The situation is more contrasted in classification, where Boosting outperforms GIF on Hastie and Musk2 for both budget constraints. Notice that stumps are not optimal for Hwang and CT slice, where a depth of 2 would yield lower errors of 11.09 ± 0.25 and 8.40 ± 0.19 at 10% and 1% respectively for Hwang and 33.53 ± 1.65 and 36.67 ± 1.36 at 10% and 1% respectively for CT slice. However, this does not change the conclusions regarding the comparison with GIF.

GIF (with $CW = 1$) is faster in both learning and prediction than Boosting, as confirmed in Table 9. Firstly, Boosting's base learners are traditional decision trees, which are slower to fit than ET for a given structure. Secondly, Boosting's base learners are shallow and they can thus take less advantage of the partitioning induced by the trees.

Overall, the performances of Boosting and GIF in terms of errors are somewhat similar. Sometimes GIF's extra-layers of regularization, combined with a greater variety of depths pays off and sometimes not. However, GIF is faster in both learning and prediction.

Table 8. Test set error (MSE/error rate ($\%$)) for stump least-sqaure Boosting/Adaboost under budget constraints ($\lambda = 10^{-1.5}$).

DATASETS	$B = 10\%$	$B = 1\%$
FRIEDMAN1	4.53 ± 0.23	3.86 ± 0.10
ABALONE	5.17 ± 0.20	4.83 ± 0.20
CT SLICE	82.44 ± 3.80	68.73 ± 1.92
HWANG $\times 10-2$	97.88 ± 2.33	88.62 ± 1.73
RINGNORM	5.48 ± 0.55	6.71 ± 0.99
TWONORM	5.09 ± 0.56	5.98 ± 0.47
HASTIE	5.65 ± 0.34	7.10 ± 0.41
MUSK2	2.70 ± 0.37	4.20 ± 0.28
MADELON	11.30 ± 0.68	11.33 ± 0.69

Table 9. Musk2: fitting/prediction times (seconds). Stump Adaboost versus GIF (trimmed loss with $\theta = 3$, $T = 1000$, $m = \sqrt{p}$, $CW = 1$) for $B = 10\%$ and $\lambda = 10^{-1.5}$.

	Adaboost	GIF
Fitting	399.17 ± 60.91	1.53 ± 0.04
Prediction	28.39 ± 5.43	0.31 ± 0.07

5. Conclusion and perspectives

In this paper, we introduced the Globally Induced Forest (GIF) whose goal is to produce lightweight yet accurate tree-based ensemble models by sequentially adding nodes to the model. Contrary to most tree-aware techniques, our method is framed as a pre-pruning method that does not require the *a priori* building of the whole forest. Several hyper-parameters govern the learning algorithm. We have proposed a set of default parameters which seems to work quite well in average, beating the baselines, under mild and severe memory constraints. Needless to say that the setting of these parameters can be further optimized if necessary, although this goes against the philosophy of building directly the pruned forest. Of the most interest is the conclusion that it is usually better not to optimize the choice of nodes. In other words, letting the algorithm optimize the forest shape is—surprisingly—harmful. Although it complicates the choice of the initial number of trees, this makes the algorithm extremely fast.

The main focus of subsequent works should be to handle multiclass problems better. Several extensions can also be thought of. For instance, one could consider introducing both children at the same time at each iteration or allow for the refitting of the already chosen nodes by leaving them in the candidate list. Finally, we would also like to explore further the comparison between GIF and boosting methods, in particular Johnson & Zhang (2014)'s regularized greedy forests, which share similar traits with GIF.

Acknowledgements

Part of this research has been carried out while Arnaud Joly was a research fellow of the FNRS, Belgium. Computational resources have been provided by the Consortium

des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11. This work is also supported by the DYSCO IUAP network of the Belgian Science Policy Office.

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.

Freund, Yoav and Schapire, Robert E. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pp. 23–37. Springer, 1995.

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

Friedman, Jerome H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

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

Johnson, Rie and Zhang, Tong. Learning nonlinear functions using regularized greedy forest. *IEEE transactions on pattern analysis and machine intelligence*, 36 (5):942–954, 2014.

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.

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

Peterson, Adam H and Martinez, Tony R. Reducing decision tree ensemble size using parallel decision dags. *International Journal on Artificial Intelligence Tools*, 18 (04):613–620, 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.

Shotton, Jamie, Sharp, Toby, Kohli, Pushmeet, Nowozin, Sebastian, Winn, John, and Criminisi, Antonio. Decision jungles: Compact and rich models for classification. In *Advances in Neural Information Processing Systems*, pp. 234–242, 2013.

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.

Vens, Celine and Costa, Fabrizio. Random forest based feature induction. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pp. 744–753. IEEE, 2011.

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