Globally Induced Forest: A Prepruning Compression Scheme

Anonymous Authors[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. 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 applications.

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

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 their associated weights, while still choosing locally, based on the standard local 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 circumvent the need to build the whole forest first, resulting in a fast learning algorithm and discarding the need for a large temporary storage.

After discussion the related work in section 2, section 3 introduces the GIF algorithm and how it can be applied for both regression (subsection 3.1) and classification (subsection 3.2). In section 4, we show that our proposed algorithm, with its default set of parameters, perform well on many datasets, even surpassing much larger models sometimes. We then conduct an analysis of the hyper-parameters entailed in GIF (subsection 4.2). Since GIF shares some resemblance to Boosting, the two are compared in subsection 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. 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. 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. Pre-pruning methods aim 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 with a sequential optimization problem in (Ren et al., 2015) by removing 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.

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: $\hat{y}^{(0)}(.) = \arg\min_{y \in \mathbb{R}^K} \sum_{i=1}^N L(y_i, 0)$
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: $C = C \cup \{j_l, j_r\}$; $t++$
13: **until** budget B is met

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), (Vens & Costa, 2011):

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

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 $w_j^{(k)}$ $(k = 1, \ldots, K)$ 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 ac-

cording 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 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 Boosting (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 loss can be done efficiently by going only over the instances reaching node j^*:

$$\text{err}^{(t-1)} \triangleq \sum_{i=1}^{N} L\left(y_i, \hat{y}^{(t-1)}(x_i)\right) \tag{2}$$

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

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

$$\text{gap}_j^{(t)} \triangleq \text{err}^{(t-1)} - \text{err}_j^{(t)} \tag{5}$$

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

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

where $Z_j = \{1 \leq i \leq N | z_j(x_i)\}$. This is due to the fact that $\hat{y}^{(t-1)}(x_i) \neq \hat{y}^{(t)}(x_i)$ only for the instances reaching node j. Due to the partitioning induced by 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 weight optimization runs in linear time in the number of instances reaching that node.

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 predictions of the model. 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 B 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 impose a development in depth of the forest, while many trees encourages 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).

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 fully employed and the algorithm will look for similar nodes at subsequent steps. In contrast, if the learning rate is high, the node will be fully employed 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.

GIF versus Boosting From a conceptual point of view, GIF is somewhat an intermediate model between Random forest and Boosting. Similarly to Boosting, an additive model is fit sequentially by optimizing globally some weight. Contrary to Boosting, the trees' splits in GIF are not optimized globally, however, and no depth restriction is imposed on the trees constituting the ensemble. Notice

also that the weight can be multidimensional to accommodate for multiclass or multioutput problems, whereas it is usually scalar in Boosting.

3.1. Regression

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

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

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:

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

In the case of a unit learning rate ($\lambda = 1$) and a single tree ($T = 1$), 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:

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

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

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

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

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

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

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

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 output representation (Equation 10), we can impose a zero-sum constraint on the prediction to get a unique solution for the kth component of $w_j^{(t)}$. If it is imposed at each stage, it means that

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

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

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

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

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

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 ($\lambda = 1$) and a single tree ($T = 1$), 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:

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

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

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.For instance, a saturation $\theta = 3$ means that the class errors imbalance is not allowed to count for more than $e^3 \approx 20$.

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. Binary versions of the Mnist, Letter and Vowel datasets have been created as well by grouping the first half and second half classes together.

Table 1. Datasets. N is the learning sample size, p is the number of features.

DATASET	N	TS SIZE	p	# CLASSES
FRIEDMAN1	300	2000	10	-
ABALONE	2506	1671	10	-
CT SLICE	2000	51500	385	-
HWANG F5	2000	11600	2	-
CADATA	12384	8256	8	-
RINGNORM	300	7100	20	2
TWONORM	300	7100	10	2
HASTIE	2000	10000	10	2
MUSK2	2000	4598	166	2
MADELON	2200	2200	500	2
MNIST8VS9	11800	1983	784	2
WAVEFORM	3500	1500	40	3
VOWEL	495	495	10	11
MNIST	50000	10000	784	10
LETTER	16000	4000	8	26

All the results presented in this section are averaged over ten folds with different learning sample/testing sample splits.

4.1. Default hyper-parameters

Our first experiment was to test the GIF against the Extremely randomized trees (ET). To get a 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 ($\text{ET}_1 0\%$) trees.

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. Note that GIF is built on top of the Scikit-Learn.

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 exponen-

tial loss with a saturation $\theta = 3$.

In all cases, the experiment were repeated ten times with different learning sample/testing sample splits. The results are reported in 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 the binarized version of Vowel and Letter to $\text{ET}_{1\%}$. At 10%, the conclusion is the same with the addition that the trimmed exponential manages to be competitive with $\text{ET}_{10\%}$ on binary Vowel. On binary Letter, GIF closes the wide 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 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 $\lambda = 10^{-1}$ and 3.54 ± 0.3 on Musk2 at 1% for $\lambda = 10^{-1}$.

Other baselines Notice that other baselines, more focused on the top of trees, such as starting with 1000 stumps and developing the nodes randomly, in breadth first fashion or with a best-first local heuristic work against the ensemble averaging. For instance, a best-first approach could be to develop the node which reduces the impurity the most. For instance, this would result—at 1% under the same protocol—with errors of 15.4 ± 0.5 for Friedman1 and 31.4 ± 18.0 for Ringnorm.

4.2. Influence of the hyper-parameters

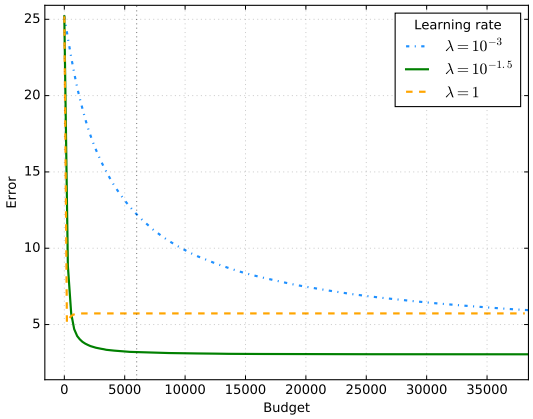

Figure 1. Friedman1: Testing set mean square error with respect to the budget ($CW = \infty$, m=$\sqrt{10}$, $T = 1000$, budget=59900 (10%)).

Learning rate Figure 1 depicts a typical evolution of the error with the budget for different values of learning rates 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 either 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 shape, provided the candidate window size is large enough. In Figure 2, 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 4 shows how the error varies with respect to both the learning rate λ and 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. Firstly, the linear dependence of the window

Table 2. Average 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. Average misclassification rate (in percent). $GIF_{SQ,\cdot}$ relates to the multi-output square loss. $GIF_{TE,\cdot}$ relates to the trimmed exponential loss with $\theta = 3$.

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

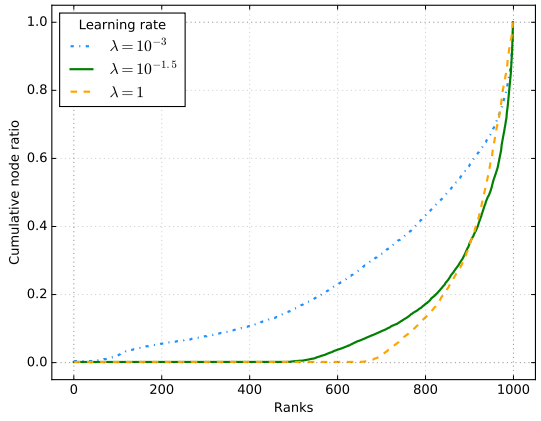

Figure 2. Friedman1: node distribution across tree ($CW = \infty$, m=$\sqrt{10}$, $T = 1000$, budget=59900 (10%)).

Table 4. Average mean square error with respect the number of featuer m and the learning rate λ ($CW = 1$, $T = 1000$, budget=10%). In bold is $m = \sqrt{p}$.

CT slice

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

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

size on the building time is clearly visible. More interestingly, the smaller window size performs best on all four datasets. All in all, this seems to be a good regularization mechanism, allowing for a dramatic decrease of computation times while allowing for better predictions.

Although this is representative of the regression and binary classification, this is not exactly the case of multiclassification, where increasing CW over 1 might improve performance slightly (see Table 5).

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

Table 6 focuses on the errors. Unsurprisingly, the models perform badly when it has only 10 trees at its disposal; this leaves only few room for the learning algorithm to optimize

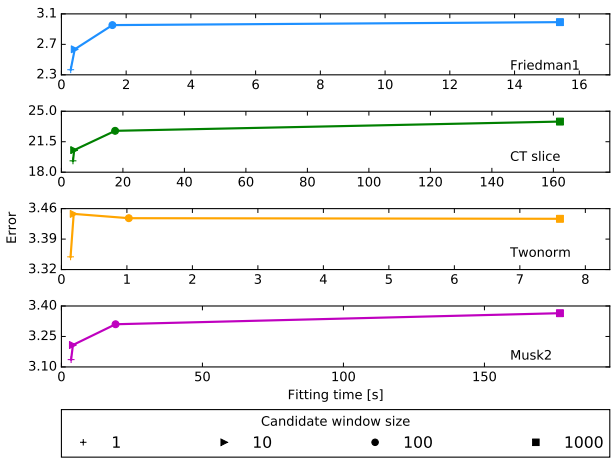

Figure 3. Test set error with respect to time for several values of candidate window size (m=$\sqrt{p}$, $T = 1000$, budget=10%). Friedman1 and CT slice were computed with the square loss and the error is mean square error. Twonorm and Musk2 were computed using the trimmed exponential loss with a saturation of 3 and the error is the misclassification rate (in percent). Results were aggregated over ten runs.

Table 5. Average misclassification rate (in percent) for the trimmed exponential with a saturation of 3 (m=$\sqrt{10}$, $T = 1000$, budget=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

globally. The more trees is not always the 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's shape.

Table 7 holds the normalized entropy of the node distribution across trees. By "normalized", we mean that the entropy was divided by $\log_2 T$ 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. 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 shapes 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.

Table 6. Average error on Friedman1 and Twonorm (trimmed exponential loss with saturation of 3.) with respect to the initial number of trees T and with a node budget of to 10% of 1000 fully-developed trees ($m = \sqrt{p}$, $\lambda = 10^{-1.5}$).

Friedman1		
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		
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

Table 7. Friedman1: average normalized node distribution across trees' entropy on Friedman1 with respect to the initial number of trees T and with a node budget of 10% of 1000 fully-developed trees ($m = \sqrt{p}$, $\lambda = 10^{-1.5}$).

	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

The number of trees also impacts the learning time, as depicted by Table 8. The linear increase in computation 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 computation time is almost not burdened by the number of trees. The slight increase is actually related to the forest's 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).

4.3. Comparison with Boosting

Although not our starting point, Boosting is an other method to build ensemble of trees. In this section, we compare how GIF fares against it.

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 in Table 2. Regression has been tackled with Gradient Boosting over a square loss (**?**). Adaboost (Freund & Schapire, 1995) is in charge of classification. As a consequence, the same losses are used for GIF and Boosting. Once again, Scikit-Learn was used as

Table 8. Average fitting time (in seconds) on Friedman1 with respect to the initial number of trees T ($m = \sqrt{p}$, $\lambda = 10^{-1.5}$, budget=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

Table 9. Average error for Boosting.

DATASETS	10%	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

Boosting implementation.

Table 9 holds the errors for Boosting at 1% and 10%. In the default settings, 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 Table 10 confirms. Firstly, Boosting's base learners are traditional decision trees, which are slower to fit than ET. Secondly, the base learners are shallow. In the fitting phase, this means that they must handle more data. In the prediction phase, it means they can take less advantage of the trees induced partitioning.

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 10. Average fitting and prediction time (in seconds) on Musk2 for Boosting (depth=1, $\lambda = 10^{-1.5}$) and GIF (trimmed exponenantial loss with saturation=3, $T = 1000$ $m = \sqrt{p}$, $\lambda = 10^{-1.5}$, $CW = 1$) for a budget of 10%.

	Boosting	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, post-pruning techniques, our method does not require the *a priori* building of the whole forest and works on the whole learning set.

Although motivated as pruning technique for Random forests, the learning algorithm shares similar traits with Boosting, the main differences being the extra layer of regularization and the shape adaptability of GIF. Note that adapting the forest shape with Boosting as been explored by Johnson & Zhang (2014) outside of pruning consideration.

Several hyper-parameters govern the learning algorithm. We have proposed a set of default parameters which seems to work quite well in average, beating, under mild and severe memory constraints, both the Extremely randomized trees and Boosting on several datasets. Needless to say that the setting can be further refined 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 no to optimize the choice of nodes. In other words, letting the algorithm optimize the forest's shape is—surprisingly—harmful. Although it complicates the choice of the initial number of trees, this makes for a fast learning algorithm.

The main focus of subsequent works should be to handle multiclass problems better. Aside from that, frequent uses of the tree-based techniques, such as measuring feature importance and using them as codebook, should be examined. Several extensions can also be thought of. For instance, one could allow for the refitting of the already chosen nodes by leaving them in the candidate list. The split might also be optimized globally if the candidate window is small enough, although this might result in overfitting. Multi-output losses which take into account the correlation between the outputs can be proposed. Finally, a online variant of the algorithm would be interesting for the case where a extremely lightweight model is desired on a very large

dataset.

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

$$\begin{aligned}
\exp\left(\frac{1}{K-1}w_j^{(l)}\right) &= \frac{1}{c_j}\alpha_{\pi_i}^{(l)} && (30)\\
&= \frac{1}{c_j}\sum_{i\in Z_j^{(l)}}\exp\left(-\frac{1}{K-1}\hat{y}_{\pi_i}^{(l)}\right) && (31)\\
&= |Z_j^{(l)}|\exp\left(-\frac{1}{K-1}\hat{y}_j^{(l)}\right) && (32)\\
\exp\left(\frac{1}{K-1}\hat{y}_j^{(l)}\right) &= \exp\left(\frac{1}{K-1}\hat{y}_{\pi_j}^{(l)}\right)\exp\left(\frac{1}{K-1}w_j^{(l)}\right) && (33)\\
&= \frac{1}{c_j}|Z_j^{(l)}| && (34)\\
P_j(l) &= \frac{\exp\left(\frac{1}{K-1}\hat{y}_j^{(l)}\right)}{\sum_{k=1}^{K}\exp\left(\frac{1}{K-1}\hat{y}_j^{(k)}\right)} = \frac{|Z_j^{(l)}|}{|Z_j|} && (35)
\end{aligned}$$

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.

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.

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.