Splitting Stump Forests: Tree Ensemble Compression for Edge Devices (Extended Version)

Fouad Alkhoury[1,5*], Sebastian Buschjäger[2,5] and Pascal Welke[3,4]

[1*] University of Bonn, Germany.
[2]Technical University of Dortmund, Germany.
[3]Lancaster University Leipzig, Germany.
[4]TU Wien, Austria.
[5] Lamarr Institute for Machine Learning and Artificial Intelligence, Germany.

*Corresponding author(s). E-mail(s): alkhoury@cs.uni-bonn.de;
Contributing authors: sebastian.buschjaeger@tu-dortmund.de;
p.welke@lancaster.ac.uk;

Abstract

We introduce Splitting Stump Forests – small ensembles of weak learners extracted from a trained random forest. The high memory consumption of random forests renders them unfit for resource-constrained devices. We show empirically that we can significantly reduce the model size and inference time by selecting nodes that evenly split the arriving training data and applying a linear model on the resulting representation. Our extensive empirical evaluation indicates that Splitting Stump Forests outperform random forests and state-of-the-art compression methods on memory-limited embedded devices.

Keywords: Ensemble compression, Random forests, Edge devices.

1 Introduction

The global count of Internet of Things (IoT) devices is expected to reach approximately 32 billion units by 2030 [1]. Many IoT devices require real-time decisions and therefore include limited computing capabilities [2]. Running machine learning models directly on embedded devices is increasingly popular due to improvements in reliability, affordability, and energy efficiency [3]. Local models also reduce or even eliminate the need for transferring data to cloud servers when connectivity, bandwidth consumption, communication costs, network latency, or privacy are relevant concerns [4, 5].

Ensemble models can outperform the predictive performance of individual classifiers in many machine learning tasks [6, 7]. However, ensemble models combine multiple base models, resulting in high memory consumption. The model size is also a primary determinant of inference time, an aspect equally important as model accuracy in real-time applications. IoT devices, however, typically contain resource-constrained microcontrollers with limited flash memory, ranging from a few Kbytes to a maximum of a few Mbytes, as shown in Table 1. As a result, the best-performing ensemble model is often too large and too slow to be deployed for real-time applications in IoT devices. This work presents small tree ensemble models that provide responsive and highly accurate predictions. Decision trees efficiently represent sequences of if-else conditions and can be compiled to run efficiently on embedded devices [8, 9]. We revisit these models and propose a lossy compression scheme for their ensembles, random forests. A random forest reduces the variance of the predictions compared to a single decision tree [10]. However, having only a few small trees in a random forest can hinder predictive performance, while large random forests pose a significant challenge for resource-limited devices. Our experiments show that high-performing random forests often exceed 100K nodes. As a result, these models may require more than 5-10 Mbytes of memory even in very compact implementations [9].

We propose a novel method to create a new, *compressed* ensemble from a large random forest model, often comprising hundreds of thousands of nodes. To enable compression, the approach extracts a subset of test nodes from a trained random forest to build a smaller ensemble of *splitting stumps* with a total size of only a few Kbytes. Our approach combines the supervised selection of splits in the training of random forests with an unsupervised measure of balance on the training data and an optional quantization step of the split nodes. We argue that this reduces the tendency to overfit the training data. The final model transforms the input data into multi-hot encoding and trains a linear classifier to map the novel representation to the target domain. We evaluate our proposed method twofold: First, we show in an extensive experimental evaluation on various datasets the superiority of splitting stump forests over random forests and state-of-the-art competitive ensemble compression techniques in terms of compression rate, inference time, and predictive performance. Second, we highlight a real-world scenario where we deploy our SSF models to a small Arduino Mega-2560-R device, showcasing its applicability to real-world devices. Moreover, our experiments demonstrate that the selected test nodes are *informative* and not accidental. This article proceeds as follows: We review related work in Section 2.

Table 1 Flash Memory on different microcontroller units [11–13]

Microcontroller Unit	Flash Memory
Arduino Mega-2560-R	8 KB
ATmega169P	16 KB
Arduino Nano	26-32 KB
Arduino Uno	32 KB
Atmel ATSAM3S2AA-AU	64 KB
Arduino Mega	256 KB

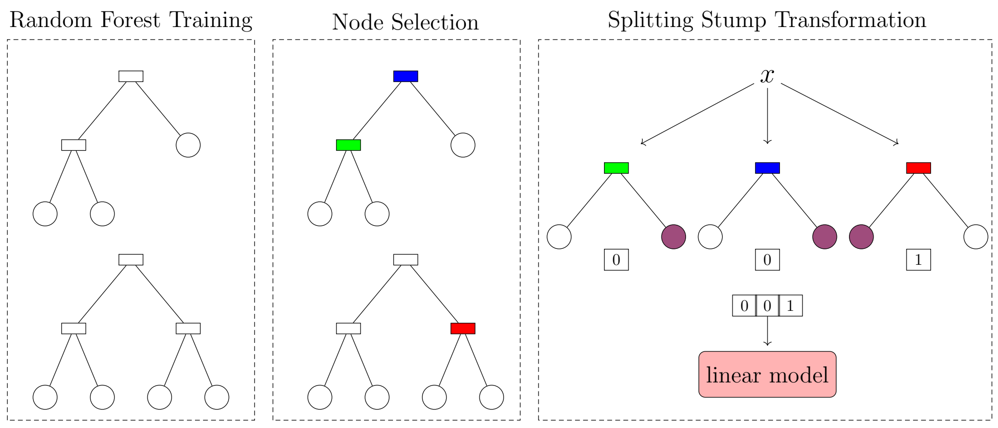

Fig. 1 Splitting stump forests at a glance. Given a random forest (left), selected nodes are selected (middle) and used as stumps (right). A training example is then transformed into a binary vector by the stumps and fed to a linear model.

Section 4 provides a detailed description of our method. Section 5 describes our empirical evaluation of SSFs. Section 6 presents the Arduino implementation before Section 7 concludes. Code for splitting stump forests is on github[1].

This article is an extended version of Alkhoury and Welke [14]. In the present version, we introduce the quantization of the split values, which further reduces the size of the compressed model. Furthermore, we extend our work with an implementation of SSFs for the Arduino Mega-2560-R. Both extensions warrant additional experiments.

2 Background

Our proposed random forest compression technique can be alternatively viewed as a model compression or representation learning approach. Deploying models on embedded devices requires special care. We now review related work in these fields.

2.1 Ensemble Compression

The existing methods for ensemble size reduction can be categorized into black box and white box approaches. Black box approaches do not assume any particular model architecture. Instead, they work on the set of models in an ensemble without changing individual base models. White box approaches update base models, e.g., by pruning individual nodes of decision trees in a random forest. We focus our discussion on white box approaches.

Seeking an optimal sub-ensemble within large random forests is often impractical. In general, identifying an optimal subset of classifiers with the best generalization performance is an NP-complete problem [15]. Thus, most approaches identify a sub-ensemble with near-optimal performance. Several studies have been conducted on the pruning of machine learning models [16]. Moreover, considerable efforts concentrated on identifying subsets of random forest nodes that can match the original forest's

[1] https://github.com/FouadAlkhoury/SplittingStumpsForests/

accuracy. Peterson and Martinez [17] introduced a post-training technique that stores unique subtrees and combines redundant nodes into "parallel nodes" while maintaining the overall behavior. Buschjäger and Morik [18] introduced an innovative method that integrates regularization into the leaf-refinement process. Their proposed algorithm jointly prunes and refines trees, thereby enhancing the performance of tree ensembles. Prior research has indicated that high diversity among ensemble models can enhance their generalization performance. Li et al. [19] select classifiers that both minimize empirical error and lead to greater ensemble diversity. Ranking-based strategies sort individual models by their associated prediction error and select a few highly ranked members to compose the sub-ensemble [20]. Nakamura and Sakurada [21] reduce the number of distinct split conditions by sharing a common condition among multiple nodes which allows for practical model size reductions. Other studies have found that removing low-impact nodes from a decision tree can simplify it while preserving accuracy [22]. In contrast, our approach constructs a new ensemble that contains more but smaller trees than the original ensemble. Ren et al. [23] enhance the fitting power of a tree ensemble model by global leaf value refinement using linear regression. Through global optimization, the approach iteratively merges insignificant pairs of adjacent leaves, effectively using complementary information from multiple trees and reducing model size.

2.2 Representation Learning

Decision trees and random forests can be interpreted as representation learning [24], with studies exploring the use of a pre-trained random forest's transformed space as input for linear models. Likewise, Nakano et al. [25] integrate a representation learning component into the random forest methodology. Their method treats ensemble nodes as clusters formed by instances during recursive partitioning. Then it creates a binary vector where each element corresponds to a cluster node and is set to 1 if a training instance traverses the node. This newly created tree embedding is then combined with the original feature set. Welke et al. [26] identify frequent subtrees in a trained random forest and train a linear model on the resulting multi-hot leaf representation. Vens and Costa [27] use the encoding of nodes visited by data instances. The final feature encoding is obtained by concatenating the binary vectors of all trees in the forest. Estruch et al. [28] leverage the common components within decision tree ensembles. In this structure, the rejected splits are not discarded but stored as suspended nodes. This allows these nodes to be further explored, allowing the generation of new models.

2.3 Model Deployment

The deployment of tree ensembles on embedded systems requires not only reducing the size of the ensemble but also careful consideration of the implementation strategy. In virtually all modern programming languages, decision trees can be implemented either as a sequence of nested if-else statements or in a 'native'-style format that stores decision nodes in an array of structs, which are traversed using a loop. Both approaches offer distinct trade-offs in terms of memory layout and runtime performance. Based on a probabilistic view of DT execution, Buschjäger et al. [8, 29]

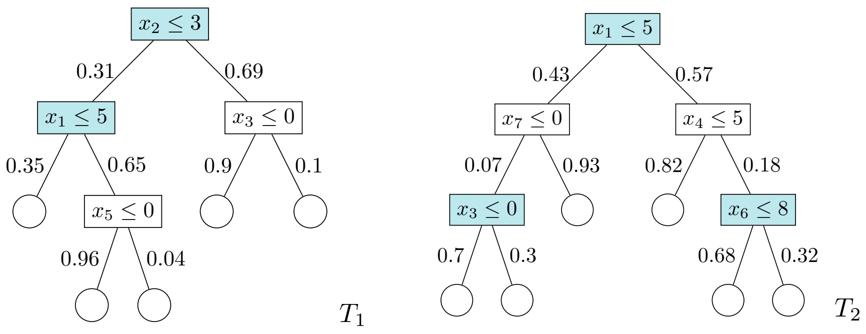

Fig. 2 A random forest F consisting of two decision trees T_1 and T_2. Round nodes represent leaves, rectangular nodes represent test nodes. The left (right) edge label represents the fraction of training instances evaluating true (false). Selected nodes in $F[p]$ for filtering threshold $p = 0.2$, are blue.

systematically investigate these implementations and propose optimal memory layouts for both, aiming to reduce cache misses and to accelerate execution. Additionally, Hakert et al. [30] introduce FLInt, a floating-point comparison operator that replaces traditional floating-point comparisons with integer and logic operations focused on if-else implementations. Beyond these classical styles, dedicated inference strategies have also emerged. One notable example is QuickScorer [31], which restructures the evaluation process by decomposing the forest into individual decision nodes. These are processed using efficient bitvector operations and bitvector comparisons, thereby improving cache locality and enabling vectorization. Extensions of this idea, such as RapidScorer [32] and parallel tree traversal methods [33], further push the efficiency of ensemble execution. Finally, with the increasing availability of tensor-based hardware such as GPUs and TPUs, new execution paradigms have been proposed. Nakandala et al. [34] introduce a tensor compiler that translates decision tree ensembles into matrix-vector operations, enabling their efficient execution on parallel hardware designed for deep learning workloads. We view these approaches as orthogonal to our work, as smaller forests benefit from a more refined implementation and vice versa.

3 Notation

In what follows, we consider a supervised learning problem where the instance space is $X \subseteq \mathbb{R}^d$ and the target space is $Y \subseteq \mathbb{R}$. Each data point $x \in X$ is a d-dimensional vector described by a set of features and mapped to the corresponding label $y \in Y$. The goal is to find a function $f\colon X \to \mathbb{R}$ such that the difference between $f(x)$ and the true label y is minimal for all $x, y \sim \mathcal{D}$ where $\mathcal{D}$ is a distribution on $\mathcal{X} \times Y$. Labeled instances $X_{train} \subseteq X$ are provided during training, while unlabeled instances $X_{test} \subseteq X$ are provided during testing. In this paper, we apply our approach to classification tasks, but an extension to regression problems is possible.

We call the root and all internal nodes of a decision tree *test nodes*. Test nodes are labeled with a split condition $x_a \leq s_v$ for a given attribute a and a split value $s_v \in \mathbb{R}$. In this work, test nodes have exactly two children, called left and right. When the split condition of node v for an instance $x \in X$ evaluates to true, the instance passes to the left child of v. An instance x recursively traverses the decision tree, following a path from the root to a leaf. These paths represent conjunctions of attribute tests, and the union of these paths constitutes the entire decision tree. Existing algorithms such as CART, ID3, and C4.5 can recursively divide the instance space into smaller subspaces to learn decision trees from labeled data [35–37]. A random forest classifier $F = \{T_j | j \in [1, t]\}$ consists of a set of t decision trees and an aggregation function to combine individual predictions, e.g., majority vote. For a comprehensive background on random forest and decision tree algorithms, refer to Breiman [10] and Quinlan [36].

4 The Splitting Stump Forests Method

In this section, we provide a description of our *splitting stump forests* method to extract a compact model from a potentially large random forest. Given a trained random forest, we (1) compute a score for each node and select nodes with high scores (Section 4.1). Our selection technique prioritizes split values that yield balanced subtrees. Subsequently, we (2) refine selected nodes by reducing split value precision, thus achieving a trade-off between model size and accuracy (Section 4.2). Afterwards, we (3) construct one decision tree per selected node and form an ensemble of stumps (Section 4.3). Finally, our approach (4) integrates a representation learning module by using the transformed space derived from the constructed ensemble as input for a linear model (Section 4.4). Figure 1 shows the pipeline of the primary steps, which we will describe in turn. Algorithm 1 shows the pseudocode of the first three steps.

4.1 Splitting Node Selection

The aim of the first step is to select a subset of balanced splits from a trained random forest F. Technically, we propose a post-hoc selection criterion that favors split conditions that lead to balanced splits. In particular, for all $T \in F$ and for all nodes v of T, we count the number of incoming training points that evaluate to true X_v^t (resp. false X_v^f) using the split condition at node v (Line 6 of Alg. 1). The score of a node v is calculated as: $score(v) \leftarrow \frac{min(|X_v^t|, |X_v^f|)}{|X_v^t| + |X_v^f|}$. For instance, when the condition evaluates to True for 6 samples out of 15 at a test node v, then the score would be 0.4. To qualify as balanced, v should attain a score that meets or exceeds a predetermined threshold p (Lines 7–8). Subsequently, we define $F[p]$ as the set of all nodes v with $score(v) \geq p$ (Line 11). In deep random forests, we can limit the size of $F[p]$ by arranging the scores in descending order and selecting a specific number of nodes with the highest scores. In a binary decision tree, $score(v) \in [0, 0.5]$ and higher values indicate a better division of training samples into two sub-samples of comparable sizes. Figure 2 shows the selection process on a small random forest. For efficient computation of $score(v)$, we store the count of training examples traversing tree edges during training. Alternatively, the counts can be derived by a single pass over an independent dataset that does not require labeling. Once these numbers are accessible, scores can be computed in constant time and high-scoring nodes can be selected by a single sweep across the random forest. Duplicate split conditions can be efficiently removed using an appropriate set data structure for $F[p]$.

Algorithm 1 Splitting Stump Forest Transformation

Input: Random forest F, training points X_{train}, threshold parameter $p \in (0, 0.5]$, quantization parameter $q \in \mathbb{N}$
Output: Splitting stump forest F' and training data representation X'_{train}.

```
1:  $F' \leftarrow \emptyset$
2:  $X'_{train} \leftarrow \emptyset$
3:  $F[p] \leftarrow \emptyset$
4:  for each tree $T \in F$ do
5:      for each node $v \in T$ do
6:          $X_v^t$(resp. $X_v^f$)$\leftarrow$ set of training points in $v$ that are True (resp. False)
7:          $score(v) \leftarrow \frac{min(|X_v^t|,|X_v^f|)}{|X_v^t|+|X_v^f|}$
8:          if $score(v) \geq p$ then
9:              Apply quantization on split value: $s'_v = round(s_v, q)$
10:             if $v \notin F[p]$ then
11:                 add $v$ to $F[p]$
12: for each $v \in F[p]$ do
13:     $T'_v \leftarrow$ construct a splitting stump by linking leaves to the node $v$
14:     add $T'_v$ to $F'$
15: for each $x \in X_{train}$ do
16:     for each $T' \in F'$ do
17:         $f_{T'}(x) \leftarrow$ assign an encoding for $x$ in $T'$
18:         add $f_{T'}(x)$ to $f_{F'}(x)$
19:     add $f_{F'}(x)$ to $X'_{train}$
20: return $F'$ and $X'_{train}$
```

Note that a scoring function for decision stump learning with a similar formula was introduced by Iba and Langley [38]. In contrast to our work, however, their score replaces e.g. the Gini index in decision stump learning and directly compares against the class label, while here we score a node in a decision tree based on the balance of training samples between its left and right branches. The most common way to train random forests is based on bagging the training data and then using a recursive algorithm (e.g. CART or ID3) with a Gini-index or mutual-information based split criterion selection. This reduces the variance of the predictions of the resulting model, but tends to increase the complexity of the model. Figure 3 shows the decision boundaries of trained random forests on two-dimensional feature-subsets of the statlog and rice datasets. In our two examples, the focus on pure splits in combination with voting results in the partition of the feature space into rather small and discontinuous regions. We argue that this may be detrimental to generalization and that simpler models may be found that yield similar performance at smaller sizes. Small regions can arise when split criteria cut off a relatively small portion of the training data with pure labels. When the split condition of a node v evaluates to true (or false) for most incoming training points $X_v \subseteq X_{train}$, it leads to imbalanced subsets at the deeper level of the tree. These imbalanced subsets consist of a nearly pure and relatively

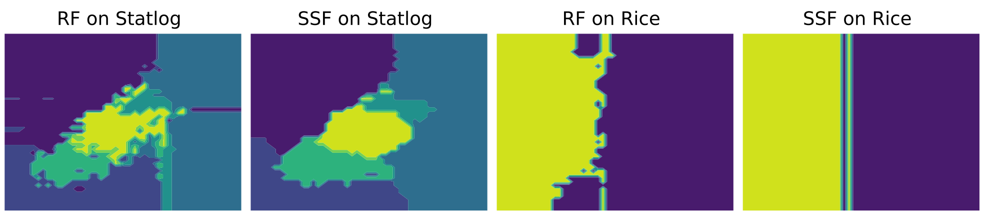

Fig. 3 Example of decision boundaries in data classification between random forests (RF) and the proposed splitting stump forests (SSF) on two-dimensional projections of two datasets. SSF achieves a comparable accuracy using only 0.002 of the total nodes employed by the RF method.

small subset, thus facilitating good prediction on the training set, but also may cause overfitting and can increase sensitivity to noise and outliers. Conversely, the other branch typically contains a large subset. This results in deeper trees, longer inference times, and reduced human readability [39, 40]. Moreover, this behavior can negatively impact prediction accuracy as the algorithm prioritizes outliers or errors less relevant in the generalization process creating overly specific rules based on limited information. In this study, we investigate the effect of choosing attribute-value combinations that result in balanced splits. These combinations of balanced splits empirically facilitate good data partitioning, thereby helping learners avoid overfitting [41]. Figure 3 shows the decision boundaries of our corresponding splitting stump forests. In these illustrative examples, selecting nodes that lead to balanced splits for SSF increases the sizes of the continuous regions while reducing overall model size and maintaining similar predictive performance of the resulting model.

4.2 Quantization of Selected Nodes

The test nodes selected in the previous step serve as the basis for the model which we will construct and explain in the following steps. However, a large number of these test nodes can significantly increase model complexity and computational cost, especially in embedded devices, where floating-point operations are expensive. To address this, we propose to use an optional node quantization technique that merges test nodes by rounding split values to a controlled level of decimal precision. By approximating split values, we effectively prune redundant test nodes, leading to a more compact and efficient model. Prior work by Koschel et al. [42] propose to use fixed-point quantization when using the QuickScorer algorithm on ARM devices. We adapt this approach to our setting focusing on test nodes: In particular, for all test nodes ($x_a \leq s_v$) selected in the previous step from the random forest model for a given attribute x_a and a split value $s_v \in R$, we apply a rounding function that maps splitting values to q decimal places: $s'_v = round(s_v, q)$ (Line 9). Nodes with identical rounded values are merged into a single node, reducing model complexity. Notably, scaling split values and data points by 10^q, transforms all split values into integers, enabling efficient integer-based comparisons on hardware. This significantly improves execution speed on embedded devices, where floating-point operations are computationally expensive.

4.3 Splitting Stump Transformation

So far, we have selected a set of test nodes, which currently lack subtrees or leaf nodes. More formally, $F[p]$, the result of the previous step, is a set of isolated vertices, each consisting of a feature and split value, but no children. By attaching two leaves to each node $v \in F[p]$, we transform v into the root of a decision tree T'_v of depth one (line 13). These decision trees can be viewed as learning a new data representation that maps a data point to a set of leaves. For each decision tree T'_v in the new ensemble F' of size k, we define a mapping function $f_{T'_v} : \mathbb{R}^d \to \{0, 1\}$ that returns one if and only if the split condition in node v evaluates to true on x (line 17). For F', we thus construct a function $f_{F'} : \mathbb{R}^d \to \{0, 1\}^k$ which maps x to a new feature vector $\{f_{T'_{vi}}(x)\}_{i=1}^{k}$. That is, each training point x is *embedded* into the concatenation of features (ones and zeros) resulting from the stumps (line 19). Using $f_{T'_v}$ instead of the two leaf features of T'_v is sufficient due to the perfect correlation between the two leaf features resulting from each stump T'_v.

4.4 Training of Splitting Stump Forests

To enable deployment to devices with limited resources, we use a linear model to combine the individual predictions of the decision trees in F'. We apply logistic regression to model the relationship between the new feature vectors $f_{F'}(x)$ and the target variable. The resulting model is both resource-efficient and easy to interpret. Conceptually, this step can be seen as simultaneously learning the leaf node assignments of all decision stumps and the voting scheme of the resulting random forest. Following the pruning of the random forest, similar post-training approaches have been shown to work well [18, 23]. Consider a single splitting stump T'_v: Training a logistic regression classifier on one-hot encoded representations $f_{T'_v}$ assigns a weight to each of the two dimensions that, for binary classification, corresponds to the likelihood of belonging to the target class. A similar approach works for regression tasks using a linear regression learner.

5 Experiments

We first evaluate the proposed method by analyzing its accuracy and efficiency. Then, in Section 6, we examine its deployment on embedded devices, focusing on memory usage and performance.

To assess the method's performance, we conducted experiments on 13 benchmark classification datasets with varying properties, primarily from the UCI repository (Adult, Letter Recognition, MAGIC, Spambase, Statlog, Waveform) [43], ALOI [44], Bank [45], Credit Card [46], Dry Bean [47], Rice [48], Room [49] and Shoppers [50]. This diverse selection enables the evaluation across varying complexities. Table 4 in the appendices presents details of the datasets. To evaluate the splitting stump forests approach (SSF), we perform a comparative analysis against the random forest (RF), the baseline cost complexity pruning (CCP) [35], and four state-of-the-art compression methods: the global refinement approach (LR) [23][2], the joint leaf refinement and

[2]available at github.com/gereleth/kaggle-telstra

ensemble pruning (LR+L1) [18][3], the diversity regularized ensemble pruning method (DREP) [19], and the individual error pruning method (IE) [20][4]. We train random forests, using the Gini index reduction for splitting, by varying the maximum depth d of individual decision trees among $d \in \{5, 10, 15\}$ and the number t of decision trees among $t \in \{16, 32, 64\}$. The same d and t values are adopted as in the SSF method for each approach being compared. To determine optimal parameters, we conduct a grid search for each method. We perform 5-fold cross-validation and report the average accuracy achieved on test data for each d. The threshold parameter p is set to values $\{0.05, 0.1, ..., 0.4, 0.45\}$. Code for SSF and all experiments is accessible online.[5]

5.1 Comparative Analysis of Performance and Efficiency

We report the average test accuracy, the number of nodes (test nodes and leaves), and the prediction time, also referred to as inference time, based on simulations conducted on a laptop with Intel Core i7-1165G7 processor. In section 6 we will discuss inference time on an Arduino embedded device. Table 2 presents a summary of this experiment, displaying the average ranking achieved by each method across the 13 datasets. This mean ranking demonstrates the consistent superiority of the SSF approach, with respective global rankings of 2.26, 1.23, and 1.77 concerning predictive performance, model size compression, and inference time. In terms of predictive performance, SSF outperforms original random forests (RF) and state-of-the-art methods in 22 out of 39 runs (across 13 datasets, each with three maximum depth values). In most other runs, SSF accuracy is within 1% of the top-performing model. SSF significantly reduces model size by two to three orders

Table 2 Average rankings of methods across 13 datasets based on accuracy, compression ratio, and inference time for each depth. Here, rank one is assigned to the best-performing method and rank seven to the worst. The last column shows the global ranking across all depths and datasets.

Method		d=5	d=10	d=15	Global
RF	Acc.	5	4.69	3.23	4.31
	Size	7	7	7	7
	Inf.	6.08	5.46	5.46	5.67
CCP	Acc.	6.46	6.62	6.85	6.64
	Size	4	2.38	1.85	2.74
	Inf.	4.69	3.54	3.46	3.90
DREP	Acc.	4.23	3.62	3.69	3.85
	Size	3.62	4.69	5	4.44
	Inf.	3.07	3.08	2.85	3
IE	Acc.	4	3.23	3.38	3.54
	Size	3.69	4.54	5.08	4.43
	Inf.	2.92	3.23	2.92	3.03
LR	Acc.	2.69	3.15	4.38	3.41
	Size	5.23	4.77	4.85	4.95
	Inf.	4.92	6.77	6.69	6.13
LR+L1	Acc.	2.23	2.77	2.69	2.56
	Size	3.38	3.46	3	3.28
	Inf.	6	4.85	5.15	5.33
SSF	Acc.	**2**	**2.38**	**2.39**	**2.26**
	Size	**1.23**	**1.15**	**1.30**	**1.23**
	Inf.	**1.69**	**1.69**	**1.92**	**1.77**

[3]available at github.com/sbuschjaeger/leaf-refinement-experiments
[4]both available at github.com/sbuschjaeger/PyPruning
[5]https://github.com/FouadAlkhoury/SplittingStumpsForests/

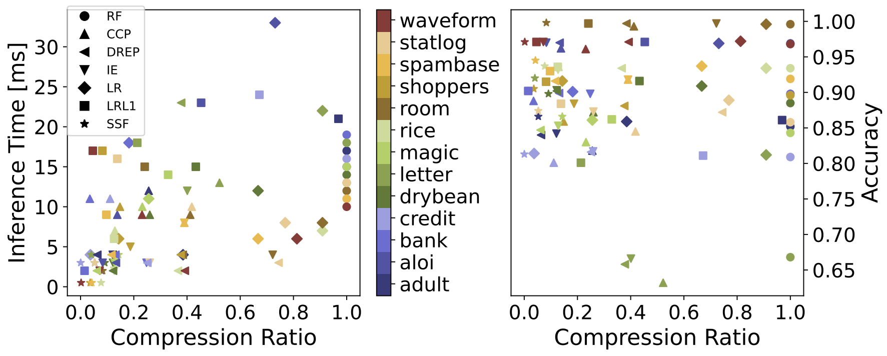

Fig. 4 The figure shows the highest test accuracy achieved with a maximum depth $d = 5$, along with its associated compression ratio and inference time, for each method and dataset.

of magnitude compared to other methods and achieves the best compression ratio in 32 out of 39 runs, achieving a global ranking of 1.23. Considering the mean ranking, SSF exhibits a marginal improvement in predictive performance compared to the LR+L1 method, while consistently excelling in reducing model size. Inference time for SSF is faster than the best-performing models of competing methods in 31 out of 39 runs, achieving a global ranking of 1.77. Scoring and selecting nodes in SSF training is efficiently done through a preorder tree traversal, with a time complexity of $O(n)$ where n is the number of nodes in the random forest if we store a record of the training examples that traverse edges during random forest training. Looking at Figure 4, we note that the SSF outperforms competing methods in model size and inference time across all datasets while achieving the best or second-best levels of accuracy. Detailed results on accuracy, compression, and inference time are in Tables 6, 7, 8 in the appendices. To isolate the effect of the linear model, we trained logistic regression directly on the original input features. Across all datasets, SSF achieved an average accuracy improvement of 3.5% over this baseline. This highlights the contribution of the selected splitting stumps beyond the linear classifier alone. Detailed results of this experiment are shown in Table 5 in the appendices.

5.2 Predictive Performance on a Space Budget

Motivated by space constraints on small embedded devices, we analyze the performance of SSF and the competitive ensemble pruning methods in a space budget. To that end, we explore various random forest configurations with $d \in \{5, 10, 15\}$, $t \in \{8, 16, 32, 64\}$, and the corresponding parameters for each competitive method, evaluating predictive performance and model size (node count) for each configuration. To accommodate devices with limited storage capacity, we select the best models that can fit within 32 KB or 16 KB of memory. Such models are suitable for deployment on microcontroller units like the Arduino Uno and ATmega169P. We estimate the model size using the baseline implementation of decision trees established by Buschjäger and Morik [9] and applied in subsequent studies [18]. This implementation indicates that each node requires $17+4C$ bytes of memory, where C represents the number of classes.

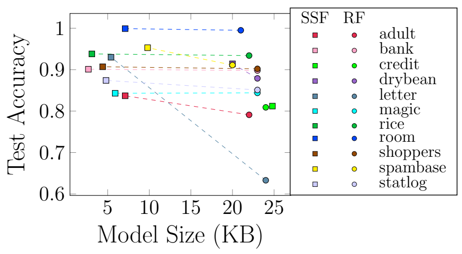

Fig. 5 The plot shows the best model attained by RF and SSF with a final model size below 32 KB.

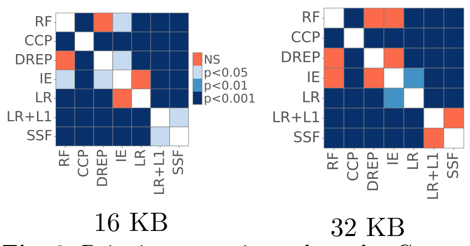

Fig. 6 Pairwise comparisons through a Conover test between the top-performing models under 16 KB (left) and 32 KB (right).

Figure 5 shows that SSF models outperform RF models across all datasets. Notably, this improvement exceeds 3% in five datasets. Particularly in multi-classification tasks like the letter recognition dataset (26 classes) and the dry bean dataset (7 classes), the improvement is significant, due to the complexity of multi-class problems [51]. In these tasks, deep random forests excel in capturing the complex decision boundaries necessary for reasonable accuracy. However, the best random forest model for the letter dataset requires 4 MB of memory, exceeding the IoT device's budget. Thus, we recommend using SSF models (of size below 10 KB in most datasets) for multi-classification tasks. Then, we employ the post-hoc Friedman test methodology as outlined by Demšar [52] for 32 KB and 16 KB budgets to check for statistically significant performance differences among the seven examined methods. We formulate the null hypothesis as all methods perform equally well without significant differences. The Friedman test ranks the methods for each dataset and each of 5 runs, assigning the top-performing method a rank of 1, the second-best a rank of 2, and so forth. This test determines whether the average ranks significantly deviate from the expected mean rank of 4. Average ranks provide a useful comparison of the methods, as illustrated in Table 9 in the appendices. Notably, the computed p-values for both 32 and 16 KB scenarios are 5.8×10^{-40} and 2.14×10^{-41} respectively, leading to the rejection of the null hypothesis at a highly significant level. As statistical significance is revealed, we apply a post-hoc procedure for multiple comparisons as proposed by García et al. [53]. Using the Conover Test, we conduct 21 pairwise comparisons among the seven methods, at confidence levels of 95%, 99%, and 99.9%. Fig. 6 demonstrates that both SSF and LR+L1 significantly deviate from the other 5 methods at the highest confidence level p-value < 0.001 in both the 32 and 16 KB scenarios. Moreover, the computed p-value between the SSF and LR+L1 is 1.1×10^{-2} in the 16 KB scenario, indicating that SSF outperforms LR+L1 with 95% confidence on memory-limited devices.

5.3 Compression on a Performance Budget

As a complementary experiment, we explore compression while tolerating a slight drop in accuracy. We identify the smallest SSF model within a 2% accuracy margin compared to the RF model with varying values of d and t. For a relatively small RF with $d = 5$ and $t = 16$, we achieve an average compression rate of 0.04 across all

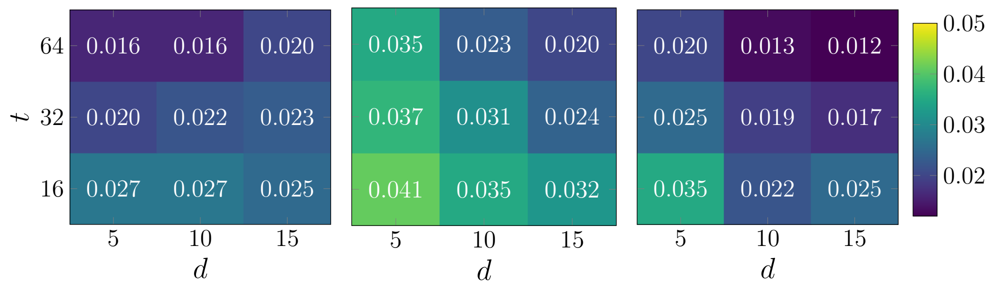

Fig. 7 The plot shows the compression ratio achieved in the datasets adult, shoppers, spambase (left to right) while permitting a 2% accuracy drop.

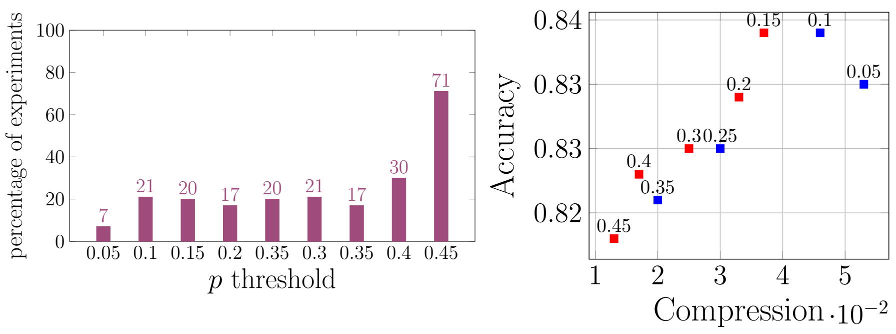

Fig. 8 The figure shows Pareto Frontier results for the min/max objectives compression ratio and accuracy. The left plot shows the percentage of experiments in which p is non-dominated by another p' in various problem settings $d \in \{5, 10, 15\}$, $t \in \{16, 32, 64\}$. Right, we exemplarily show the non-dominated thresholds in red, and dominated ones in blue on adult, $t = 64$, $d = 10$.

datasets. Moreover, as the random forest size increases, so does the compression rate for most datasets. Notably, the SSF method yields compression values of 0.012, 0.02 and 0.017 for the spambase, shoppers, and adult datasets, respectively, under the random forest configuration with $d = 15$ and $t = 64$; see Figure 7.

5.4 Optimizing Accuracy vs. Compression

To validate our assumption regarding the informativeness of nodes with highly balanced branches, and as our problem involves balancing a trade-off between the predictive performance and model compression, we investigate the impact of varying filtering thresholds p on the trade-off between the two objectives. In particular, we examine the Pareto frontier which enables us to concentrate on the set of efficient choices of p known as non-dominated solutions [54]. A Pareto-optimal filtering threshold p^* is where we cannot find another p that improves accuracy without sacrificing compression, or vice versa. We determine the set of Pareto-optimal points for each dataset and for each RF setting $d \in \{5, 10, 15\}$ and $t \in \{16, 32, 64\}$. Next, we compute the frequency of each threshold p in the Pareto-optimal set, focusing on data points achieving accuracy within 2% of the original RF accuracy. Omitting this step would result in any data point with the maximum threshold of 0.45 being incorrectly classified as Pareto-optimal, given the monotonically increasing nature of the compression function. The findings, as shown in Figure 8, indicate that the threshold $p = 0.45$ exhibits Pareto-optimality in 71% of the experiments, while $p = 0.40$ demonstrates Pareto-optimality in 30% of the cases. The other thresholds are Pareto-optimal in about 20%, except for $p = 0.05$ in only 7% of runs. Our findings support our assumption that test nodes with high splitting power, like those with $p = 0.45$, provide more information than nodes with low splitting power. These high-scoring nodes represent only a small fraction of the entire nodes set in the random forest, producing well-balanced branches, and resulting in a highly accurate and compact model.

We validate the informativeness of our selected nodes by comparing their predictive performance to that of a randomly selected sample of the same size. For a given dataset D, we report the accuracy and number of selected nodes n using the parameters: $d = 15$, $t = 64$, $p = 0.4$. Then we randomly sample n nodes from the entire node set i.e. this case corresponds to $p = 0.0$. These nodes are then transformed into splitting stumps, and we proceed to train a linear model using their data representation. To ensure experiment validity, we repeat sampling ten times and calculate the mean and standard deviation. We conduct two additional experiments: one where we sample n nodes that achieve a score better than $p = 0.2$, and another where we sample n nodes with scores less than $p = 0.1$. Comparing the predictive performance of score-based splitting stumps with $p = 0.4$ against sampling-based stumps, we find that score-based stumps tend to perform better across most datasets, as shown in Figure 9. These high-scoring stumps also outperform equivalent-sized sets of lower-scoring nodes, both those with scores of greater than $p = 0.2$, and those with scores less than $p = 0.1$, reinforcing our assumption that nodes with higher splitting power provide more information.

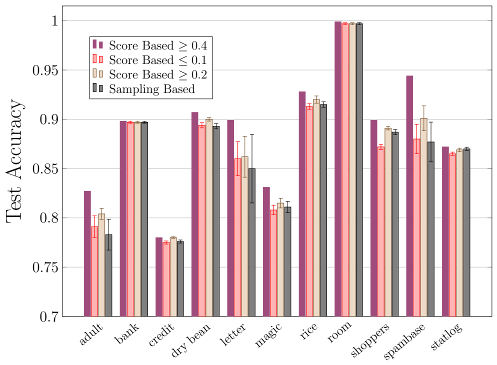

Fig. 9 Comparison of the predictive performance of the splitting stumps when $p = 0.4$, $p = 0.2$, sampling-based stumps of all scores, and low-scoring stumps ($\leq p = 0.1$). We report the mean and standard deviation of 10 random samples.

5.5 Impact of Stump Quantization

We conduct experiments across multiple datasets to evaluate the impact of decision stump quantization (cf. Section 4.2). Adjusting the decimal precision parameter q, we can control the complexity of the model according to hardware constraints and application needs. As we progressively reduce precision by keeping $q = 4, 3, 2, 1, 0$ decimal places of the split values, the number of unique stumps (and hence the SSF size) decreases, with only a slight drop in accuracy. Figure 10 shows the accuracy against the compression ratio of quantized models. Similarly, on Rice, accuracy holds up until $q = 3$, and on Waveform, rounding to $q = 2$ retains performance within the RF accuracy range. These results show that stump quantization significantly reduces model

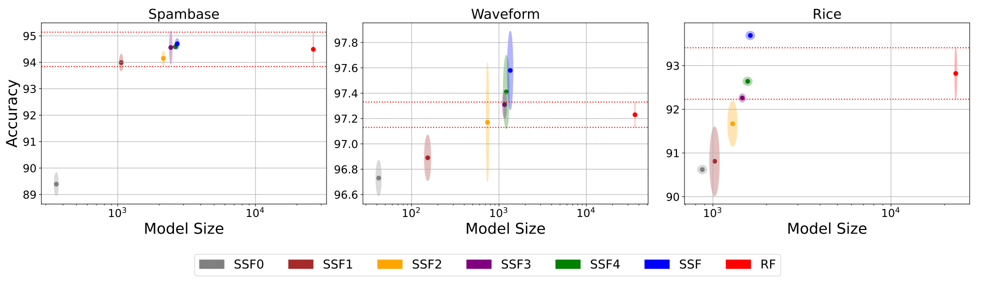

Fig. 10 Impact of stump refinement across multiple datasets (Spambase, Waveform, and Rice). The figure shows the trade-off between model size and accuracy as we progressively round the split values. It includes standard deviation in both accuracy and size (shown as error ellipses). The red dotted lines represents the RF accuracy ± standard deviation, highlighting the range within which refined models maintain comparable performance while reducing model size.

size without compromising accuracy, making it well-suited for resource-constrained environments.

6 Test Case: SSF Deployment on Arduino

To systemically explore the limits of SSF deployment on resource-constrained devices, we conduct experiments on the Arduino Mega-2560-R3 [55], which is limited to 8KB of SRAM. We test various configurations and record the highest achievable accuracy that can be achieved within the available memory constraints. During deployment, the Arduino receives the decision stumps, the Logistic Regression (LR) model parameters (weights and bias), and the test points. Each test point is first transformed into a new data representation based on the selected stumps before being processed by the LR model. The transformed data is then multiplied by the LR parameters to generate the final prediction. The memory footprint of an SSF model deployed on the Arduino is primarily influenced by the number of decision stumps and the number of classes in the dataset, both of which affect the storage requirements for the LR model. As the number of stumps increases, additional memory is required to store the corresponding weights and bias. Additionally, memory usage is affected by the number of test points and their feature dimensions. In our implementation, each decision stump requires 25 bytes for binary classification ($C = 2$), calculated as $17 + 4C$. In multi-class classification tasks where $C > 2$, the memory requirement per stump increases to $17 + 8C$ bytes. Additionally, feature representation impacts memory consumption, as all floating-point numbers are stored using 4 bytes per value. Consequently, each test point consisting of $|F|$ features requires an additional $4|F|$ bytes. Each model requires also a base overhead of around 300 bytes. The total RAM consumption for a binary classification task is therefore: $\text{RAM}_{\text{used}} = 300 + (17 + 4 \times C) \times |S| + 4 \times |F| \times |X|$ where C is the number of classes, $|S|$ is the number of decision stumps, $|F|$ is the number of features, and $|X|$ is the number of test points used during inference. Experimental results, illustrated in Table 3, demonstrate that SSF models consistently outperform RF models when deployed on devices with an 8 KB SRAM constraint. While the transformation of test points into the new data representation and the subsequent LR

Table 3 The table presents the highest-performing SSF model and RF model deployed on the Arduino Mega 2560 (8KB RAM), highlighting its accuracy, model size, time needed to transform a testing point using the constructed ensemble of stumps, inference time per test point, and energy consumption per prediction.

Dataset	SSF					RF			
	Size (KB)	Acc. (%)	Transform. (ms)	Inf. (ms)	Energy (mJ)	Size (KB)	Acc. (%)	Inf. (ms)	Energy (mJ)
shoppers	5.43	**91.10**	10.91	25.23	1.01	5.67	88.21	1.6	0.04
spambase	5.12	**93.56**	8.80	16.53	0.58	6.06	91.69	2.4	0.06
adult	5.44	**85.88**	11.13	19.16	0.69	5.54	82.27	1.8	0.04
drybean	6.55	**90.01**	4.72	27.49	0.68	6.21	78.84	2.9	0.07
letter	6.36	**65.28**	1.62	28.21	0.63	7.25	45.35	9.1	0.21
rice	4.95	**91.39**	10.08	21.44	0.69	5.84	91.07	1.4	0.03
room	5.64	**99.75**	5.59	23.42	0.61	5.28	98.82	2.1	0.05
magic	5.04	**85.28**	9.40	26.49	0.68	5.55	83.09	1.6	0.04
credit	4.33	**82.52**	8.33	17.93	0.53	5.84	81.76	1.8	0.04

prediction introduce additional computational overhead, accuracy remains the most critical factor in many real-world applications. Future work will focus on optimizing the SSF implementation to enhance efficiency and reduce inference time while maintaining its accuracy advantage. To measure energy consumption during inference, we employ a USB meter to record the power difference between the Arduino's idle and inference state. Power is calculated as $P = I \cdot V$, where I is current, and V is voltage, and energy per prediction as $E = P \cdot t$, where t is the inference time. Code for SSF deployment on Arduino is on github[6].

7 Conclusion

We introduced Splitting Stump Forests, an approach that extracts nodes from a trained random forest based on their splitting capabilities. Subsequently, we constructed decision trees for high-scoring nodes, that are optionally quantized by rounding the split values and trained a linear model over the derived data representations. Our extensive empirical tests indicate significant reductions in model size and improved inference speed without sacrificing accuracy across diverse datasets. We conducted a comprehensive comparison with competing methods and an ablation study of our split criterion. Moreover, a deployment study on an Arduino Mega-2560-R with only 8 KM of memory showed the applicability of our methods in real-world scenarios. Our encouraging experimental findings revealed our method's superiority in model size compression and inference time acceleration while maintaining a comparable level of predictive performance. These outcomes raise interesting directions for future research. In particular, to develop practical deployment strategies, ensuring that the benefits of model compression can be fully realized in real-world applications following the ongoing integration of machine learning models in edge devices.

[6]https://github.com/sbuschjaeger/mlgen3

8 Acknowledgements

FA and SB have been funded by the Federal Ministry of Education and Research of Germany and the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence, LAMARR24B. PW acknowledges support from the Vienna Science and Technology Fund (WWTF) project StruDL (ICT22-059).

References

[1] Vailshery, L.: Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033. www.statista.com (2024)

[2] Merenda, M., Porcaro, C., Iero, D.: Edge machine learning for AI-enabled IoT devices: A review. Sensors **20**(9), 2533 (2020)

[3] Filho, C.P., Marques, E., Chang, V., Santos, L., Bernardini, F., Pires, P.F., Ochi, L., Delicato, F.C.: A systematic literature review on distributed machine learning in edge computing. Sensors **22**(7), 2665 (2022)

[4] Hua, H., Li, Y., Wang, T., Dong, N., Li, W., Cao, J.: Edge computing with artificial intelligence: A machine learning perspective. ACM Computing Surveys **55**(9), 1–35 (2023)

[5] Murshed, M.S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., Hussain, F.: Machine learning at the network edge: A survey. ACM Computing Surveys **54**(8), 1–37 (2021)

[6] Dietterich, T.G.: Ensemble methods in machine learning. In: International Workshop on Multiple Classifier Systems, pp. 1–15 (2000)

[7] Sagi, O., Rokach, L.: Ensemble learning: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(4), 1249 (2018)

[8] Buschjäger, S., Morik, K.: Decision tree and random forest implementations for fast filtering of sensor data. Transactions on Circuits and Systems I: Regular Papers **65**(1), 209–222 (2017)

[9] Buschjäger, S., Morik, K.: Improving the accuracy-memory trade-off of random forests via leaf-refinement. arXiv preprint arXiv:2110.10075 (2021)

[10] Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001)

[11] Branco, S., Ferreira, A.G., Cabral, J.: Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey. Electronics **8**(11), 1289 (2019)

[12] Atmel: ATMEGA169P 8-bit AVR microcontroller with 16k bytes in-system

datasheet (2016). www.alldatasheet.com

[13] Atmel: Atsam3s2aa-au-mc 32bit 64kb lqfp-48 (2016). distrelec.de

[14] Alkhoury, F., Welke, P.: Splitting stump forests. In: International Conference on Discovery Science (DS) (2024). https://doi.org/10.1007/978-3-031-78980-9_1

[15] Partalas, I., Tsoumakas, G., Vlahavas, I.: Pruning an ensemble of classifiers via reinforcement learning. Neurocomputing **72**(7-9) (2009)

[16] Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: ICLR (2017)

[17] Peterson, A.H., Martinez, T.R.: Reducing decision tree ensemble size using parallel decision dags. International Journal on Artificial Intelligence Tools **18**(04), 613–620 (2009)

[18] Buschjäger, S., Morik, K.: Joint leaf-refinement and ensemble pruning through l1 regularization. Data Mining and Knowledge Discovery **37**(3), 1230–1261 (2023)

[19] Li, N., Yu, Y., Zhou, Z.-H.: Diversity regularized ensemble pruning. In: ECML PKDD, pp. 330–345 (2012)

[20] Jiang, Z., Liu, H., Fu, B., Wu, Z.: Generalized ambiguity decompositions for classification with applications in active learning and unsupervised ensemble pruning. In: AAAI Conference on Artificial Intelligence (2017)

[21] Nakamura, A., Sakurada, K.: An algorithm for reducing the number of distinct branching conditions in a decision forest. In: ECML PKDD, pp. 578–589 (2019)

[22] Esposito, F., Malerba, D., Semeraro, G., Kay, J.: A comparative analysis of methods for pruning decision trees. Transactions on Pattern Analysis and Machine Intelligence **19**(5), 476–491 (1997)

[23] Ren, S., Cao, X., Wei, Y., Sun, J.: Global refinement of random forest. In: Conference on Computer Vision and Pattern Recognition (2015)

[24] Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. Transactions on Pattern Analysis and Machine Intelligence **35**(8), 1798–1828 (2013)

[25] Nakano, F.K., Pliakos, K., Vens, C.: Deep tree-ensembles for multi-output prediction. Pattern Recognition **121**, 108211 (2022)

[26] Welke, P., Alkhoury, F., Bauckhage, C., Wrobel, S.: Decision snippet features. In: International Conference on Pattern Recognition, pp. 4260–4267 (2021)

[27] Vens, C., Costa, F.: Random forest based feature induction. In: International

Conference on Data Mining, pp. 744–753 (2011)

[28] Estruch, V., Ferri, C., Hernández-Orallo, J., Ramírez-Quintana, M.J.: Bagging decision multi-trees. In: Multiple Classifier Systems: International Workshop (2004)

[29] Buschjaeger, S., Chen, K.-H., Chen, J.-J., Morik, K.: Realization of random forest for real-time evaluation through tree framing. In: The IEEE International Conference on Data Mining Series (ICDM) (2018). https://ieeexplore.ieee.org/document/8594826

[30] Hakert, C., Chen, K.-H., Chen, J.-J.: Flint: Exploiting floating point enabled integer arithmetic for efficient random forest inference. In: 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1–2 (2024). IEEE

[31] Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Tonellotto, N., Venturini, R.: Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 73–82 (2015)

[32] Ye, T., Zhou, H., Zou, W.Y., Gao, B., Zhang, R.: Rapidscorer: Fast tree ensemble evaluation by maximizing compactness in data level parallelization. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 941–950 (2018)

[33] Lettich, F., Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Tonellotto, N., Venturini, R.: Parallel traversal of large ensembles of decision trees. IEEE Transactions on Parallel and Distributed Systems **30**(9), 2075–2089 (2018)

[34] Nakandala, S., Saur, K., Yu, G.-I., Karanasos, K., Curino, C., Weimer, M., Interlandi, M.: A tensor compiler for unified machine learning prediction serving. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pp. 899–917 (2020)

[35] Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Chapman and Hall, New York, NY, USA (1984)

[36] Quinlan, J.R.: Induction of decision trees. Machine Learning **1**, 81–106 (1986)

[37] Quinlan, J.R.: C4.5: Programs for machine learning. In: ICML (1993)

[38] Iba, W., Langley, P.: Induction of one-level decision trees. In: International Workshop on Machine Learning, pp. 233–240 (1992)

[39] Leroux, A., Boussard, M., Dès, R.: Information gain ratio correction: improving prediction with more balanced decision tree splits. arXiv preprint arXiv:1801.08310 (2018)

[40] Thakur, D., Markandaiah, N., Raj, D.S.: Re optimization of id3 and c4.5 decision tree. In: International Conference on Computer and Communication Technology, pp. 448–450 (2010)

[41] Bringmann, B., Zimmermann, A.: The chosen few: On identifying valuable patterns. In: International Conference on Data Mining (2007)

[42] Koschel, S., Buschjäger, S., Lucchese, C., Morik, K.: Fast inference of tree ensembles on arm devices. arXiv preprint arXiv:2305.08579 (2023)

[43] Dua, D., Graff, C.: UCI Machine Learning Repository (2017)

[44] Geusebroek, J., Burghouts, G., Smeulders, A.: The amsterdam library of object images. International Journal of Computer Vision **61**, 103–112 (2005)

[45] Moro, S., Cortez, P., Rita, P.: A data-driven approach to predict the success of bank telemarketing. Decision Support Systems **62**, 22–31 (2014)

[46] Yeh, I.-C., Lien, C.-h.: The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications **36**(2), 2473–2480 (2009)

[47] Koklu, M., Ozkan, I.A.: Multiclass classification of dry beans using computer vision and machine learning techniques. Computers and Electronics in Agriculture **174**, 105507 (2020)

[48] Cinar, I., Koklu, M.: Classification of rice varieties using artificial intelligence methods. International Journal of Intelligent Systems and Applications in Engineering **7**(3), 188–194 (2019)

[49] Singh, A.P., Jain, V., Chaudhari, S., Kraemer, F.A., Werner, S., Garg, V.: Machine learning-based occupancy estimation using multivariate sensor nodes. In: IEEE Globecom Workshops, pp. 1–6 (2018)

[50] Sakar, C.O., Polat, S.O., Katircioglu, M., Kastro, Y.: Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks. Neural Computing and Applications **31**(10), 6893–6908 (2019)

[51] Yan, J., Zhang, Z., Lin, K., Yang, F., Luo, X.: A hybrid scheme-based one-vs-all decision trees for multi-class classification tasks. Knowledge-Based Systems **198**, 105922 (2020)

[52] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research **7**, 1–30 (2006)

[53] García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests

for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Information Sciences **180**(10), 2044–2064 (2010)

[54] Lin, J.G.: Three methods for determining pareto-optimal solutions of multiple-objective problems. In: Directions in Large-Scale Systems: Many-Person Optimization and Decentralized Control (1976)

[55] Atmel: Arduino mega 2560 rev3 (2025). datasheet

A Detailed Experimental Results

Table 4 presents the details of the used datasets in our experiments. Table 5 compares the classification accuracy of logistic regression with that of the SSF model across different tree depths. Further insights are provided in Tables 6, 7, 8, which report test accuracy, compression ratio, and inference time (in ms) for each method and each dataset across three different depth values $d \in \{5, 10, 15\}$. RF Size denotes the number of nodes, and the compression ratio is calculated by dividing the number of nodes in the corresponding model by the original random forest size. Bold entries highlight the best values achieved per dataset. Finally, Table 9 summarizes the best accuracies achieved with a model size below 16 KB and 32 KB.

Table 4 We report sources, number of instances, number of attributes, number of classes, and dates for the datasets used.

Dataset	#instances	#attributes	#classes	Date
Adult [43]	48842	14	2	1996
ALOI [44]	50000	27	2	2005
Bank [45]	45211	17	2	2012
Credit Card [46]	30000	24	2	2009
Dry Bean [47]	13611	17	7	2020
Letter Rec. [43]	20000	16	26	1991
MAGIC [43]	19020	11	2	2007
Rice [48]	3810	8	2	2019
Room [49]	10129	16	4	2018
Shoppers [50]	12330	18	2	2019
Spambase [43]	4601	57	2	1999
Statlog [43]	6435	36	7	1993
Waveform [43]	3443	21	2	1988

Table 5 Classification accuracy of logistic regression and SSF at different tree depths ($d = 5$, $d = 10$, and $d = 15$).

Dataset	Log. Reg.	SSF ($d = 5$)	SSF ($d = 10$)	SSF ($d = 15$)
adult	0.828	0.866	0.861	0.858
aloi	0.97	0.97	0.97	0.971
bank	0.899	0.898	0.898	0.902
credit	0.808	0.811	0.814	0.814
drybean	0.847	0.898	0.914	0.907
letter	0.773	0.92	0.929	0.93
magic	0.805	0.836	0.839	0.831
rice	0.919	0.937	0.939	0.935
room	0.990	0.998	0.998	0.999
shopping	0.875	0.901	0.901	0.905
spambase	0.922	0.945	0.953	0.947
satlog	0.802	0.874	0.875	0.874
waveform	0.969	0.971	0.971	0.975

Table 6 Results attained by depth $d = 5$

Method		adult	aloi	bank	credit	drybean	letter	magic	rice	room	shoppers	spambase	statlog	waveform
RF	Acc.	85.2	96.9	89.8	80.9	88.5	66.8	84.3	93.4	99.6	89.6	91.9	85.8	96.8
	Size	3632	3701	3810	3924	3714	3800	3744	3608	3144	3722	3102	3836	3043
	Inf.	17	15	19	16	14	18	15	11	12	13	11	13	10
CCP	Acc.	81.8	96.2	88.8	80.1	87.2	63.2	83.0	93.6	99.3	85.9	91.8	84.5	96.1
	Size	0.256	0.138	0.035	0.111	0.26	0.522	0.232	0.129	0.412	0.148	0.39	0.418	0.231
	Inf.	12	9	11	11	9	13	10	7	9	10	8	10	9
DREP	Acc.	84.0	97.0	89.9	81.6	90.2	65.8	84.7	93.4	99.7	88.1	91.6	87.2	97.1
	Size	0.063	0.133	0.131	0.251	0.122	0.378	**0.061**	0.366	0.384	0.376	0.116	0.744	0.392
	Inf.	4	3	4	**3**	**2**	23	**2**	2	4	4	4	**3**	2
IE	Acc.	84.2	97.0	89.8	**81.7**	90.3	66.6	85.3	93.2	99.7	88.4	91.8	87.3	97.1
	Size	0.121	0.083	0.248	0.259	0.122	0.401	0.128	0.127	0.722	0.187	0.184	0.26	0.072
	Inf.	4	3	3	**3**	3	12	3	2	4	5	4	**3**	2
LR	Acc.	85.9	96.9	90.1	81.4	90.9	81.2	86.1	93.4	99.6	**91.6**	93.7	**88.9**	**97.2**
	Size	0.385	0.731	0.182	0.037	0.667	0.909	0.256	0.909	0.909	0.143	0.667	0.769	0.813
	Inf.	4	33	20	4	12	22	11	7	8	6	6	8	6
LR+L1	Acc.	86.1	**97.1**	**90.2**	81.1	**91.6**	80.1	86.2	93.6	99.7	91.5	93.0	88.4	97.1
	Size	0.969	0.453	**0.015**	0.672	0.433	0.213	0.329	0.126	0.241	0.082	0.097	0.138	0.046
	Inf.	21	23	**2**	24	15	18	14	6	15	17	9	16	17
SSF	Acc.	**86.6**	**97.1**	**90.2**	81.3	89.8	**92.0**	**86.6**	**93.7**	**99.8**	90.5	**94.5**	87.4	97.1
	Size	**0.053**	**0.079**	**0.015**	**0.001**	**0.091**	**0.04**	0.143	**0.077**	**0.083**	**0.036**	**0.042**	**0.053**	**0.002**
	Inf.	**3**	**2**	**2**	**3**	3	**4**	4	**<1**	**2**	**<1**	**<1**	**3**	**<1**

Table 7 Results attained by depth $d = 10$

Method		adult	aloi	bank	credit	drybean	letter	magic	rice	room	shoppers	spambase	statlog	waveform
RF	Acc.	85.8	96.9	89.8	81.1	92.5	89.3	87.0	93.6	99.7	90.9	93.6	89.5	97.1
	Size	34628	38256	45192	50670	32990	48142	35846	17288	7482	35780	15570	26692	38256
	Inf.	25	21	22	17	15	20	16	11	13	14	12	14	8
CCP	Acc.	81.9	95.8	88.8	80.6	87.4	69.7	83.5	93.7	99.4	85.9	92.2	85.3	97.0
	Size	0.031	0.013	**0.003**	0.009	0.030	0.081	0.029	0.035	0.173	0.018	0.102	0.079	0.089
	Inf.	12	9	11	10	10	14	10	8	9	9	8	10	11
DREP	Acc.	85.8	96.9	90.0	81.3	91.2	88.2	87.2	**93.9**	99.8	89.8	94.8	**91.1**	97.2
	Size	0.378	0.278	0.136	0.119	0.204	0.392	0.384	0.741	0.396	0.369	0.719	0.379	0.063
	Inf.	16	9	14	**7**	7	33	6	2	4	5	6	3	**2**
IE	Acc.	85.7	96.9	90.0	81.4	90.0	90.5	87.2	**93.9**	**99.9**	90.4	94.9	**91.1**	**97.5**
	Size	0.521	0.321	0.509	0.956	0.202	0.221	0.224	0.255	0.164	0.512	0.385	0.063	0.481
	Inf.	17	15	32	12	**4**	15	5	13	3	3	5	**2**	2
LR	Acc.	85.9	97.1	90.1	81.2	92.3	93.4	87.2	93.7	99.7	**91.3**	94.3	89.3	97.3
	Size	0.053	0.833	0.022	0.016	0.526	0.909	0.813	0.083	0.909	0.167	0.091	0.556	0.909
	Inf.	39	38	34	20	20	38	29	13	18	19	12	15	13
LR+L1	Acc.	86.5	**97.2**	**90.2**	80.3	**93.0**	**94.6**	**87.6**	93.6	99.8	90.9	94.0	91.0	97.2
	Size	0.843	0.191	0.015	0.531	0.294	0.128	0.205	0.074	0.093	0.063	0.135	0.209	**0.031**
	Inf.	14	12	**2**	26	14	16	14	7	11	22	12	22	9
SSF	Acc.	**86.1**	**97.2**	90.1	**81.5**	91.4	92.9	87.3	**93.9**	**99.9**	90.1	**95.3**	87.5	97.1
	Size	**0.013**	**0.001**	0.014	**0.002**	**0.025**	**0.004**	**0.021**	**0.027**	**0.041**	**0.017**	**0.013**	**0.046**	0.032
	Inf.	**6**	**2**	10	13	6	**5**	**4**	**<1**	**2**	**2**	**<1**	6	**2**

Table 8 Results attained by depth $d = 15$

Method		adult	aloi	bank	credit	drybean	letter	magic	rice	room	shopper	spambase	statlog	waveform
RF	Acc.	86.0	97.0	89.9	81.0	92.6	**99.1**	**87.7**	92.8	99.8	91.0	94.5	90.6	97.2
	Size	119538	56102	186530	177988	67334	166128	100982	23092	8324	87148	26312	42204	36121
	Inf.	32	17	25	25	16	24	18	12	14	16	15	15	18
CCP	Acc.	82.3	94.6	88.8	80.7	87.2	69.6	83.8	92.8	99.2	86.9	91.9	86.1	95.1
	Size	**0.008**	0.017	**0.001**	0.004	0.015	0.024	0.021	**0.018**	0.162	**0.006**	0.058	0.051	0.021
	Inf.	17	11	12	11	10	14	10	8	9	14	9	9	11
DREP	Acc.	85.7	97.2	90.1	81.1	90.4	95.5	87.6	93.6	99.8	90.6	94.2	90.2	97.5
	Size	0.195	0.325	0.189	0.511	0.758	0.381	0.374	0.182	0.497	0.751	0.489	0.376	0.121
	Inf.	24	5	29	**9**	5	28	8	**2**	3	6	5	3	**2**
IE	Acc.	86.0	97.1	90.0	81.4	90.5	96.6	87.6	93.5	99.7	90.6	94.4	90.1	97.5
	Size	0.513	0.162	0.136	0.741	0.517	0.423	0.769	0.377	0.162	0.384	0.374	0.504	0.251
	Inf.	41	8	**11**	19	**3**	16	11	3	4	**5**	6	**2**	3
LR	Acc.	85.9	97.0	90.0	81.3	92.1	96.3	86.1	92.9	99.6	90.5	94.5	89.2	97.3
	Size	0.031	0.431	0.101	0.007	0.256	0.901	0.388	0.206	0.901	0.729	0.431	0.909	0.078
	Inf.	70	73	32	29	20	80	46	16	20	32	16	26	14
LR+L1	Acc.	**86.1**	**97.4**	**90.2**	80.5	**93.6**	96.5	87.6	93.3	99.8	91.0	94.3	**91.1**	97.2
	Size	0.023	0.033	0.028	0.044	0.173	0.101	0.164	0.069	0.071	0.052	0.112	0.168	0.029
	Inf.	22	38	20	34	29	18	14	6	10	21	11	29	5
SSF	Acc.	**86.1**	97.3	**90.2**	**81.5**	90.7	93.0	86.1	**93.7**	**99.9**	**91.2**	**94.7**	87.4	**97.6**
	Size	0.014	**0.001**	0.005	**0.002**	**0.012**	**0.001**	**0.01**	0.037	**0.025**	0.034	**0.015**	**0.045**	**0.001**
	Inf.	**16**	**3**	**11**	24	11	**7**	**7**	**2**	**2**	**5**	**<1**	**2**	**2**

Table 9 The table shows the best accuracy with a model size below 16 KB and 32 KB. Bold entries indicate the best method for each dataset. The last line shows the average ranking of the accuracy of each method across all datasets.

	16 KB							32 KB						
Dataset	RF	CCP	DREP	IE	LR	LRL1	SSF	RF	CCP	DREP	IE	LR	LRL1	SSF
adult	82	81.9	80.5	82.9	84.4	85.7	**86.1**	85.1	82.3	83.4	84.3	85.9	**86.2**	86.1
	±0.4	±0.3	±0.2	±0.6	±0.4	±0.3	±0.3	±0.3	±0.4	±0.4	±0.5	±0.3	±0.3	±0.3
aloi	96.7	96.2	96.9	96.9	96.1	97.0	**97.1**	96.8	96.2	96.9	97	96.1	**97.1**	**97.1**
	±0.4	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1	±0.1
bank	89.9	88.9	89.8	89.7	90.0	**90.2**	90.1	90	88.9	89.8	89.7	90.1	**90.4**	90.1
	±0.1	±0.5	±0.1	±0.1	±0.3	±0.3	±0.2	±0.1	±0.3	±0.1	±0.1	±0.2	±0.2	±0.1
credit	81	80.7	80.8	81.1	**81.4**	81.1	81.1	80.9	80.7	81	81.3	**81.4**	80.9	81.2
	±0.2	±0.3	±0.1	±0.2	±0.3	±0.1	±0.2	±0.1	±0.2	±0.1	±0.1	±0.2	±0.1	±0.1
dry bean	88.1	85.4	89.1	88.7	89.2	**91.8**	91.4	87.9	87.4	89.3	89.9	89.4	**91.8**	91.4
	±0.7	±0.3	±0.1	±0.3	±0.3	±0.1	±0.4	±1.0	±0.4	±0.5	±0.6	±0.5	±0.3	±0.4
letter	62.5	64.3	62.6	62.1	62.9	76.3	**93.0**	63.3	61.2	65.9	65.8	78.2	71.4	**93.0**
	±2.2	±1.1	±0.9	±1.2	±0.9	±0.9	±1.7	±3.6	±1.4	±1.0	±1.7	±1.5	±1.4	±1.9
magic	84.9	83.2	83.7	84.2	84.9	85.8	**86.3**	86	83.5	83.8	83.9	86.1	**86.5**	86.3
	±0.5	±0.9	±0.5	±0.5	±0.01	±0.2	±0.7	±0.7	±0.7	±0.3	±0.5	±0.6	±0.2	±0.4
rice	92.5	93.7	93.1	0.93	93.2	93.5	**93.8**	93.4	93.7	93.6	93.5	93.2	93.5	**93.8**
	±0.6	±0.7	±0.3	±0.4	±0.7	±0.1	±0.1	±0.7	±0.5	±0.2	±0.3	±0.6	±0.1	±0.1
room	99.2	97.0	99.5	99.6	99.2	99.8	**99.9**	99.5	99.3	99.7	**99.9**	99.2	99.8	**99.9**
	±0.3	±0.4	±0.1	±0.2	±0.6	±0.2	±0.2	±0.1	±0.2	±0.1	±0.2	±0.3	±0.1	±0.1
shoppers	87.2	86.9	90.1	91.0	**91.6**	91.3	90.7	90.2	86.9	90.3	91.2	91.6	**91.3**	90.7
	±1.5	±0.6	±0.3	±0.2	±0.4	±0.3	±0.4	±1.2	±0.2	±0.4	±0.1	±0.4	±0.2	±0.2
spambase	90.8	91.3	90.9	92.4	91.6	92.7	**95.3**	91.1	91.7	92.9	92.4	94.3	93.2	**95.3**
	±0.6	±0.4	±0.2	±1.0	±0.5	±0.2	±0.4	±0.6	±0.6	±0.6	±1.2	±0.4	±0.2	±0.2
statlog	85.2	84.1	84.7	84.8	84.9	86.5	**87.4**	85.1	84.9	84.8	84.7	87.1	**87.9**	87.4
	±1.6	±1.0	±0.4	±0.4	±0.8	±0.2	±0.3	±0.8	±0.6	±0.7	±0.6	±0.5	±0.2	±0.1
waveform	96.9	95.1	96.9	96.9	96.9	97.0	**97.1**	96.6	95.1	97	97	96.9	**97.2**	97.1
	±0.2	±0.2	±0.1	±0.1	±0.3	±0.1	±0.1	±0.2	±0.1	±0.1	±0.1	±0.2	±0.1	±0.1
avg rank	4.79	5.86	4.79	4.17	3.94	2.09	**1.54**	4.72	6.32	4.45	4.28	3.51	2.1	**1.88**

B Parameters selection across methods

B.1 Cost Complexity Pruning Method

To identify the optimal parameters for the CCP baseline method, we conduct a grid search involving the parameter $ccp\_alpha \in \{0.005, 0.01, 0.015, 0.02\}$ and $min\_samples\_leaf \in \{1, 5, 10, 20\}$. It's worth noticing that increasing the ccp_alpha value increases the number of pruned nodes while $min\_samples\_leaf$ defines the minimum data points required in a leaf node.

B.2 Diversity Regularized Ensemble Pruning

We varied the balance hyperparameter $\rho \in \{0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$ that controls the fraction of classifiers considered when minimizing the empirical error. Larger values of ρ put more emphasis on the empirical error, while a small ρ pays more attention on the diversity.

B.3 Individual Error Pruning

We varied the hyperparameter $n_l \in \{2^6, 2^7, 2^8, 2^9\}$ that controls the maximum number of leaf nodes in each individual tree in the random forest.

B.4 Global refinement of random forest

We varied the hyperparameter $n_t \in \{2^3, 2^4, 2^5\}$ that controls the number of selected trees from the random forest. Each pruning process ran for 25 and 50 epochs.

B.5 Joint leaf-refinement and ensemble pruning through L1 regularization

We varied the number of selected trees from the random forest hyperparameter $n_t \in \{2^3, 2^4, 2^5, 2^6\}$. We ran the experiment for 25 and 50 epochs. The regularization strength varied between $\{0.1, 0.2, ..., 0.9\}$.