Splitting Stump Forests: Tree Ensemble Compression for Edge Devices

Fouad Alkhoury[1,3][0000−0001−9470−4890] and Pascal Welke[2][0000−0002−2123−3781]

[1] University of Bonn, Germany
alkhoury@iai.uni-bonn.de
[2] TU Wien, Austria
pascal.welke@tuwien.ac.at
[3] Lamarr Institute for Machine Learning and Artificial Intelligence, Germany

Abstract. We introduce Splitting Stump Forests – small ensembles of weak learners extracted from a trained random forest. The high memory consumption of random forest ensemble models 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 20 billion units by 2025 [41]. Many IoT devices require real-time decisions and therefore include limited computing capabilities [28]. Running machine learning models directly on embedded devices is increasingly popular due to improvements in reliability, affordability, and energy efficiency [17]. Local models also reduce or even eliminate the need for transferring data to cloud servers when connectivity, bandwidth consumption, communication costs, network latency, and privacy are significant concerns [20, 30].

Ensemble models can outperform the predictive performance of individual classifiers in many machine learning tasks [13, 37]. 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 slow 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 [5]. 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. High-performing random forests often exceed 100K nodes (cf. Table 4 in the supplementary material). As a result, these models may demand 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. 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. Our extensive experimental evaluation on various datasets shows 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. 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. Section 3 provides a detailed description of our method. Section 4 describes our empirical evaluation before Section 5 concludes. Code for the splitting stump forests is available on github.

Table 1: Flash Memory on different microcontroller units [1, 2, 4]

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

2 Background

Our proposed random forest compression technique can be alternatively viewed as model compression or representation learning approach. We now review related work in these fields.

Ensemble Compression The existing methods for ensemble size reduction can be categorized into blackbox and whitebox approaches. Blackbox 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 whitebox approaches.

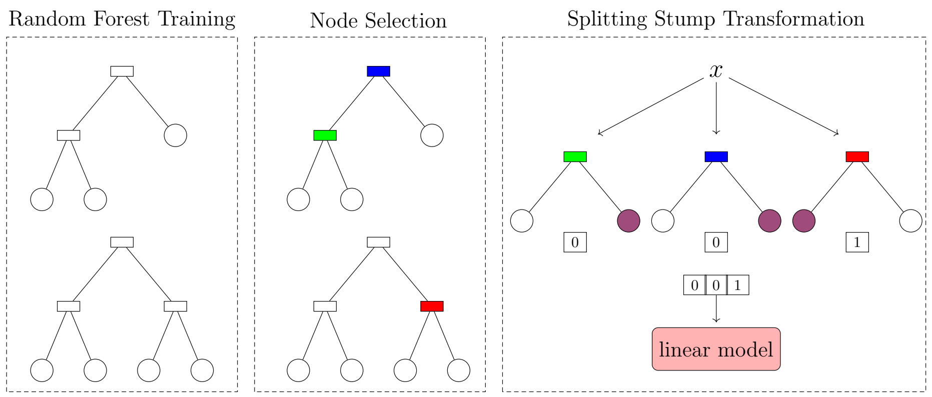
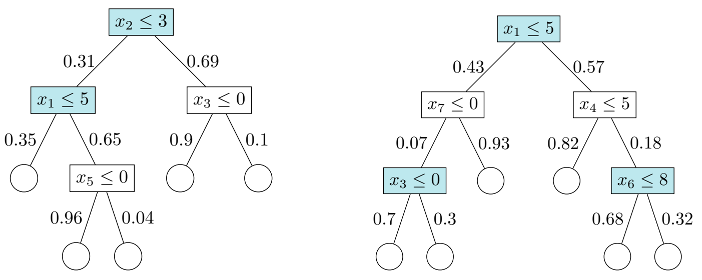


Fig. 1: Splitting stump forests at a glance. Given a random forest (left), selected nodes are selected (middle) and used as stumps (right). A training example is then transformed into a binary vector by the stumps and fed to a linear model.

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 [33]. Thus, most approaches identify a sub-ensemble with near-optimal performance. Several studies have been conducted on the pruning of machine learning models [25]. Moreover, considerable efforts concentrated on identifying subsets of random forest nodes that can match the original forest's accuracy. Peterson and Martinez [34] introduced a post-training technique that stores unique subtrees and combines redundant nodes into "parallel nodes" while maintaining the overall behavior. Buschjäger and Morik [10] 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. [26] 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 [22]. Nakamura and Sakurada [31] 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 [15]. In contrast, our approach constructs a new ensemble that contains more but smaller trees than the original ensemble.

Representation Learning Decision trees and random forests can be interpreted as representation learning [3], with studies exploring the use of a pre-trained random forest's transformed space as input for linear models. Ren et al. [36] enhance the fitting power of a tree ensemble model by global leaf value refinement using linear regression. Through global optimization, the approach iteratively merges



(a) T_1 (b) T_2

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 to true (false). Selected nodes in $T[p]$ for filtering threshold $p = 0.2$, are colored in blue.

insignificant pairs of adjacent leaves, effectively using complementary information from multiple trees and reducing model size. Likewise, Nakano et al. [32] 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. [43] identify frequent subtrees in a trained random forest and train a linear model on the resulting multi-hot leaf representation. Vens and Costa [42] 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. [16] 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.

3 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. See, e.g., Breiman [5] or Quinlan [35] for a background on random forest and decision tree algorithms. Given a trained random forest, we (1) compute a score for each node and select nodes with high scores (Section 3.1). Our selection technique prioritizes split values that yield balanced subtrees. Subsequently, we (2) construct one decision tree per selected node and form an ensemble (Section 3.2). Finally, our approach (3) integrates a representation learning module by using the transformed space derived from the constructed ensemble as input for a linear model (Section 3.3). Figure 1 shows the pipeline of the three primary steps,

Algorithm 1 Splitting Stump Forest Transformation

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

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

which we will describe in turn. The pseudo-code of the first two steps is presented in Algorithm 1. In Section 3.4 we discuss balanced splits in more detail.

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 $f : X \rightarrow \mathbb{R}$ such that the difference between $f(x)$ and the true label y is minimal for all $x \in X$. 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. A random forest classifier $F = \{T_j | j \in [1, t]\}$ consists of a set of t decision trees and a method to combine individual predictions, e.g., majority vote.

3.1 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 (resp. false) using the split condition at node v (Line 5 of Alg. 1). To qualify as balanced, v should attain a score that meets or exceeds a predetermined threshold p (Lines 6–7). Subsequently, we define $T[p]$ as the set of all nodes v with $score(v) \geq p$ (Line 8). In deep random forests, we can limit the size of $T[p]$ 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 $T[p]$.

Note that a scoring function for decision stump learning with a similar formula was introduced by Iba and Langley [21]. 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.

3.2 Splitting Stump Transformation

$T[p]$, the result of the previous step, is a set of isolated vertices, each consisting of a feature and split value, and is hence not a random forest by our definition. By attaching two leaves to each node $v \in T[p]$, we transform v into the root of a decision tree T'_v of depth one (line 10). 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 length 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 (See line 14). 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 16). 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.

3.3 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

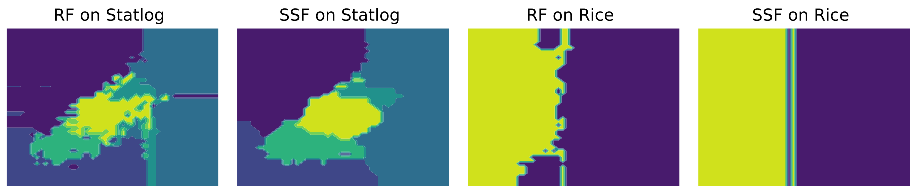


Fig. 3: Example of decision boundaries in data classification between random forests (RF) and the proposed splitting stump forests (SSF) on two-dimensional projections of two datasets. SSF achieves a comparable accuracy using only 0.002 of the total nodes employed by the RF method.

random forest. Following the pruning of the random forest, similar post-training approaches have been shown to work well [10, 36]. 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.

3.4 Further Details on Splitting Node Selection

The most common way to train random forests is based on bagging the training data and then using a recursive algorithm 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 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 [24, 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 [7]. 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 Experiments

Datasets We experiment on 13 benchmark classification datasets with varying properties, primarily from the UCI repository (Adult, Letter Recognition, MAGIC, Spambase, Statlog, Waveform) [14], ALOI [19], Bank [29], Credit Card [45], Dry Bean [23], Rice [11], Room [39] and Shoppers [38]. This diverse selection enables the evaluation across varying complexities.

Competitive Methods To evaluate the splitting stump forests approach (SSF), we perform a comparative analysis against the random forest (RF), the baseline cost complexity pruning (CCP) [6], and four state-of-the-art compression methods: the global refinement approach (LR) [36][4], the joint leaf refinement and ensemble pruning (LR+L1) [10][5], the diversity regularized ensemble pruning method (DREP) [26], and the individual error pruning method (IE) [22][6].

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, two to the second-best, and so on. 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**

Experimental Setting 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,

[4] available at github.com/gereleth/kaggle-telstra
[5] available at github.com/sbuschjaeger/leaf-refinement-experiments
[6] both available at github.com/sbuschjaeger/PyPruning

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.[7]

Comparison of Methods We report the average test accuracy, the number of nodes (test nodes and leaves), and the prediction time, also referred to as inference time. 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 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.

4.1 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 [10]. This implementation indicates that each node requires $17 + 4C$ bytes of memory, where C represents the number of classes.

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

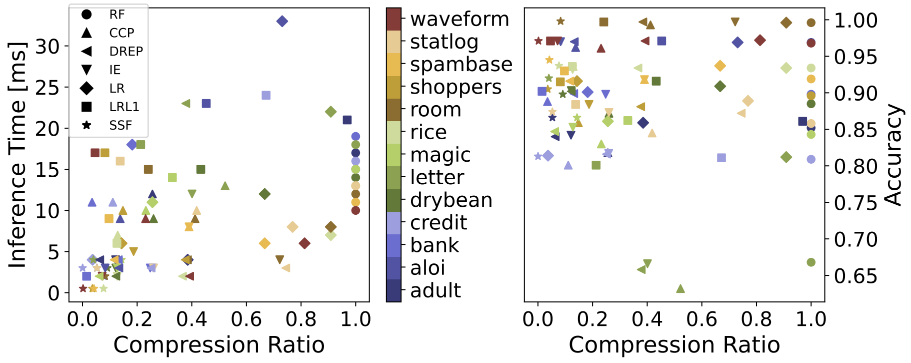


Fig. 4: The figure shows the highest test accuracy achieved with a maximum depth $d = 5$, along with its associated compression ratio and inference time, for each method and dataset.

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 [44]. 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 [12] 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 3. 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. [18]. 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.

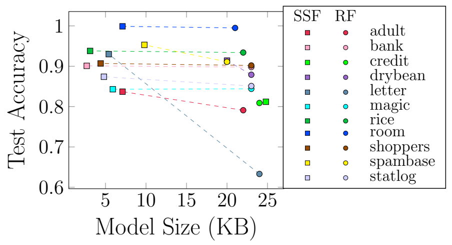
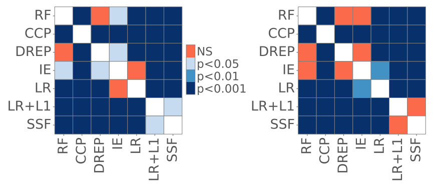


Fig. 5: The plot shows the best model attained by RF and SSF with a final model size below 32 KB.



(a) 16 KB

(b) 32 KB

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

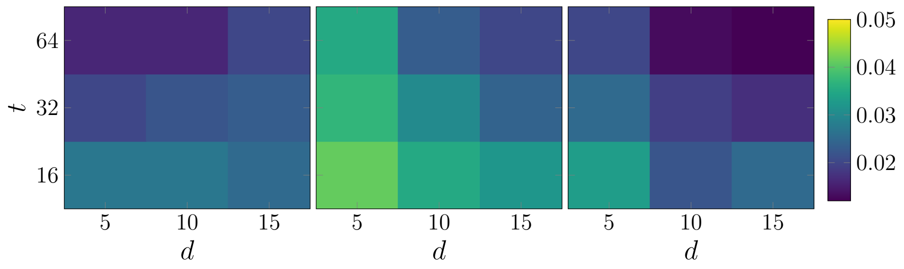


Fig. 7: The plot shows the compression ratio achieved in the datasets adult, shoppers, spambase (left to right) while permitting a 2% accuracy drop.

4.2 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 achieved an average compression rate of 0.04 across all datasets. Moreover, as the random forest size increased, so did the compression rate for most datasets. Notably, the SSF method yielded compression values of $0.012, 0.02, 0.017, 0.025, 0.037$, and 0.005 for the datasets spambase, shoppers, adult, room, rice, and bank, respectively; see Figure 7.

4.3 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 [27].

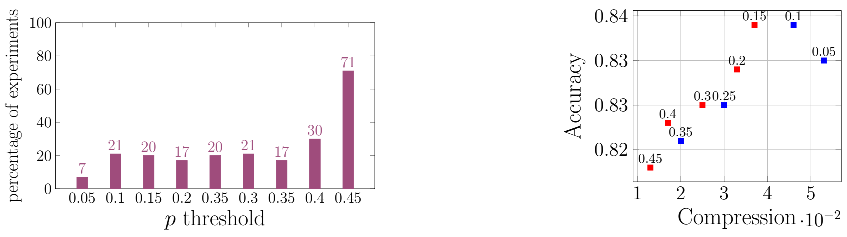


Fig. 8: The figure shows Pareto Frontier results for the min/max objectives compression ratio and accuracy. The left plot shows the percentage of experiments in which p is non-dominated by another p' in various problem settings $t \in \{16, 32, 64\},\ d \in \{5, 10, 15\}$. Right, we exemplarily show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

Table 3: 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**

A non-dominated filtering threshold p^* is where we cannot find another p that improves accuracy without sacrificing compression, or vice versa. We determine the set of non-dominated points for each dataset and for each RF setting $t \in \{16, 32, 64\}$ and $d \in \{5, 10, 15\}$. Next, we compute the frequency of each threshold p in the non-dominated 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 non-dominated, given the monotonically increasing nature of the compression function. The findings, as shown in Figure 8, indicate that the threshold $p = 0.45$ exhibits non-dominance in 71% of the experiments, while $p = 0.40$ demonstrates non-dominance in 30% of the cases. The other thresholds are non-dominant 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 also sample n nodes that achieve a score better than $p = 0.2$. 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 an equivalent-sized set of lower-scoring nodes of $p = 0.2$, reinforcing our assumption that nodes with higher splitting power provide more information.

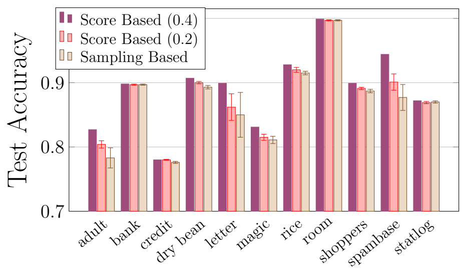


Fig. 9: Comparison of the predictive performance of the splitting stumps when $p = 0.4$, $p = 0.2$, and sampling-based splitting stumps of all scores. We report the mean and standard deviation of 10 random samples.

5 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 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. 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.

Acknowledgement FA has 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 is financed by the Vienna Science and Technology Fund (WWTF) project StruDL (ICT22-059).

References

1. Atmel: ATMEGA169P 8-bit AVR microcontroller with 16k bytes in-system datasheet (2016), www.alldatasheet.com
2. Atmel: Atsam3s2aa-au-mc 32bit 64kb lqfp-48 (2016), distrelec.de
3. 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)
4. 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)
5. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001)
6. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Chapman and Hall, New York, NY, USA (1984)
7. Bringmann, B., Zimmermann, A.: The chosen few: On identifying valuable patterns. In: International Conference on Data Mining (2007)
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. Buschjäger, S., Morik, K.: Joint leaf-refinement and ensemble pruning through l1 regularization. Data Mining and Knowledge Discovery **37**(3), 1230–1261 (2023)
11. 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)

12. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research **7**, 1–30 (2006)
13. Dietterich, T.G.: Ensemble methods in machine learning. In: International Workshop on Multiple Classifier Systems, pp. 1–15 (2000)
14. Dua, D., Graff, C.: UCI machine learning repository (2017)
15. 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)
16. Estruch, V., Ferri, C., Hernández-Orallo, J., Ramírez-Quintana, M.J.: Bagging decision multi-trees. In: Multiple Classifier Systems: International Workshop (2004)
17. Filho, C.P., Marques, E., Chang, V., dos 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)
18. 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)
19. Geusebroek, J., Burghouts, G., Smeulders, A.: The amsterdam library of object images. International Journal of Computer Vision **61**, 103–112 (2005)
20. 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)
21. Iba, W., Langley, P.: Induction of one-level decision trees. In: International Workshop on Machine Learning, pp. 233–240 (1992)
22. 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)
23. 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)
24. Leroux, A., Boussard, M., Dès, R.: Information gain ratio correction: improving prediction with more balanced decision tree splits. arXiv preprint arXiv:1801.08310 (2018)
25. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (2017)
26. Li, N., Yu, Y., Zhou, Z.H.: Diversity regularized ensemble pruning. In: European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 330–345 (2012)
27. 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)
28. Merenda, M., Porcaro, C., Iero, D.: Edge machine learning for AI-enabled IoT devices: A review. Sensors **20**(9), 2533 (2020)

29. Moro, S., Cortez, P., Rita, P.: A data-driven approach to predict the success of bank telemarketing. Decision Support Systems **62**, 22–31 (2014)
30. 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)
31. Nakamura, A., Sakurada, K.: An algorithm for reducing the number of distinct branching conditions in a decision forest. In: European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 578–589 (2019)
32. Nakano, F.K., Pliakos, K., Vens, C.: Deep tree-ensembles for multi-output prediction. Pattern Recognition **121**, 108211 (2022)
33. Partalas, I., Tsoumakas, G., Vlahavas, I.: Pruning an ensemble of classifiers via reinforcement learning. Neurocomputing **72**(7-9) (2009)
34. 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)
35. Quinlan, J.R.: Induction of decision trees. Machine Learning **1**, 81–106 (1986)
36. Ren, S., Cao, X., Wei, Y., Sun, J.: Global refinement of random forest. In: Conference on Computer Vision and Pattern Recognition (2015)
37. Sagi, O., Rokach, L.: Ensemble learning: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(4), e1249 (2018)
38. 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)
39. 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)
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. Vailshery, L.: Number of internet of things (IoT) connected devices worldwide from 2019 to 2030, by vertical (2023), www.statista.com
42. Vens, C., Costa, F.: Random forest based feature induction. In: International Conference on Data Mining, pp. 744–753 (2011)
43. Welke, P., Alkhoury, F., Bauckhage, C., Wrobel, S.: Decision snippet features. In: International Conference on Pattern Recognition, pp. 4260–4267 (2021)
44. 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)
45. 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)