Data Mining and Knowledge Discovery (2023) 37:1230–1261
https://doi.org/10.1007/s10618-023-00921-z

Joint leaf-refinement and ensemble pruning through L_1 regularization

Sebastian Buschjäger[1] · **Katharina Morik**[1]

Received: 10 December 2021 / Accepted: 5 January 2023 / Published online: 15 March 2023

Abstract

Ensembles are among the state-of-the-art in many machine learning applications. With the ongoing integration of ML models into everyday life, e.g., in the form of the Internet of Things, the deployment and continuous application of models become more and more an important issue. Therefore, small models that offer good predictive performance *and* use small amounts of memory are required. Ensemble pruning is a standard technique for removing unnecessary classifiers from a large ensemble that reduces the overall resource consumption and sometimes improves the performance of the original ensemble. Similarly, leaf-refinement is a technique that improves the performance of a tree ensemble by jointly re-learning the probability estimates in the leaf nodes of the trees, thereby allowing for smaller ensembles while preserving their predictive performance. In this paper, we develop a new method that combines both approaches into a single algorithm. To do so, we introduce L_1 regularization into the leaf-refinement objective, which allows us to jointly prune and refine trees at the same time. In an extensive experimental evaluation, we show that our approach not only offers statistically significantly better performance than the state-of-the-art but also offers a better accuracy-memory trade-off. We conclude our experimental evaluation with a case study showing the effectiveness of our method in a real-world setting.

Keywords Ensemble · Ensemble pruning · Random Forest · Memory management

Responsible editor: Albrecht Zimmermann

✉ Sebastian Buschjäger
sebastian.buschjaeger@tu-dortmund.de

Katharina Morik
katharina.morik@tu-dortmund.de

[1] Chair for Artificial Intelligence, TU Dortmund University, Otto-Hahn-Straße 12, 44221 Dortmund, NRW, Germany

1 Introduction

Ensemble algorithms offer state-of-the-art performance in many applications and often outperform single classifiers by a large margin. With the ongoing integration of embedded systems and machine learning models into our everyday life, e.g., in the form of the Internet of Things, the hardware platforms which execute ensembles must also be taken into account when training ensembles.

From a hardware perspective, a small ensemble with minimal execution time and a small memory footprint is desired. Moreover, learning theory indicates that ensembles of small models should generalize better so that they are ideal candidates for small, resource constraint devices (Koltchinskii 2002; Cortes et al. 2014). Practical problems, on the other hand, often require ensembles of complex base learners to achieve good results, and some ensemble techniques such as Random Forest (RF) even prefer individual trees to be as large as possible, leading to overall large ensembles (Breiman 2000; Biau 2012; Denil et al. 2014; Biau and Scornet 2016). As depicted in Table 1, microcontroller units (MCU) for IoT devices typically only offer a few KB to a few MB of memory, while RF models can easily grow beyond that limit (see Tables 4 and 5). Hence, to deploy RF onto these small devices, we require an algorithm that trains good models for a variety of different memory constraints.

Ensemble pruning is a standard technique for transforming a large, memory-hungry ensemble into a smaller ensemble that can be deployed onto a small device by removing classifiers from it (Tsoumakas et al. 2009; Zhang et al. 2006). Remarkably, this removal can sometimes lead to a *better* predictive performance (Margineantu and Dietterich 1997; Martínez-Muñoz and Suárez 2006; Li et al. 2012) leading to smaller *and* better ensembles at the same time. Similar, leaf-refinement (LR) is a technique that jointly refines the probability estimates in a given tree ensemble (Ren et al. 2015; Buschjäger and Morik 2021) to improve its performance. Hence, with leaf-refinement, it is possible to refine smaller forests, i.e., a Random Forest with only a few trees, so that it achieves a performance comparable to larger forests, i.e., an RF with many trees (c.f. Ren et al. 2015 and Table 4). We wonder whether we can combine both approaches into a single algorithm that jointly removes unnecessary classifiers from a tree ensemble while further improving its performance by refining the probability estimates in the leaf nodes. To do so, we incorporate L_1 regularization into the leaf-refinement objective and adopt proximal gradient descent to solve this objective. Our contributions are as follows:

- *Unified objective for leaf-refinement and pruning:* We present a novel optimization objective that leverages L_1 regularization to select only a few trees from the ensemble while jointly refining the probability estimates in all trees.
- *Pruning via proximal gradient descent (L1 + LR):* We present a new algorithm that uses proximal gradient descent to refine the probability estimates of the tree ensemble while pruning it by minimizing our novel objective.
- *Experiments:* We study the performance of our algorithm on 20 datasets and compare it against 8 state-of-the-art methods. We conduct over 13,200 experiments with a variety of different hyperparameter configurations and show that our L1 + LR method has the statistically significant best performance in terms of accuracy

Table 1 Typical microcontroller units (MCUs) found in edge and IoT devices

MCU	Arch.	Clock	Float	Word	SIMD	Flash	(S)RAM	Power
Arduino Uno	ATMega128P	16MHz	✗	8bit	✗	32 KB	2 KB	12 mA
Arduino Mega	ATMega2560	16MHz	✗	8bit	✗	256 KB	8 KB	6 mA
Arduino Nano	ATMega2560	16MHz	✗	8bit	✗	26–32 KB	1–2 KB	6 mA
STM32L0	ARM Cortex-M0	32MHz	✗	32bit	✗	192 KB	20 KB	7 mA
Arduino MKR1000	ARM Cortex-M0	48MHz	✗	32bit	✗	256 KB	32 KB	4 mA
Arduino Due	ARM Cortex-M3	84MHz	✗	32bit	✗	512 KB	96 KB	50 mA
STM32F2	ARM Cortex-M3	120MHz	✗	32bit	✗	1 MB	128 KB	21mA
STM32F4	ARM Cortex-M4	180MHz	✚	32bit	✚	2 MB	384 KB	50 mA
Raspberry PI A+	ARMv6	700MHz	✓	32bit	✗	SD Card	256 MB	80 mA
Raspberry PI Zero	ARMv6	1GHz	✓	32bit	✗	SD Card	512 MB	80 mA
Raspberry PI 3B	ARMv8	4@1.2GHz	✓	64bit	✓	SD Card	1 GB	260 mA
Intel i7-7700K	x86	4@4.5GHz	✓	64bit	✓	HDD/SSD	2–64 GB	≈ 80 A

The top group shows bare-metal MCUs, which typically do not run an operating system. The bottom group shows MCUs that typically also run an operating system. ✓ denotes the availability of a feature, ✗ marks its absence and ✚ denotes optional/partial support. The original table is due to Branco et al. (2019), but the float, SIMD, word, and architecture columns have been added by taking the corresponding values from the referenced data sheets. For comparison, the Intel i7-7700K CPU has also been added as a typical desktop/server CPU

and F_1 score. Moreover, we show that leaf-refinement, including our L1 + LR method, has the best performance when less than 768 KB of memory is available. Last, we present a real-world use case in the context of IoT warehousing and highlight how our method can be applied to deploy tree ensembles to tiny, ultra-low-power IoT devices.

The paper is organized as follows. Section 2 presents our notation and related work, including detailed explanations on ensemble pruning and leaf-refinement. In Sect. 3 we present our combination of ensemble pruning and leaf-refinement into a single objective and present a novel algorithm that solves this objective. Section 4 contains the experimental analysis, including a real-world use case in IoT Warehousing. Section 5 concludes the paper.

2 Background and notation

We consider a supervised learning setting in which training and test points are drawn i.i.d. according to some distribution $\mathcal{D}$ over the input space $\mathcal{X} \subseteq \mathbb{R}^d$ of d-dimensional feature vectors and labels $\mathcal{Y} \subseteq \mathbb{R}^C$. For classification problems with $C \geq 2$ classes we encode each label as a one-hot vector $y = (0, \ldots, 0, 1, 0, \ldots, 0)$ which contains a '1' at coordinate c for label $c \in \{0, \ldots, C-1\}$.

We assume that we have given an *already trained* additive tree ensemble $\{h_1, \ldots, h_M\}$ of M axis-aligned decision trees (DT) with the following decision function:

$$f(x) = \frac{1}{M} \sum_{i=1}^{M} h_i(x) \tag{1}$$

A DT partitions the input space $\mathcal{X}$ into d-dimensional hypercubes called leaves and uses independent predictions for each leaf in the tree. To do so, it uses a series of axis-aligned splits of the form $\mathbb{1}\{x_i \leq t\}$ and $\mathbb{1}\{x_i > t\}$ where i is a pre-computed feature index and t is a precomputed threshold to determine the leaf nodes. Each leaf node l contains a probability estimate $\widehat{y}_l \in \mathbb{R}^C$ using the class frequencies of the observations from the training points occurring in that leaf node. Let $s_l(x) \colon \mathcal{X} \to \{0, 1\}$ be an indicator function that is '1' if x belongs to leaf l and '0' if not, and let L be the total number of leaf nodes in the tree, then the prediction of a tree is given by

$$h(x) = \sum_{l=1}^{L} \widehat{y}_l s_l(x) \tag{2}$$

Note that per tree construction there is exactly one leaf node visited per example so that $s_l(x) = 1$ for exactly one $l \in \{1, \ldots, L\}$ whereas the remaining indicators evaluate for to zero. Hence, $h(x)$ effectively evaluates to $\widehat{y}_l$ where l is the corresponding leaf node of the input x.

In this paper, we assume that we have given an already trained forest of DTs, but we do not assume that any specific training algorithm was used to train it. For example, the forest can be a Random Forest (Breiman 2001), a forest of boosted decision trees (Schapire and Freund 2012), etc. For simplicity, we assume that each tree in the ensemble is equally weighted. If the forest is weighted (e.g. as in AdaBoost) so that each classifier h'_i has a corresponding weight w_i, then we re-scale the individual classifier's predictions to include the weight. To do so, we scale the probability estimates in all leaves of each tree h'_i by $M \cdot w_i$, so that

$$f(x) = \sum_{i=1}^{M} w_i h'_i(x) = \frac{1}{M} \sum_{i=1}^{M} M w_i h'_i(x) = \frac{1}{M} \sum_{i=1}^{M} h_i(x) \tag{3}$$

In addition to the trained ensemble, we receive a labeled pruning sample $\mathcal{S} = \{(x_i, y_i) \mid i = 1, \ldots, N\}$. This sample can either be the original training data used to train f or another pruning data set not related to the training or test data. In this paper, we will focus on classification problems, but note that our approach is also directly applicable to regression tasks. Moreover, we will focus on Random Forests (RF), but note that most of our discussion directly translates to other tree ensembles such as Bagging (Breiman 1996), ExtraTrees (Geurts et al. 2006), Random Subspaces (Ho 1998) or Random Patches (Louppe and Geurts 2012).

2.1 Ensemble pruning

The goal of ensemble pruning is to select a subset of K classifier from $\{h_1, \ldots, h_M\}$ that forms a small and accurate subensemble. Formally, each classifier h_i receives a

corresponding pruning weight $w_i \in \{0, 1\}$ so that the ensemble's prediction can be expressed as

$$f(x) = \frac{1}{\|w\|_0} \sum_{i=1}^{M} w_i h_i(x) \tag{4}$$

where $\|w\|_0 = \sum_{i=1}^{M} 1\{w_i > 0\}$ is the L_0 norm that counts the number of nonzero entries in the weight vector $w = (w_1, \ldots, w_M)$. Many effective ensemble pruning methods have been proposed in the literature. These methods usually differ in the specific loss function used to measure the performance of a subensemble and the way this loss is minimized. Tsoumakas et al. (2009) give a detailed taxonomy of pruning methods that was later expanded in Zhou (2012).

Ranking-based pruning: Early works on ensemble pruning focus on ranking-based approaches that assign a rank to each classifier depending on their individual performance and then select the top K classifier from that ranking. Formally, ranking-based approaches use the following optimization problem:

$$\arg\min_{w \in \{0,1\}^M} \frac{1}{N} \sum_{(x,y) \in \mathcal{S}} \sum_{i=1}^{M} w_i \ell(h_i(x), y) \text{ st. } \|w\|_0 = K \tag{5}$$

where $\ell \colon \mathbb{R}^C \times \mathcal{Y} \to \mathbb{R}$ is a loss function. To solve this objective the following approach can be used: First, all individual losses $\ell(h_i(x), y)$ are computed and sorted in decreasing order. Then, the K models with the smallest losses are selected, and their corresponding weights are set to 1. The remaining weights are set to 0. This makes ranking-based pruning methods appealing since they are very fast, easy to implement, and the optimum is easily obtained. One of the earliest ranking-based pruning methods was due to Margineantu and Dietterich who employ the Cohen-Kappa statistic to rate the effectiveness of each classifier (Margineantu and Dietterich 1997). Later, Martinez-Munoz and Suarez propose the use of the cosine similarity to measure how close the ensemble prediction is to the subensemble (Martínez-Muñoz and Suárez 2006). More recent approaches also incorporate the ensemble's diversity into the selection. Lu et al. propose to measure the individual contribution of each classifier to form a diverse and effective subensemble (Lu et al. 2010) and Guo et al. propose to directly maximize the classification margin, as well as the diversity of the subensemble (Guo et al. 2018).

Mixed Quadratic Integer Programming (MQIP): MQIP-based pruning methods enhance ranking-based methods by also adding a pairwise loss function that measures the relationship between two classifiers h_i and h_j. Formally, they use the following objective:

$$\arg\min_{w \in \{0,1\}^M} \frac{1}{N} \sum_{(x,y) \in \mathcal{S}} \left(\alpha \sum_{i=1}^{M} w_i \ell_1(h_i(x), y) + (1 - \alpha) \cdot \sum_{i=1}^{M} \sum_{j=1}^{M} w_i w_j \ell_2(h_i(x), h_j(x), y) \right) \text{ st. } \|w\|_0 = K \tag{6}$$

where $\alpha \in [0, 1]$ models the trade-off between the two losses $\ell_1 \colon \mathbb{R}^C \times \mathcal{Y} \to \mathbb{R}$ and $\ell_2 \colon \mathbb{R}^C \times \mathbb{R}^C \times \mathcal{Y} \to \mathbb{R}$. Here, ℓ_1 is again a loss function that relates the predictions of each classifier to the true label and ℓ_2 is a loss that relates the predictions of two classifiers $h_i(x)$ and $h_j(x)$ to each other and potentially also to the true label y.

Note that MQIP encapsulates ranking-based methods and recovers them for $\alpha = 1$. However, also note that solving MQIP problems can be difficult and often takes much more time compared to e.g. ranking-based approaches. Originally this approach was proposed by Zhang et al. (2006) which uses the pairwise errors of each classifier and $\alpha = 0$ (ℓ_1 is not used). Cavalcanti et al. (2016) expand this idea and combine 5 different measures into ℓ_1 and ℓ_2 including the diversity, correlation, kappa-statistic, disagreement, and double fault measure.

Clustering-based pruning: Another approach to pruning is to first cluster the different models into groups and then select one representative from each group. To do so, let

$$H_i = (h_i(x_1)_1, \ldots, h_i(x_1)_C, h_i(x_2)_1, \ldots, h_i(x_2)_C, \ldots, h_i(x_N)_1, \ldots, h_i(x_N)_C)$$

denote the (stacked) vector of all predictions of classifier h_i on the sample $\mathcal{S}$ with $N \cdot C$ entries. Further, let

$$c(i) = \underset{j=1,\ldots,K}{\arg\min} \left\{ d(\mu_j, H_i) \right\}$$

be the index of the closest cluster center $\{\mu_1, \ldots, \mu_K\} \subseteq \mathbb{R}^{NC}$ to H_i given a distance function $d \colon \mathbb{R}^{NC} \times \mathbb{R}^{NC} \to \mathbb{R}_+$. Then, clustering-based pruning formally solves the following optimization problem:

$$\begin{aligned} \underset{\substack{w \in \{0,1\}^M \\ \mu_1, \ldots, \mu_K \in \mathbb{R}^{NC}}}{\arg\min} \quad & \frac{1}{N} \sum_{(x,y) \in \mathcal{S}} \ell \left(\frac{1}{\|w\|_0} \sum_{i=1}^{M} w_i h_i(x), y \right) + \sum_{i=1}^{M} d(\mu_{c(i)} - H_i) \\ \text{st. } & \forall w_i = 1, w_j = 1, i \neq j : c(i) \neq c(j) \text{ and } \|w\|_0 = K \end{aligned} \tag{7}$$

Equation 7 has three parts: The first part $\frac{1}{N} \sum_{(x,y) \in \mathcal{S}} \ell \left(\sum_{i=1}^{M} w_i h_i(x), y \right)$ measures the error of the selected subensemble whereas $\sum_{i=1}^{M} d(\mu_{c(i)} - H_i)$ computes the appropriate cluster centers. Finally, the constraints combine both parts to select one representative from each cluster. This optimization problem can be solved with existing clustering algorithms in two steps: First, a clustering is obtained (e.g. by using K-Means (Lazarevic and Obradovic 2001) or Hierarchical Agglomerative Clustering (Giacinto et al. 2000)) and then representatives are selected from each cluster based on the loss ℓ. For example, Giacinto et al. (2000) propose to use hierarchical agglomerative clustering using the pairwise error probability as distance. Then, once the clusters have been obtained they select the most distant representatives from each cluster to form a diverse ensemble. Lazarevic and Obradovic (2001) propose to use K-means clustering with the Euclidean distance. In contrast to Giacinto et al., they iteratively remove the least accurate classifier from a cluster until only one classifier is left which is then included in the subensemble. More recent works on cluster-based pruning also directly include the diversity into the distance measure (Zyblewski and Woźniak 2019, 2020).

Algorithm 1 Ordering-based optimization.

Input Trained forest with M trees, constraint $K < M$, Loss ℓ
Output Weight vector $w \in \{0, 1\}^M$ so that $\|w\|_0 = K$

1: $w \leftarrow (0, \ldots, 0)$
2: $i \leftarrow \underset{i=1,\ldots,M}{\arg\min} \left\{ \frac{1}{N} \sum_{(x,y)\in\mathcal{S}} \ell\left(h_i(x), y\right) \right\}$
3: $w_i \leftarrow 1$
4: **for** $k = 1, \ldots, K-1$ **do**
5: $\quad i \leftarrow \underset{i=1,\ldots,M}{\arg\min} \left\{ \sum_{(x,y)\in\mathcal{S}} \ell\left(\frac{1}{\|w\|_0+1} \sum_{j=1}^{M} w_j h_j(x) + h_i(x) \right) | w_i \neq 1 \right\}$
6: $\quad w_i \leftarrow 1$
7: **end for**

Ordering-based pruning: Ordering-based pruning orders all ensemble members according to their individual performances as well as their overall contribution to the ensemble and then picks the top K classifier from this list. In this sense, ordering-based approaches are the most general method for ensemble pruning as they allow to directly minimize the ensemble error:

$$\underset{w\in\{0,1\}^M}{\arg\min} \frac{1}{N} \sum_{(x,y)\in\mathcal{S}} \ell\left(\frac{1}{\|w\|_0} \sum_{i=1}^{M} w_i h_i(x), y \right) \text{ st. } \|w\|_0 \leq K \tag{8}$$

where $\ell : \mathbb{R}^C \times \mathcal{Y} \to \mathbb{R}$ is again a loss function. To do so, ordering-based approaches sort individual classifiers according to their performance and greedily select the tree that minimizes the overall ensemble error the most. Algorithm 1 depicts the ordering-based optimization approach. First, the classifier with the best individual loss is selected in line 2. Then, line $4-6$ selects the classifier that minimizes ℓ the most given the already selected ensemble $\sum_{j=1}^{M} w_j h_j(x)$. In a sense, ordering-based approaches are greedy, because they select the model which improves the ensemble the most without considering all different combinations. Ordering-based pruning was also first presented by Margineantu and Dietterich (1997) which proposed to greedily minimize the overall ensemble error. A series of works by Martínez-Muñoz and Suárez (2004, 2006) and Martínez-Muñoz et al. (2008) add to this work by proposing different error measures. More recently, theoretical insights from Probably Approximately Correct Learning (PAC) theory and the bias-variance decomposition were also transformed into greedy pruning approaches (Li et al. 2012; Jiang et al. 2017).

2.2 Leaf-refinement

Looking beyond ensemble pruning itself there are numerous orthogonal methods to deploy ensembles to small devices. First, 'classic' decision tree pruning algorithms (e.g. minimal cost complexity pruning or sample complexity pruning, c.f. Barros et al. 2015) and more recent adaptations, such as cost-complexity forest pruning (Ravi and Serra 2017) already reduce the size of DTs while offering a better accuracy. Second, in the context of model compression (see e.g. Choudhary et al. 2020 for an overview)

specific models such as Bonsai (Kumar et al. 2017), Decision Jungles (Shotton et al. 2013) or X-CLEaVER (Lucchese et al. 2018) aim to find smaller tree ensembles already during training, sometimes involving pruning as a sub-procedure.

One particular interesting method called leaf-refinement refines the probability estimates in the leaf nodes of each tree by using a global loss that exploits complementary information between multiple trees (Ren et al. 2015; Buschjäger and Morik 2021). Since we can incorporate the ensemble weights into the leaf nodes as described above, leaf-refinement is a generalization of the re-weighting of ensembles (Akash et al. 2019; Shahhosseini et al. 2022; Shahhosseini and Hu 2020) making it a very general framework for improving tree ensembles. Formally, let $\theta_i = (\widehat{y}_{i,1}, \ldots, \widehat{y}_{i,L_i})$ be the probability estimates of all leaf nodes in tree h_i and let $h_{i,\theta_i}(x)$ denote the prediction of tree i using the probability estimates θ_i. Further, let $\theta = [\theta_1, \ldots, \theta_M]$ be the matrix of all probability estimates of all trees in the ensemble and let $f_\theta(x)$ denote the prediction of the ensemble with estimates θ. Then, leaf-refinement proposes to minimize a global loss function

$$\theta = \underset{\theta_1,\ldots,\theta_M}{\arg\min} \frac{1}{N} \sum_{(x_i,y_i)\in\mathcal{S}} \ell\left(\frac{1}{M}\sum_{j=1} h_{j,\theta_j}(x_i),\, y_i\right) \tag{9}$$

This global loss takes into account all the interactions between individual trees to refine the probability estimates in the leaves, but it does not change the structure of individual trees. Hence, it can be easily minimized by stochastic gradient descent (SGD). SGD is an iterative algorithm that takes a small step into the negative direction of the gradient in each iteration by using an estimate of the true gradient:

$$\theta \leftarrow \theta - \alpha g_{\mathcal{B}}(\theta) \tag{10}$$

where $g_{\mathcal{B}}(\theta)$ is the gradient of ℓ w.r.t. to θ computed on a mini-batch $\mathcal{B}$ and $\alpha \in \mathbb{R}_+$ is the step-size. The gradients for the individual entries θ_i are given by the chain rule:

$$g_{\mathcal{B}}(\theta_i) = \frac{1}{|\mathcal{B}|}\left(\sum_{(x,y)\in\mathcal{B}} \frac{\partial \ell(f_\theta(x), y)}{\partial f_\theta(x)} \frac{1}{M} s_{i,l}(x)\right)_{l=1,2,\ldots,L_i} \tag{11}$$

Algorithm 2 summarizes the Leaf-Refinement (LR) algorithm. First, the original probability estimates from the tree in the leaf nodes are used as an initialization for the parameter vectors θ_i in line 2. Then, SGD is performed for E epochs using Eqs. (10) and (11) in lines 4–10. Here, one epoch refers to one linear scan over the entire dataset in which $\left\lceil \frac{N}{|\mathcal{B}|} \right\rceil$ batches are processed so that each example occurs exactly in one batch during each epoch.

The specific choice for the loss functions differs in the literature. Ren et al. (2015) propose to use the hinge-loss in combination with a L_2 regularization term similar to

Algorithm 2 Leaf-Refinement (LR)

Input Trained forest with trees $h_1, \ldots, h_M$, Loss ℓ, Step size α
Number of epochs E, Batch size $|\mathcal{B}|$
Output Trees $h_1, \ldots, h_M$ with refined leaf-values

for $i = 1, \ldots, M$ **do** ▷ Init. leaf predictions
  $\theta_i \leftarrow (\widehat{y}_{i,1}, \widehat{y}_{i,2}, \ldots)$
end for
for epoch $1, \ldots, E$ **do** ▷ Perform SGD for E epochs
  for next batch $\mathcal{B}$ in epoch **do**
    for $i = 1, \ldots, M$ **do**
      $\theta_i \leftarrow \theta_i - \alpha g_{\mathcal{B}}(\theta_i)$ ▷ Perform update using Eq. (11)
    end for
  end for
end for

the SVM. Let $\lambda \in \mathbb{R}_+$ be a regularization strength, then they propose to minimize

$$\ell_\lambda(f_\theta(x), y) = \lambda \cdot \max(0, 1 - f_\theta(x) \cdot y) + \frac{1}{2}\|\theta\|_2^2 \tag{12}$$

where $\|\cdot\|_2^2$ is the L_2 norm introduced to combat overfitting.

Buschjäger and Morik (2021) adapt the negative correlation learning algorithm (NCL) from the context of neural network training for leaf-refinement to enforce different levels of diversity. NCL is based on the bias-(co-)variance decomposition that is then transformed into a regularized learning objective (c.f. Brown et al. 2005). Again, let $\lambda \in \mathbb{R}_+$ be the regularization strength then they propose to minimize

$$\ell_\lambda(f_\theta(x), y) = \frac{1}{M}\sum_{i=1}^{M}(h_{i,\theta_i}(x) - y)^2 - \frac{\lambda}{2M}\sum_{i=1}^{M} d_i{}^T D d_i \tag{13}$$

where $d_i = (h_{i,\theta_i}(x) - f(x))$, $D = 2 \cdot I_C$ is the $C \times C$ identity matrix with 2 on the main diagonal and $\lambda \in \mathbb{R}_+$ is the regularization strength. For $\lambda = 0$ this trains M classifier independently and no further diversity among the ensemble members is enforced, for $\lambda > 0$ more diversity is enforced during training and for $\lambda < 0$ diversity is discouraged.

3 Combining leaf-refinement and ensemble pruning

leaf-refinement as well as ensemble pruning enable better and smaller tree ensembles. However, both approaches tackle this challenge from different points of view. Ensemble pruning removes entire trees from the ensemble to reduce its memory consumption and, as a by-product, improves its predictive performance. leaf-refinement on the other hand, refines the probability estimates in the trees to improve the performance and, as a byproduct, enables the use of smaller forests with similar performance.

This leads to two questions: First, which of the two methods is better suited to deploy tree ensembles to small devices? Second, can we combine both methods to further

improve the predictive performance of the forest while having a smaller memory consumption at the same time? In this section, we present a method that combines leaf-refinement with ensemble pruning to compute a small and powerful ensemble at the same time.

Arguably, the simplest method to combine both approaches is to first prune the ensemble and then refine it afterward. However, this method does not consider the interactions between the pruning algorithm and leaf refinement. It is conceivable that pruning would select different trees if the probability estimates had been refined *before* the pruning process. Similarly, it is conceivable that refinement would compute different leaf values if it had been performed on the unpruned ensemble. We advocate that the selection of trees, as well as the refinement of the corresponding leaf values, should be performed *simultaneously* to find the overall smallest and best ensemble. The key challenge in this scenario is to incorporate the selection of trees into the gradient-based approach of leaf-refinement. In ensemble pruning each tree either receives weight 0 (not selected) or 1 (selected). Unfortunately, it is difficult to optimize over discrete values $\{0, 1\}^M$ with gradient-based approaches because we apply small, non-binary changes to the weights during optimization. One possible approach to solve this dilemma is to relax the constraints and optimize over real-valued weights $w \in \mathbb{R}^M$ in combination with an L_1 regularization penalty that enforces sparsity:

$$\theta, w = \underset{\theta, w \in \mathbb{R}^M}{\arg\min} \frac{1}{N} \sum_{(x_i, y_i) \in \mathcal{S}} \ell \left(\sum_{j=1}^{M} w_j h_{j,\theta_j}(x_i), y_i \right) + \lambda \|w\|_1 \tag{14}$$

Enforcing sparsity through a L_1 regularization has a long history in Data Mining and Machine Learning. Arguably the largest application of it can be found in feature selection via the LASSO and related methods (see e.g. Tibshirani 1996; Li et al. 2017), but also other application areas such as Matrix Factorization (Kumar and Sindhwani 2015), Neural Network Pruning (Li et al. 2016), Dictionary Learning (Jiang et al. 2015) have been explored.

Objective 14 is non-smooth due to the L_1 norm and hence cannot be minimized via SGD directly. Stochastic proximal gradient descent (SPGD) is an adaption of SGD that incorporates a projection operation into the updates so that it can cope with non-smooth objectives (Parikh and Boyd 2014). SPGD is an iterative algorithm, where every iteration consists of two steps: first, a gradient descent update of the objective function is performed without considering its non-smooth part (e.g. ignoring the L_1 regularizer). Then, a projection operator (sometimes called `prox`) is applied to project the updated parameters onto the correct solution considering the non-smooth part of the objective. Let w be the weight vector at step t and let $g_{\mathcal{B}}(w)_i$ be the gradient of the $i-th$ entry in w *without* considering the L_1 term. Furthermore, let $\mathcal{P}_\alpha$ be the `prox` operator of $\lambda \|w\|_1$ with step size α, then PSGD performs the following updates

$$w \leftarrow \mathcal{P}_\alpha \left(w - \alpha g_{\mathcal{B}}(w) \right) \tag{15}$$

using the gradient via the chain-rule

$$g_{\mathcal{B}}(w) = \frac{1}{|\mathcal{B}|}\left(\sum_{(x,y)\in\mathcal{B}} \frac{\partial \ell(f_{w,\theta}(x), y)}{\partial f_{w,\theta}(x)} h_{i,\theta_i}(x)\right)_{i=1,\dots,M} \tag{16}$$

and the prox $\mathcal{P}_\alpha : \mathbb{R}^M \to \mathbb{R}^M$ (Parikh and Boyd 2014):

$$\mathcal{P}_\alpha(w) = (\text{sign}(w_i)\max(|w| - \lambda\alpha, 0))_{i=1,\dots,M} \tag{17}$$

Since there is no regularizer for the leaf nodes we can directly minimize the objective wrt. θ without using the prox. In this case, the gradient for h_i now also contains its weights (again using the chain rule):

$$g_{\mathcal{B}}(\theta_i) = \frac{1}{|\mathcal{B}|}\left(\sum_{(x,y)\in\mathcal{B}} \frac{\partial \ell(f_{w,\theta}(x), y)}{\partial f_{w,\theta}(x)} w_i s_{i,l}(x)\right)_{l=1,2,\dots,L_i} \tag{18}$$

Algorithm 3 summarizes this approach. Similar to before the probability estimates in the leaf nodes are used as an initialization for the parameter vectors θ_i in line 2. Then, PSGD is performed for E epochs using Eqs. (16), (18), and (17). To do so, the gradient for each weight $g_{\mathcal{B}}(w)_i$ is computed, and a regular weight update is performed in line 7. Similarly, the gradient for the leaf nodes of each tree $g_{\mathcal{B}}(\theta_i)$ is computed in line 8 and a regular gradient descent update is performed. After the leaf nodes of each tree as well as its weights have been updated the prox operator is applied in line 10. For $\lambda > 0$ we call this algorithm leaf-refinement with L_1 regularization (L1 + LR). Setting $\lambda = 0$ and ignoring any weight updates (line 7) recovers the original leaf-refinement (LR) algorithm. Similarly, ignoring any updates for the leaf nodes in line 8 yields a new pruning algorithm that selects trees purely based on the L_1 norm which we call L_1 pruning.

Algorithm 3 Leaf-Refinement with L_1 regularization (L1 + LR)

Input Trained forest with trees $h_1, \dots, h_M$, Loss ℓ, Step size α
Number of epochs E, Batch size $|\mathcal{B}|$, regularization strength λ
Output Trees $h_1, \dots, h_M$ with refined leaf-values and weights $w_1, \dots, w_M$

1: **for** $i = 1, \dots, M$ **do** ▷ Init. leaf predictions
2: $\theta_i \leftarrow (\widehat{y}_{i,1}, \widehat{y}_{i,2}, \dots)$
3: **end for**
4: **for** epoch $1, \dots, E$ **do** ▷ Perform PSGD for E epochs
5: **for** next batch $\mathcal{B}$ in epoch **do**
6: **for** $i = 1, \dots, M$ **do**
7: $w_i \leftarrow w_i - \alpha^t g_{\mathcal{B}}(w)_i$ ▷ Perform update using Eq. (16)
8: $\theta_i \leftarrow \theta_i - \alpha g_{\mathcal{B}}(\theta_i)$ ▷ Perform update using Eq. (18)
9: **end for**
10: $w \leftarrow \mathcal{P}_\alpha(w)$ ▷ Apply the prox using Eq. (17)
11: **end for**
12: **end for**

4 Experiments

In this section, we experimentally evaluate the combination of leaf-refinement and pruning (L1 + LR) and compare its performance with vanilla Random Forests, pruned RFs, and vanilla Leaf refinement in the context of IoT. As argued before, our main concern is the final model size as it determines the resource consumption, runtime, and energy of the model application during deployment (Buschjäger and Morik 2017; Buschjäger et al. 2018). Hence, we adopt a hardware-agnostic view and ask the following three questions:

- **Question 1** What method has the best predictive performance?
- **Question 2** What method has the best predictive performance under memory constraints?
- **Question 3** How do these methods behave in a real-world use case?

An overview of all the hyperparameters for our experiments is given in Table 2. We use the following experimental protocol: The basic idea of ensemble pruning is to first overtrain the ensemble and then remove unnecessary classifiers from this overtrained pool. Oshiro et al. studied the impact of the number of trees on the performance of a regular RF and showed on a variety of datasets that there is no significant performance improvement when using more than 128 trees (Oshiro et al. 2012). Therefore, we 'overtrain' our base Random Forests with $M = 256$ trees to increase the classifier pool for pruning without increasing the training time significantly. To control the individual errors of trees, we set the maximum number of leaf nodes n_l to values between $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$. From greedy pruning methods, we use complementariness pruning (COMP) (Martínez-Muñoz and Suárez 2004), reduced error pruning (RE) (Margineantu and Dietterich 1997), and DREP (Li et al. 2012). COMP uses complementariness, i.e., the number of examples on which an estimator disagrees with the ensemble's prediction but is correct to rate each member of the ensemble. RE uses the error of the subensemble to rate each estimator, whereas DREP uses a PAC-style bound to rate each classifier. For cluster-based pruning we utilize largest mean distance (LMD) pruning (Giacinto et al. 2000) that first builds an agglomerative clustering of the estimators' accuracies and then selects those estimators that are the farthest away from each cluster into the new ensemble. For rank-based pruning, we employ individual error (IE) pruning (Lu et al. 2010), and individual contribution (IC) pruning (Jiang et al. 2017). IE uses the individual error of each estimator, whereas IC computes the individual contributions to the ensemble's prediction. Last, we also experimented with MIQP-based pruning (Zhang et al. 2006), but unfortunately, the MIQP optimizer would frequently fail or time-out during experiments[1] Each pruning method is tasked to select $K \in \{2, 4, 8, 16, 32, 64, 128\}$ trees from the base forest. For DREP we additionally vary $\rho \in \{0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$. During the development of our leaf-refinement method, we found that 50 epochs in combination with a batch size of 1024 minimizing the MSE loss works well on a variety of datasets. Hence, for leaf-refinement, we randomly select $K \in \{2, 4, 8, 16, 32, 64, 128\}$ trees from the random forests (which is similar to training a smaller forest directly) and

[1] We used Gurobi https://www.gurobi.com/. so that we did not include it in our evaluation.

Table 2 The methods and their corresponding hyperparameters

Method	Abbreviation	Hyperparameter
Random Forest (Breiman 2001)	RF	$n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$ $M \in \{2, 4, 8, 16, 32, 64, 128, 256\}$
Reduced error pruning (Margineantu and Dietterich 1997)	RE	$K \in \{2, 4, 8, 16, 32, 64, 128\}$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
Individual contribution pruning (Lu et al. 2010)	IC	$K \in \{2, 4, 8, 16, 32, 64, 128\}$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
Individual error pruning (Jiang et al. 2017)	IE	$K \in \{2, 4, 8, 16, 32, 64, 128\}$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
Complementariness pruning (Martínez-Muñoz and Suárez 2004)	comp	$K \in \{2, 4, 8, 16, 32, 64, 128\}$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
Largest mean distance pruning (Giacinto et al. 2000)	LMD	$K \in \{2, 4, 8, 16, 32, 64, 128\}$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
DREP (Li et al. 2012)	DREP	$\rho \in \{0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$, $K \in \{2, 4, 8, 16, 32, 64, 128, 256\}$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
Leaf refinement (Ren et al. 2015)	LR	$K \in \{2, 4, 8, 16, 32, 64, 128\}$, MSE loss Adam, $\alpha = 0.01$, $E = 50$, $\lvert\mathcal{B}\rvert = 128$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
L1	L1	$\lambda \in \{0.1, 0.2 \ldots, 0.9, 0.925, 0.955, 0.975, 1\}$, MSE loss Adam, $\alpha = 0.01$, $E = 50$, $\lvert\mathcal{B}\rvert = 128$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$
Leaf refinement + L1	LR + L1	$\lambda \in \{0.1, 0.2 \ldots, 0.9, 0.925, 0.955, 0.975, 1\}$, MSE loss Adam, $\alpha = 0.01$, $E = 50$, $\lvert\mathcal{B}\rvert = 128$ RF with $M = 256$, $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$

Each pruning method and leaf refinement receive an RF trained with $M = 256$ trees and $n_l \in \{16, 32, 64, 128, 256, 512, 1024\}$ leaf nodes. Each pruning method is tasked to select $K \in \{2, 4, 8, 16, 32, 64, 128\}$ trees from this forest. ρ denotes the hyperparameter of DREP as outlined in Li et al. (2012). λ is the regularization strength as outlined in Sect. 3. For leaf-refinement the MSE loss has been minimized for 50 epochs with a batch size of 1024 using Adam

minimize the MSE loss for 50 epochs with a batch size of 1024 using the Adam optimizer (Kingma and Ba 2015) implemented in PyTorch (Paszke et al. 2019). Recall that our L1 + LR method indirectly controls the number of trees in the forest through the regularization strength $\lambda \in \{1, 0.5, 0.1, 0.05, 0.01\}$. As discussed previously, we study two variations of our algorithm. In the first version, we do not perform any leaf-refinement, but only select trees using the L_1 norm and call this algorithm L1. In the second version, we combine leaf-refinement with the L_1 regularization as outlined in Algorithm 3 and call this algorithm L1 + LR. For our experiments, we use 20 publicly available classification datasets with 6435 to 78,095 examples as outlined in Table 3. Here, N denotes the total number of data points, d is the dimensionality, and C is the number of classes ranging from 2 to 11. The class distribution is also given for each dataset and each class. A dash '–' indicates that the corresponding dataset has fewer classes, e.g., adult has only two classes, and hence entries for C_2–C_{17} are marked with a dash. In all experiments, we perform a 5-fold cross-validation except when the dataset has a dedicated train/test split in which case we perform five repetitions of the experiment using different random seeds. We use the training set for both, training the initial forest and pruning it. For a fair comparison, we made sure that each method receives the same forest in each cross-validation run. In all experiments, we use minimal preprocessing and encode categorical features as one-hot encoding. The random forests have been trained with scikit-learn (Pedregosa et al. 2011). We implemented all pruning algorithms in a Python package for other researchers called PyPruning, which is available under https://github.com/sbuschjaeger/PyPruning. The code for the experiments in this paper is available under https://github.com/sbuschjaeger/leaf-refinement-experiments. In total, we evaluated 660 hyperparameter configurations per dataset, leading to a total of 13,200 experiments.

4.1 What method has the best predictive performance?

In the first experiment, we study the predictive performance of pruning and leaf-refinement without considering any memory constraints. To do so, we pick the hyperparameter configuration of each method that has the best predictive performance. To account for imbalanced datasets (e.g. ida2016) we study the predictive performance in terms of accuracy and F_1 score.

Table 4 shows the accuracy of each method on each dataset with the corresponding model size. For datasets without a dedicated train/test split we report the average accuracy and its standard deviation over the cross-validation folds. For datasets with a dedicated train/test split, we repeat the experiments with 5 different random seeds and report the average accuracy and its standard deviation over these repetitions. The highest accuracy is marked in bold. It can be clearly seen that the combination of leaf-refinement and L1 regularization (LR + L1) offers the best accuracy on 13 datasets (adult, avila, bank, chess, connect, eeg, elec, fashion, har, ida2016, mnist, mozilla, statlog) and is tied for the first place on 3 datasets (anuran, magic postures). LR is the best method on gas-drift and nursery, whereas RF ranks first on jm1. As was to be expected, Random Forest seems to underperform on most datasets and improvements are possible due to leaf refinement or pruning. However, it is also

Table 3 Datasets used for the experiments

dataset	N	d	C	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}
adult	32561	108	2	0.759	0.241	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
anuran	7195	22	10	0.093	0.483	0.075	0.043	0.066	0.156	0.038	0.016	0.009	0.021	-	-	-	-	-	-	-	-
avila	20867	10	12	0.411	0.000	0.010	0.034	0.105	0.188	0.043	0.050	0.080	0.004	0.050	0.026	-	-	-	-	-	-
bank	45211	51	2	0.883	0.117	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
chess	28056	23	18	0.100	0.051	0.102	0.077	0.017	0.007	0.162	0.061	0.003	0.024	0.021	0.014	0.071	0.149	0.003	0.128	0.009	0.001
connect	67557	42	3	0.095	0.246	0.658	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
eeg	14980	14	2	0.551	0.449	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
elec	45312	14	2	0.575	0.425	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
fashion	70000	784	10	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	-	-	-	-	-	-	-	-
gas-drift	13910	128	6	0.184	0.210	0.118	0.139	0.216	0.132	-	-	-	-	-	-	-	-	-	-	-	-
har	10299	561	6	0.167	0.150	0.137	0.173	0.185	0.189	-	-	-	-	-	-	-	-	-	-	-	-
ida2016	76000	170	2	0.983	0.017	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
japanese-vowels	9961	14	9	0.110	0.099	0.162	0.148	0.079	0.097	0.117	0.101	0.086	-	-	-	-	-	-	-	-	-
jm1	10885	16	2	0.807	0.193	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
magic	19019	10	2	0.648	0.352	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
mnist	70000	784	10	0.099	0.112	0.099	0.102	0.097	0.090	0.099	0.104	0.098	0.099	-	-	-	-	-	-	-	-
mozilla	15545	5	2	0.329	0.671	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
nursery	12960	27	5	0.333	0.331	0.311	0.025	-	-	-	-	-	-	-	-	-	-	-	-	-	-
postures	78095	9	5	0.208	0.192	0.209	0.189	0.201	-	-	-	-	-	-	-	-	-	-	-	-	-
statlog	6435	36	6	0.242	0.108	0.217	0.094	0.106	0.234	-	-	-	-	-	-	-	-	-	-	-	-

N denotes the total number of data points, d is the dimensionality and C is the number of classes ranging from 2 to 18. The class distribution is also given for each dataset and each class. A dash '–' indicates that the corresponding dataset has fewer classes, e.g. adult has only two classes and hence entries for C_2–C_{17} are marked with a dash

noteworthy that large improvements seem only to be possible with refinement and not with pruning. For example, RF only achieves 76.23% accuracy on the connect dataset and L1 + LR achieves up to 84.16% whereas the best pruning method (here L1) achieves 71.86% accuracy. Table 5 shows the F_1 score for each method in each dataset. Again, for datasets without a dedicated train/test split we report the average F_1 score and its standard deviation over the cross-validation folds. For datasets with a dedicated train/test split, we repeat the experiments with 5 different random seeds and report the average F_1 score and its standard deviation over these repetitions. The best method is marked in bold. Similar to before, L1 + LR ranks first on 14 datasets (adult, avila, bank, chess, connect, eeg, elec, fashion, har, ida2016, jm1, magic, mozilla, statlog) and is tied for first place on four datasets (anuran, mnist, nursery, postures) with LR. LR is the best method on two datasets (gas-drift, japanese-vowels). Interestingly, L1 + LR now also ranks first on the bank and jm1 datasets using the F_1 score which was not the case for the accuracy. We explain this behavior with the more imbalanced class distribution of these datasets. As expected, the model size greatly varies between data sets in both tables, but there is also a sizable difference between the individual methods. RF has arguably the largest models, followed by the various pruning methods, whereas LR, as well as L1 + LR, seem to have the smallest models, although it is difficult to give a general recommendation here. We will examine the model size in more detail in the next section.

To give a statistically meaningful comparison we present the results in Tables 4 and 5 as a CD diagram (Demšar 2006). In a CD diagram, each method is ranked according to its performance, and a Friedman-Test is used to determine if there is a statistical difference between the average rank of each method. If this is the case, then a pairwise Wilcoxon-Test between all methods checks whether there is a statistical difference between two classifiers. CD diagrams visualize this evaluation by plotting the average rank of each method on the x-axis and connecting all classifiers whose performances are statistically similar via a horizontal bar. Figure 1 shows the corresponding CD diagram for the accuracy (left side) and F_1 score (right side), where $p = 0.95$ was used for all statistical tests. In both cases, we see that L1 + LR ranks first, followed by LR. L1 + LR and LR are the statistically significant best methods. With some distance, L1, COMP, IC, RE, IE, RF, DREP, and LMD follow. Random Forest, LMD, and DREP are generally ranked last, whereas IE, RE, COMP, and L1 form one (for the accuracy) and two (in the case of the F_1 score) cliques ranked in the middle. We conclude that pruning and leaf-refinement improve the accuracy over the base Random Forest in almost all cases, confirming the results in the literature. However, leaf-refinement seems to perform better than pruning, and larger improvements in terms of accuracy and F_1 are possible when leaf values are refined. Last, the joint selection and refinement of trees via the L1 + LR algorithm seem to generally perform best ranking first in both cases, thereby supporting our initial hypothesis that both pruning and refinement should be integrated into each other for the best performance.

Table 4 The accuracy and model size of each method on each dataset

method	COMP	DREP	IC	IE	L1	L1+LR	LMD	LR	RE	RF
adult	86.48 ± 0.52 1.56 MB	86.38 ± 0.41 12.49 MB	86.45 ± 0.39 3.12 MB	86.48 ± 0.34 6.25 MB	86.53 ± 0.39 10.49 MB	**86.86 ± 0.26** 0.65 MB	86.38 ± 0.41 12.49 MB	85.96 ± 0.25 0.05 MB	86.45 ± 0.37 6.25 MB	86.41 ± 0.39 6.25 MB
anuran	98.23 ± 0.32 1.56 MB	98.12 ± 0.41 3.09 MB	98.17 ± 0.36 3.1 MB	98.14 ± 0.43 3.04 MB	98.15 ± 0.42 6.17 MB	**98.67 ± 0.28** 6.12 MB	98.12 ± 0.32 1.58 MB	**98.67 ± 0.21** 3.55 MB	98.17 ± 0.42 0.78 MB	98.12 ± 0.33 1.54 MB
avila	99.31 ± 0.03 7.13 MB	98.38 ± 0.07 30.8 MB	99.41 ± 0.06 3.2 MB	99.43 ± 0.05 3.18 MB	99.15 ± 0.06 24.25 MB	**99.85 ± 0.04** 4.08 MB	98.38 ± 0.07 30.8 MB	99.73 ± 0.06 4.05 MB	99.27 ± 0.09 3.46 MB	98.38 ± 0.07 30.82 MB
bank	90.63 ± 0.22 6.25 MB	90.5 ± 0.2 12.49 MB	90.62 ± 0.16 3.12 MB	90.66 ± 0.21 3.12 MB	90.72 ± 0.23 9.9 MB	**90.96 ± 0.35** 0.72 MB	90.53 ± 0.16 6.25 MB	90.34 ± 0.25 12.49 MB	90.59 ± 0.18 3.12 MB	90.52 ± 0.15 6.25 MB
chess	66.96 ± 0.41 5.56 MB	65.45 ± 0.22 44.48 MB	67.3 ± 0.31 5.56 MB	65.64 ± 0.22 11.12 MB	67.91 ± 0.45 43.85 MB	**83.19 ± 0.37** 37.74 MB	65.91 ± 0.29 11.12 MB	83.05 ± 0.46 22.23 MB	66.29 ± 0.48 11.12 MB	65.45 ± 0.22 44.48 MB
connect	77.35 ± 0.41 0.91 MB	76.32 ± 0.22 3.62 MB	77.34 ± 0.31 0.91 MB	77.29 ± 0.25 0.91 MB	77.86 ± 0.26 14.21 MB	**84.16 ± 0.16** 3.54 MB	76.32 ± 0.21 7.25 MB	83.27 ± 0.19 1.81 MB	77.1 ± 0.47 0.91 MB	76.23 ± 0.19 14.49 MB
eeg	92.95 ± 0.39 6.25 MB	92.84 ± 0.27 6.25 MB	92.92 ± 0.39 6.25 MB	92.92 ± 0.39 6.25 MB	92.98 ± 0.31 11.84 MB	**95.34 ± 0.24** 6.17 MB	92.82 ± 0.37 12.49 MB	95.32 ± 0.38 6.24 MB	92.89 ± 0.31 6.25 MB	92.82 ± 0.37 12.49 MB
elec	88.0 ± 0.33 3.12 MB	87.43 ± 0.3 12.49 MB	88.04 ± 0.37 3.12 MB	87.94 ± 0.24 0.78 MB	88.53 ± 0.29 12.12 MB	**92.64 ± 0.22** 12.49 MB	87.43 ± 0.3 12.49 MB	92.63 ± 0.13 12.49 MB	87.75 ± 0.28 1.56 MB	87.43 ± 0.3 12.49 MB
fashion	86.74 ± 0.1 28.49 MB	86.74 ± 0.1 28.49 MB	86.74 ± 0.1 28.49 MB	86.74 ± 0.1 28.49 MB	86.74 ± 0.1 28.49 MB	**89.09 ± 0.15** 12.06 MB	86.74 ± 0.1 28.49 MB	88.98 ± 0.13 28.49 MB	86.74 ± 0.1 28.49 MB	86.74 ± 0.1 28.49 MB
gas-drift	99.45 ± 0.08 5.58 MB	99.45 ± 0.08 5.58 MB	99.45 ± 0.1 1.27 MB	99.45 ± 0.07 2.74 MB	99.45 ± 0.07 5.58 MB	99.5 ± 0.08 0.88 MB	99.45 ± 0.08 5.58 MB	**99.52 ± 0.1** 1.27 MB	99.45 ± 0.08 5.58 MB	99.45 ± 0.08 5.58 MB
har	98.1 ± 0.39 1.64 MB	98.01 ± 0.45 6.45 MB	98.03 ± 0.49 1.28 MB	98.01 ± 0.45 6.45 MB	98.06 ± 0.49 6.45 MB	**98.94 ± 0.41** 1.17 MB	98.01 ± 0.45 6.45 MB	98.92 ± 0.32 1.27 MB	98.09 ± 0.51 1.28 MB	98.01 ± 0.49 3.23 MB
ida2016	99.25 ± 0.02 2.1 MB	99.24 ± 0.02 4.14 MB	99.24 ± 0.04 0.53 MB	99.24 ± 0.02 4.14 MB	99.25 ± 0.01 3.95 MB	**99.32 ± 0.02** 4.13 MB	99.24 ± 0.02 4.14 MB	99.31 ± 0.02 4.14 MB	99.25 ± 0.03 0.39 MB	99.24 ± 0.02 1.04 MB
japanese-vowels	97.68 ± 0.34 9.73 MB	97.62 ± 0.28 19.39 MB	97.72 ± 0.37 9.68 MB	97.62 ± 0.28 19.39 MB	97.74 ± 0.29 19.39 MB	98.6 ± 0.34 6.61 MB	97.65 ± 0.24 9.86 MB	**98.65 ± 0.25** 6.61 MB	97.72 ± 0.28 9.75 MB	97.62 ± 0.28 19.39 MB
jm1	81.93 ± 0.39 1.56 MB	81.92 ± 0.51 6.24 MB	81.95 ± 0.52 1.56 MB	81.92 ± 0.51 6.24 MB	81.98 ± 0.53 6.24 MB	81.83 ± 0.53 0.81 MB	81.92 ± 0.51 6.24 MB	81.18 ± 0.35 12.49 MB	81.92 ± 0.51 6.24 MB	**82.01 ± 0.68** 3.12 MB
magic	88.11 ± 0.38 6.25 MB	88.06 ± 0.31 3.12 MB	88.07 ± 0.45 6.25 MB	88.07 ± 0.33 6.25 MB	88.09 ± 0.39 12.36 MB	**88.42 ± 0.39** 10.24 MB	88.04 ± 0.53 6.25 MB	**88.42 ± 0.38** 12.49 MB	88.04 ± 0.49 12.49 MB	88.04 ± 0.49 12.49 MB
mnist	96.01 ± 0.05 14.24 MB	95.99 ± 0.07 28.49 MB	96.03 ± 0.07 14.24 MB	95.99 ± 0.07 28.49 MB	96.02 ± 0.08 27.97 MB	**98.11 ± 0.05** 27.97 MB	95.99 ± 0.07 28.49 MB	98.1 ± 0.06 28.49 MB	95.99 ± 0.07 28.49 MB	95.99 ± 0.07 28.49 MB
mozilla	95.36 ± 0.44 0.78 MB	95.37 ± 0.39 0.96 MB	95.36 ± 0.41 0.39 MB	95.42 ± 0.37 0.39 MB	95.35 ± 0.36 5.79 MB	**95.6 ± 0.26** 2.31 MB	95.3 ± 0.38 7.78 MB	95.57 ± 0.25 1.56 MB	95.33 ± 0.36 0.98 MB	95.31 ± 0.35 3.9 MB
nursery	99.26 ± 0.17 7.51 MB	99.17 ± 0.24 7.48 MB	99.29 ± 0.22 7.19 MB	99.24 ± 0.25 3.37 MB	99.19 ± 0.23 14.88 MB	99.8 ± 0.07 3.01 MB	99.13 ± 0.2 15.05 MB	**99.81 ± 0.05** 4.12 MB	99.2 ± 0.18 3.74 MB	99.13 ± 0.2 15.05 MB
postures	95.32 ± 0.17 4.62 MB	95.01 ± 0.17 18.49 MB	95.34 ± 0.17 4.62 MB	95.13 ± 0.17 9.25 MB	95.49 ± 0.17 18.09 MB	**98.64 ± 0.11** 18.49 MB	95.01 ± 0.17 18.49 MB	**98.64 ± 0.11** 18.49 MB	95.14 ± 0.17 4.62 MB	95.01 ± 0.17 18.49 MB
statlog	91.15 ± 0.25 7.09 MB	91.15 ± 0.25 7.09 MB	91.15 ± 0.25 7.09 MB	91.15 ± 0.25 7.09 MB	91.16 ± 0.24 7.09 MB	**91.66 ± 0.21** 7.09 MB	91.15 ± 0.25 7.09 MB	91.63 ± 0.2 3.54 MB	91.2 ± 0.34 1.79 MB	91.15 ± 0.25 7.09 MB

For datasets without a dedicated train/test split the average accuracy and its standard deviation over the cross-validation folds are reported. For datasets with a dedicated train/test split the experiments are repeated with 5 different random seeds and the average accuracy and its standard deviation over these repetitions are reported. Each entry is rounded to the second digit after to decimal point. Each row represents one dataset and each column is one method. Larger is better. The best average accuracy is marked in bold

Table 5 The F_1 score and model size of each method on each dataset

dataset	COMP	DREP	IC	IE	L1	L1+LR	LMD	LR	RE	RF
adult	0.8002 ± 0.006 3.12 MB	0.7985 ± 0.006 12.49 MB	0.8002 ± 0.006 3.12 MB	0.7998 ± 0.005 6.25 MB	0.8004 ± 0.006 10.49 MB	**0.8087 ± 0.004** 0.65 MB	0.7985 ± 0.006 12.49 MB	0.7972 ± 0.004 0.19 MB	0.8005 ± 0.007 1.56 MB	0.7994 ± 0.006 6.25 MB
anuran	0.9546 ± 0.014 1.56 MB	0.9528 ± 0.015 6.17 MB	0.9543 ± 0.012 3.09 MB	0.9543 ± 0.015 3.04 MB	0.9537 ± 0.015 6.17 MB	0.968 ± 0.008 1.74 MB	0.9541 ± 0.011 1.58 MB	**0.9688 ± 0.007** 3.55 MB	0.9528 ± 0.015 6.17 MB	0.9529 ± 0.012 1.54 MB
avila	0.9925 ± 0.0 7.13 MB	0.9854 ± 0.001 30.8 MB	0.9933 ± 0.001 3.2 MB	0.9934 ± 0.001 6.74 MB	0.9913 ± 0.0 24.25 MB	**0.9974 ± 0.0** 3.1 MB	0.9854 ± 0.001 30.8 MB	0.9964 ± 0.001 4.05 MB	0.9924 ± 0.001 3.46 MB	0.9854 ± 0.001 30.82 MB
bank	0.7233 ± 0.006 0.78 MB	0.7023 ± 0.008 0.39 MB	0.7248 ± 0.006 0.78 MB	0.7216 ± 0.006 3.12 MB	0.7236 ± 0.005 11.89 MB	**0.7524 ± 0.007** 0.72 MB	0.6994 ± 0.007 6.25 MB	0.7414 ± 0.004 12.49 MB	0.7242 ± 0.009 0.78 MB	0.7031 ± 0.009 0.78 MB
chess	0.6411 ± 0.017 5.56 MB	0.6273 ± 0.014 44.48 MB	0.6461 ± 0.009 5.56 MB	0.6277 ± 0.023 5.56 MB	0.6527 ± 0.014 43.85 MB	**0.8373 ± 0.012** 37.74 MB	0.632 ± 0.021 11.12 MB	0.8348 ± 0.012 22.23 MB	0.6384 ± 0.024 5.56 MB	0.6273 ± 0.014 44.48 MB
connect	0.5145 ± 0.01 0.11 MB	0.5137 ± 0.006 0.11 MB	0.5233 ± 0.013 0.11 MB	0.5274 ± 0.009 0.11 MB	0.5161 ± 0.004 14.21 MB	**0.6838 ± 0.006** 10.52 MB	0.4885 ± 0.003 0.45 MB	0.674 ± 0.003 1.81 MB	0.5227 ± 0.012 0.23 MB	0.4899 ± 0.011 0.11 MB
eeg	0.9284 ± 0.004 6.25 MB	0.9272 ± 0.003 6.25 MB	0.9281 ± 0.004 6.25 MB	0.9281 ± 0.004 6.25 MB	0.9287 ± 0.003 11.84 MB	**0.9528 ± 0.002** 6.17 MB	0.9271 ± 0.004 12.49 MB	0.9526 ± 0.004 6.24 MB	0.9278 ± 0.003 6.25 MB	0.9271 ± 0.004 12.49 MB
elec	0.8764 ± 0.004 3.12 MB	0.8705 ± 0.003 12.49 MB	0.8768 ± 0.004 3.12 MB	0.876 ± 0.003 0.78 MB	0.8821 ± 0.003 12.12 MB	**0.9246 ± 0.002** 12.49 MB	0.8705 ± 0.003 12.49 MB	0.9245 ± 0.001 12.49 MB	0.8738 ± 0.003 1.56 MB	0.8705 ± 0.003 12.49 MB
fashion	0.8655 ± 0.001 28.49 MB	0.8655 ± 0.001 28.49 MB	0.8655 ± 0.001 28.49 MB	0.8655 ± 0.001 28.49 MB	0.8655 ± 0.001 28.49 MB	**0.8904 ± 0.001** 12.06 MB	0.8655 ± 0.001 28.49 MB	0.889 ± 0.001 28.49 MB	0.8655 ± 0.001 28.49 MB	0.8655 ± 0.001 28.49 MB
gas-drift	0.9943 ± 0.001 5.58 MB	0.9943 ± 0.001 5.58 MB	0.9944 ± 0.001 1.27 MB	0.9943 ± 0.001 2.74 MB	0.9944 ± 0.001 5.58 MB	0.995 ± 0.001 2.29 MB	0.9943 ± 0.001 5.58 MB	**0.9952 ± 0.001** 1.27 MB	0.9943 ± 0.001 5.58 MB	0.9943 ± 0.001 5.58 MB
har	0.9809 ± 0.004 1.64 MB	0.9799 ± 0.005 6.45 MB	0.9801 ± 0.005 1.28 MB	0.9799 ± 0.005 6.45 MB	0.9804 ± 0.005 6.45 MB	**0.9895 ± 0.004** 1.17 MB	0.9799 ± 0.005 6.45 MB	0.9894 ± 0.003 1.27 MB	0.9807 ± 0.005 1.28 MB	0.9799 ± 0.005 6.45 MB
ida2016	0.9082 ± 0.002 2.1 MB	0.9056 ± 0.002 4.14 MB	0.9068 ± 0.005 0.53 MB	0.9056 ± 0.002 4.14 MB	0.908 ± 0.002 3.95 MB	**0.9176 ± 0.002** 4.13 MB	0.906 ± 0.005 0.39 MB	0.9164 ± 0.003 2.07 MB	0.9077 ± 0.004 0.39 MB	0.9062 ± 0.003 1.04 MB
japanese-vowels	0.9755 ± 0.004 9.73 MB	0.9747 ± 0.003 19.39 MB	0.9759 ± 0.004 9.68 MB	0.9747 ± 0.003 19.39 MB	0.9763 ± 0.003 19.39 MB	0.9853 ± 0.004 6.61 MB	0.9751 ± 0.003 9.86 MB	**0.9858 ± 0.003** 6.61 MB	0.976 ± 0.003 9.75 MB	0.9747 ± 0.003 19.39 MB
jm1	0.6121 ± 0.005 3.12 MB	0.6174 ± 0.011 1.56 MB	0.6112 ± 0.008 1.56 MB	0.6112 ± 0.017 0.2 MB	0.6104 ± 0.008 12.37 MB	**0.6292 ± 0.006** 10.23 MB	0.6147 ± 0.014 3.12 MB	0.6283 ± 0.005 6.25 MB	0.6185 ± 0.013 0.78 MB	0.6116 ± 0.005 0.78 MB
magic	0.8658 ± 0.005 6.25 MB	0.8652 ± 0.004 3.12 MB	0.8651 ± 0.005 6.25 MB	0.8651 ± 0.004 6.25 MB	0.8654 ± 0.005 12.36 MB	**0.8699 ± 0.004** 10.24 MB	0.8648 ± 0.006 12.49 MB	0.8696 ± 0.004 12.49 MB	0.8648 ± 0.006 12.49 MB	0.8648 ± 0.006 12.49 MB
mnist	0.9598 ± 0.0 14.24 MB	0.9597 ± 0.001 28.49 MB	0.96 ± 0.001 14.24 MB	0.9597 ± 0.001 28.49 MB	0.9599 ± 0.001 27.97 MB	**0.9809 ± 0.001** 27.97 MB	0.9597 ± 0.001 28.49 MB	**0.9809 ± 0.001** 28.49 MB	0.9597 ± 0.001 28.49 MB	0.9597 ± 0.001 28.49 MB
mozilla	0.946 ± 0.005 0.78 MB	0.9461 ± 0.005 0.96 MB	0.9461 ± 0.005 0.39 MB	0.9467 ± 0.005 0.39 MB	0.9457 ± 0.004 5.79 MB	**0.949 ± 0.003** 2.31 MB	0.9452 ± 0.005 7.78 MB	0.9486 ± 0.003 1.56 MB	0.9457 ± 0.004 0.98 MB	0.9453 ± 0.004 3.9 MB
nursery	0.9869 ± 0.003 7.51 MB	0.9847 ± 0.004 7.48 MB	0.9894 ± 0.004 7.19 MB	0.9879 ± 0.004 3.37 MB	0.986 ± 0.005 14.88 MB	**0.9981 ± 0.001** 3.01 MB	0.9816 ± 0.005 15.05 MB	**0.9981 ± 0.001** 4.12 MB	0.9843 ± 0.003 7.52 MB	0.9828 ± 0.005 7.57 MB
postures	0.9527 ± 0.002 4.62 MB	0.9496 ± 0.002 18.49 MB	0.953 ± 0.002 4.62 MB	0.9508 ± 0.002 9.25 MB	0.9544 ± 0.002 18.09 MB	**0.9863 ± 0.001** 18.49 MB	0.9496 ± 0.002 18.49 MB	**0.9863 ± 0.001** 18.49 MB	0.9509 ± 0.002 4.62 MB	0.9496 ± 0.002 18.49 MB
statlog	0.8969 ± 0.002 3.58 MB	0.8969 ± 0.003 7.09 MB	0.8969 ± 0.003 7.09 MB	0.8969 ± 0.003 7.09 MB	0.897 ± 0.003 7.09 MB	**0.903 ± 0.003** 2.13 MB	0.8969 ± 0.003 7.09 MB	0.9026 ± 0.003 3.54 MB	0.8972 ± 0.004 1.79 MB	0.8969 ± 0.003 7.09 MB

For datasets without a dedicated train/test split the average F_1 score and its standard deviation over the cross-validation folds are reported. For datasets with a dedicated train/test split the experiments are repeated with 5 different random seeds and the average F_1 score and its standard deviation over these repetitions are reported. Each entry is rounded to the fourth digit after to decimal point. Each row represents one dataset and each column is one method. Larger is better. The best F_1 score is marked in bold

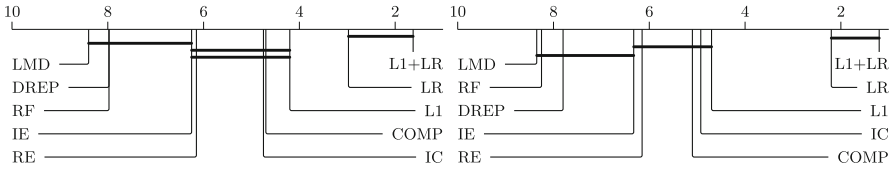

Fig. 1 CD-diagram for the accuracy (left side) and F_1 score (right side) of the different methods over multiple datasets. For all statistical tests, $p = 0.95$ was used. More to the right (lower rank) is better. The methods in connected cliques are statistically similar

4.2 What method has the best predictive performance under memory constraints?

In the second experiment, we study the predictive performance of pruning and leaf-refinement under memory constraints. Recall that small IoT devices are often severely limited in terms of memory (c.f. Table 1) and we can only deploy models that fit the available memory. For our analysis, we adopt a hardware-agnostic view which assumes that we are given a fixed memory budget for our model, which should, naturally, maintain a state-of-the-art performance. To do so, we pick the hyperparameter configuration of each method that has the best predictive performance while having a model size smaller than {256, 512, 768, 1024, 2048} KB. The size of the model is computed as follows: A baseline implementation of DTs stores each node in an array and iterates over it. Each node inside the array requires a pointer to the left / right child (8 bytes in total assuming `int` is used), a boolean flag if it is a leaf-node (1 byte), the feature index as well as the threshold to compare the feature against (8 bytes assuming `int` and `float` is used). Finally, entries for class probabilities are required for the leaf nodes (4 bytes per class, assuming that `float` is used). Thus, in total, a single node requires $17 + 4 \cdot C$ Bytes per node, which we sum over all nodes in the entire ensemble (Buschjäger et al. 2018).

We could not find meaningful differences between the F_1 score and the accuracy, and hence we will focus on the accuracy for now and revisit the F_1 score later on. Moreover, we will focus on {256, 768, 2048} KB constraints. Additional tables with additional memory constraints, as well as the F_1 score, are given in the appendix. Table 6 shows the accuracy for model sizes below 256 KB. Contrary to the accuracies without any memory constraints, this table is now more fragmented. L1 + LR is the best method on 4 datasets (adult, har, nursery, statlog), whereas vanilla LR ranks first on 10 datasets (anuran, chess, connect, eeg, elec fashion, gas-drift, japanese-vowels, mnist postures). RE pruning is the best option on two datasets (avila, mozilla), IC is the best option on the jm1 dataset, COMP is the best algorithm on the magic dataset, and IC and COMP are the best options on the jda2016 dataset. Somewhat surprisingly, pruning via L1 did not lead to valid models on any dataset, whereas L1 + LR produces valid models on 12 datasets. We suspect that L1 and L1 + LR require different values for λ to select a similar amount of trees. We investigate this phenomenon in more detail in the next section. Going from 256 KB constraints to 768 KB constraints in Table 7, L1 + LR seems to improve. It now ranks first on 7 datasets (adult, anuran, bank, eeg, har, magic, statlog), followed by LR, which ranks first on 10 datasets (chess,connect,elec, fashion,

Table 6 The accuracy of each method on each dataset with a model size below 256 KB

dataset	COMP	DREP	IC	IE	L1	L1+LR	LMD	LR	RE	RF
adult	86.1 ± 0.45	85.86 ± 0.39	85.97 ± 0.32	86.2 ± 0.49	-	**86.32 ± 0.29**	85.57 ± 0.42	85.96 ± 0.25	86.29 ± 0.38	85.98 ± 0.23
anuran	97.5 ± 0.38	97.43 ± 0.41	97.51 ± 0.53	97.57 ± 0.5	-	97.78 ± 0.33	97.28 ± 0.49	**98.0 ± 0.46**	97.51 ± 0.31	97.23 ± 0.42
avila	94.8 ± 0.48	92.94 ± 0.83	94.69 ± 0.16	94.76 ± 0.23	-	-	83.63 ± 1.22	92.43 ± 1.89	**94.85 ± 0.27**	87.99 ± 1.54
bank	90.37 ± 0.1	90.07 ± 0.15	90.37 ± 0.23	**90.48 ± 0.18**	-	90.47 ± 0.14	90.14 ± 0.23	90.33 ± 0.35	90.43 ± 0.12	90.08 ± 0.22
chess	51.61 ± 0.59	50.29 ± 0.59	50.97 ± 0.55	52.08 ± 0.68	-	-	49.59 ± 0.65	**56.95 ± 0.83**	51.75 ± 1.48	48.56 ± 0.77
connect	76.26 ± 0.25	75.83 ± 0.1	76.63 ± 0.35	76.74 ± 0.27	-	-	74.99 ± 0.33	**80.73 ± 0.1**	76.6 ± 0.49	74.83 ± 0.16
eeg	89.09 ± 0.57	88.33 ± 0.38	88.77 ± 0.3	88.71 ± 0.46	-	90.53 ± 0.55	87.88 ± 0.45	**90.55 ± 0.29**	88.64 ± 0.6	87.82 ± 0.71
elec	86.0 ± 0.28	85.66 ± 0.41	86.49 ± 0.51	86.71 ± 0.25	-	-	85.24 ± 0.5	**87.7 ± 0.36**	86.56 ± 0.26	85.43 ± 0.44
fashion	84.06 ± 0.14	83.81 ± 0.22	84.07 ± 0.12	84.12 ± 0.24	-	-	83.8 ± 0.17	**85.32 ± 0.12**	84.25 ± 0.16	83.66 ± 0.16
gas-drift	99.23 ± 0.18	99.25 ± 0.05	99.28 ± 0.08	99.31 ± 0.06	-	99.23 ± 0.05	99.24 ± 0.06	**99.35 ± 0.08**	99.23 ± 0.15	99.22 ± 0.09
har	97.37 ± 0.38	96.63 ± 0.5	97.16 ± 0.36	96.7 ± 0.26	-	**97.89 ± 0.34**	96.18 ± 0.39	97.72 ± 0.36	97.0 ± 0.49	96.3 ± 0.52
ida2016	**99.23 ± 0.02**	99.18 ± 0.03	**99.23 ± 0.02**	99.21 ± 0.03	-	99.12 ± 0.02	99.19 ± 0.05	99.09 ± 0.04	99.2 ± 0.03	99.17 ± 0.03
japanese-vowels	93.34 ± 0.53	92.8 ± 0.65	92.89 ± 0.49	92.73 ± 0.45	-	-	92.7 ± 0.57	**95.38 ± 0.43**	93.29 ± 0.34	92.85 ± 0.43
jm1	81.63 ± 0.5	81.65 ± 0.67	**81.77 ± 0.61**	81.53 ± 0.4	-	81.59 ± 0.5	81.71 ± 0.39	80.6 ± 0.83	81.53 ± 0.43	81.54 ± 0.55
magic	**87.42 ± 0.58**	87.1 ± 0.51	87.2 ± 0.57	87.22 ± 0.56	-	87.41 ± 0.41	87.23 ± 0.51	86.93 ± 0.47	87.13 ± 0.67	87.08 ± 0.59
mnist	92.11 ± 0.23	91.46 ± 0.13	92.14 ± 0.2	91.74 ± 0.28	-	-	91.16 ± 0.33	**94.24 ± 0.15**	91.94 ± 0.16	91.4 ± 0.46
mozilla	95.02 ± 0.37	95.01 ± 0.35	95.19 ± 0.37	95.15 ± 0.43	-	95.12 ± 0.33	94.83 ± 0.36	94.8 ± 0.41	**95.23 ± 0.3**	94.85 ± 0.31
nursery	98.04 ± 0.19	96.97 ± 0.46	97.92 ± 0.48	97.61 ± 0.33	-	**99.06 ± 0.25**	96.05 ± 0.32	98.72 ± 0.26	97.73 ± 0.35	96.58 ± 0.29
postures	89.63 ± 0.25	88.92 ± 0.32	89.36 ± 0.26	89.52 ± 0.16	-	-	88.14 ± 0.27	**90.27 ± 0.27**	89.45 ± 0.29	88.14 ± 0.23
statlog	89.36 ± 0.32	89.49 ± 0.37	89.59 ± 0.23	89.42 ± 0.41	-	**90.16 ± 0.26**	89.43 ± 0.17	89.78 ± 0.22	89.72 ± 0.21	89.22 ± 0.31

For datasets without a dedicated train/test split, the average accuracy and its standard deviation over the cross-validation folds are reported. For datasets with a dedicated train/test split, the experiments are repeated with 5 different random seeds, and the average accuracy and its standard deviation over these repetitions are reported. A dash '–' indicates that a method did not produce any model that fits the memory constraint. Each entry is rounded to the second digit after to decimal point. Each row represents one dataset, and each column is one method. Larger is better. The best method is marked in bold

Table 7 The accuracy of each method on each dataset with a model size below 768 KB

dataset	COMP	DREP	IC	IE	L1	L1+LR	LMD	LR	RE	RF
adult	86.36 ± 0.49	86.14 ± 0.26	86.23 ± 0.39	86.23 ± 0.46	86.07 ± 0.46	**86.86 ± 0.26**	85.87 ± 0.63	85.96 ± 0.25	86.31 ± 0.5	85.98 ± 0.23
anuran	97.93 ± 0.19	97.75 ± 0.45	97.89 ± 0.25	98.0 ± 0.28	-	**98.5 ± 0.32**	97.72 ± 0.44	98.28 ± 0.35	97.83 ± 0.36	97.82 ± 0.48
avila	98.77 ± 0.17	95.41 ± 0.69	99.03 ± 0.24	**99.11 ± 0.15**	-	98.98 ± 0.12	90.35 ± 0.55	98.78 ± 0.55	98.58 ± 0.1	94.93 ± 0.88
bank	90.44 ± 0.14	90.16 ± 0.09	90.48 ± 0.2	90.48 ± 0.18	90.25 ± 0.15	**90.96 ± 0.35**	90.29 ± 0.1	90.33 ± 0.35	90.45 ± 0.06	90.22 ± 0.18
chess	59.74 ± 0.75	58.66 ± 0.66	59.74 ± 0.81	60.13 ± 0.55	-	-	58.42 ± 0.39	**68.95 ± 0.76**	59.99 ± 0.25	57.14 ± 0.85
connect	77.0 ± 0.26	76.15 ± 0.16	77.13 ± 0.28	77.16 ± 0.21	73.44 ± 0.24	78.17 ± 0.24	75.87 ± 0.2	**81.82 ± 0.25**	76.95 ± 0.45	75.62 ± 0.27
eeg	90.23 ± 0.41	89.9 ± 0.1	90.15 ± 0.45	90.03 ± 0.42	83.56 ± 0.59	**93.47 ± 0.41**	89.23 ± 0.67	91.44 ± 0.53	89.98 ± 0.45	89.53 ± 0.75
elec	87.14 ± 0.37	86.78 ± 0.42	87.35 ± 0.32	87.6 ± 0.36	82.85 ± 0.17	88.22 ± 0.19	86.27 ± 0.46	**88.64 ± 0.21**	87.26 ± 0.36	86.4 ± 0.12
fashion	85.12 ± 0.07	84.8 ± 0.21	84.91 ± 0.18	84.91 ± 0.16	-	84.06 ± 0.14	84.62 ± 0.17	**86.32 ± 0.2**	85.1 ± 0.15	84.55 ± 0.23
gas-drift	99.39 ± 0.09	99.37 ± 0.11	99.42 ± 0.09	99.42 ± 0.12	-	99.48 ± 0.05	99.36 ± 0.11	**99.51 ± 0.08**	99.4 ± 0.1	99.38 ± 0.09
har	98.01 ± 0.36	97.58 ± 0.39	97.78 ± 0.31	97.56 ± 0.39	-	**98.78 ± 0.31**	97.42 ± 0.47	98.56 ± 0.29	97.96 ± 0.49	97.36 ± 0.46
ida2016	99.24 ± 0.03	99.2 ± 0.03	99.24 ± 0.04	99.23 ± 0.03	99.1 ± 0.02	99.26 ± 0.01	99.23 ± 0.04	**99.28 ± 0.04**	99.25 ± 0.03	99.21 ± 0.04
japanese-vowels	94.61 ± 0.61	94.62 ± 0.42	94.87 ± 0.34	94.37 ± 0.63	-	96.63 ± 0.47	94.43 ± 0.35	**96.7 ± 0.77**	94.84 ± 0.37	94.61 ± 0.46
jm1	81.66 ± 0.66	81.76 ± 0.45	**81.77 ± 0.61**	81.57 ± 0.23	-	81.74 ± 0.61	81.71 ± 0.39	80.6 ± 0.83	81.64 ± 0.32	81.75 ± 0.5
magic	87.42 ± 0.58	87.28 ± 0.43	87.41 ± 0.53	87.49 ± 0.55	86.72 ± 0.51	**88.15 ± 0.36**	87.49 ± 0.44	86.93 ± 0.47	87.3 ± 0.26	87.29 ± 0.42
mnist	93.38 ± 0.23	92.98 ± 0.09	93.48 ± 0.12	93.12 ± 0.31	-	-	92.71 ± 0.14	**95.62 ± 0.1**	93.39 ± 0.09	92.88 ± 0.09
mozilla	95.34 ± 0.41	95.26 ± 0.33	95.36 ± 0.41	**95.42 ± 0.37**	94.62 ± 0.42	**95.42 ± 0.38**	94.96 ± 0.35	94.92 ± 0.45	95.28 ± 0.29	95.2 ± 0.19
nursery	98.73 ± 0.18	98.12 ± 0.43	98.77 ± 0.26	98.56 ± 0.14	95.25 ± 0.26	99.57 ± 0.16	97.71 ± 0.23	**99.65 ± 0.12**	98.78 ± 0.17	97.96 ± 0.37
postures	94.33 ± 0.13	93.81 ± 0.35	94.19 ± 0.37	94.1 ± 0.16	78.36 ± 0.22	-	93.25 ± 0.23	**95.69 ± 0.29**	94.03 ± 0.28	93.35 ± 0.39
statlog	90.4 ± 0.13	90.32 ± 0.35	90.45 ± 0.33	90.43 ± 0.33	-	**91.16 ± 0.26**	90.12 ± 0.19	90.79 ± 0.27	90.2 ± 0.33	90.2 ± 0.48

For datasets without a dedicated train/test split, the average accuracy and its standard deviation over the cross-validation folds are reported. For datasets with a dedicated train/test split, the experiments are repeated with 5 different random seeds, and the average accuracy and its standard deviation over these repetitions are reported. A dash '–' indicates that a method did not produce any model that fits the memory constraint. Each entry is rounded to the second digit after to decimal point. Each row represents one dataset, and each column is one method. Larger is better. The best method is marked in bold

gas-drift, ida2016, japanese-vowels, mnist, nursery, postures) and IE that ranks first on one dataset (avila). L1 + LR and IE share the first place on the mozilla dataset, and IC ranks first on one dataset (jm1). This trend continues for larger memory sizes as depicted in Table 8. Here, L1 + LR now ranks first on 12 datasets with a performance close to that of the unconstrained ones in Table 4. LR ranks first on 7 datasets, and IC ranks first on one dataset.

We conclude that for small model sizes below 256 KB, pruning and refinement offer better predictive performance than a vanilla random forest, but it is difficult to give a clear recommendation of what method works best in this scenario. We hypothesize that due to the small model size, each method can only pick a few comparably small trees all with similar performance, and hence we find similar performances across the methods. Furthermore, LR seems to perform slightly better than L1 + LR. Once more memory is available, each method can pick more and larger trees, thereby leaving more room for picking 'good' and 'bad' trees. Hence, we see more differences between the individual methods and a clear trend toward refinement. Finally, for larger models with 2048 KB constraints, there is a clear trend towards L1 + LR for the best performance.

The difference between vanilla LR and L1 + LR for smaller model sizes can be explained by the choice of hyperparameters in this experiment. LR considers $K \in \{2, 4, 8, 16, 32, 64, 128\}$ trees for refinement, whereas L1 + LR indirectly chooses the number of trees via $\lambda \in \{0.1, 0.2, \ldots, 0.9, 0.925, 0.955, 0.975, 1.0\}$. We suspect that a more fine-grained selection for values of λ would have led to a more fine-grained distribution of different models with potentially better performance. Figure 2 shows the average number of estimators across all datasets and all configurations selected for different λ values in L1 + LR. The error band shows the standard deviation. As expected, increasing λ leads to a reduction in the number of trees. Between $\lambda = 0.1$ and $\lambda = 0.9$, there is a large, almost linear drop in the number of estimators from more than 200 to just under 100. An even steeper drop occurs for $\lambda > 0.9$, but the number of estimators remains above 50 on average, even for $\lambda = 1.0$. Hence, it is conceivable that choosing additional values $\lambda \in [0.9, 1.0]$, and maybe even values $\lambda > 1$ would lead to a selection of even fewer trees below 50, and, therefore, leading to better performance for model sizes below 256 KB.

To further study this phenomenon, we investigate the performance of pruning and leaf-refinement over the number of trees directly. Figure 3 shows the average test accuracy (left column) and average F_1 score (right column) on the chess dataset (top row, average is computed over the cross-validation folds) and eeg dataset (bottom row, average is computed over the cross-validation folds). Please note that we found a similar behavior on the other datasets and hence decided to focus on these two datasets as they show the most distinctive behavior. On the chess dataset, one can see that the L1 + LR never selects less than just under 100 trees, indicating that a more careful choice for λ would have been necessary to select fewer trees. Here, LR with fewer estimators gives a better trade-off between the accuracy (and F_1 score) and the number of trees. Last, we see how pruning can improve the performance over a regular RF: All pruning methods improve over the vanilla RF between 16 and 64 trees and then slowly converge to the original RF's performance. In all cases, methods with leaf-refinement outperform pruning. Looking at the eeg dataset (bottom row), we see a slightly different picture. Here, L1 + LR selects ensembles with 16 to 256 trees

Table 8 The accuracy of each method on each dataset with a model size below 768 KB

dataset	COMP	DREP	IC	IE	L1	L1+LR	LMD	LR	RE	RF
adult	86.48 ± 0.52	86.27 ± 0.46	86.39 ± 0.41	86.4 ± 0.47	86.26 ± 0.41	**86.86 ± 0.26**	86.18 ± 0.51	85.96 ± 0.25	86.44 ± 0.49	86.17 ± 0.42
anuran	98.23 ± 0.32	97.94 ± 0.39	98.17 ± 0.27	98.08 ± 0.46	96.86 ± 0.52	**98.67 ± 0.25**	98.12 ± 0.32	98.65 ± 0.24	98.17 ± 0.42	98.12 ± 0.33
avila	99.11 ± 0.1	97.85 ± 0.28	99.34 ± 0.07	99.29 ± 0.13	88.85 ± 0.26	**99.8 ± 0.04**	91.9 ± 0.42	99.42 ± 0.08	99.2 ± 0.15	97.66 ± 0.23
bank	90.56 ± 0.2	90.37 ± 0.13	90.52 ± 0.18	90.55 ± 0.25	90.4 ± 0.14	**90.96 ± 0.35**	90.41 ± 0.14	90.33 ± 0.35	90.57 ± 0.23	90.37 ± 0.22
chess	63.24 ± 0.51	61.71 ± 0.44	64.04 ± 0.54	62.95 ± 0.19	46.28 ± 0.66	67.76 ± 0.54	62.09 ± 1.06	**73.65 ± 0.7**	63.38 ± 0.62	60.37 ± 0.63
connect	77.35 ± 0.41	76.25 ± 0.2	77.34 ± 0.31	77.29 ± 0.25	74.92 ± 0.33	**83.93 ± 0.21**	76.25 ± 0.32	83.27 ± 0.19	77.1 ± 0.47	76.17 ± 0.25
eeg	92.53 ± 0.25	92.08 ± 0.3	92.5 ± 0.27	92.21 ± 0.28	86.44 ± 0.64	**94.89 ± 0.25**	91.72 ± 0.36	93.96 ± 0.17	92.42 ± 0.25	92.0 ± 0.48
elec	87.95 ± 0.36	87.27 ± 0.31	88.02 ± 0.26	87.94 ± 0.24	85.25 ± 0.22	**91.05 ± 0.21**	87.0 ± 0.25	90.59 ± 0.29	87.75 ± 0.28	87.15 ± 0.4
fashion	86.11 ± 0.09	86.07 ± 0.16	86.2 ± 0.19	85.92 ± 0.26	82.17 ± 0.23	**87.77 ± 0.11**	85.92 ± 0.17	87.12 ± 0.13	86.15 ± 0.08	86.03 ± 0.13
gas-drift	99.44 ± 0.1	99.4 ± 0.11	99.45 ± 0.1	99.42 ± 0.09	98.42 ± 0.22	99.5 ± 0.08	99.41 ± 0.11	**99.52 ± 0.1**	99.4 ± 0.1	99.4 ± 0.12
har	98.1 ± 0.39	97.95 ± 0.37	98.03 ± 0.49	97.81 ± 0.46	96.23 ± 0.29	**98.94 ± 0.41**	97.85 ± 0.44	98.92 ± 0.32	98.09 ± 0.51	97.86 ± 0.36
ida2016	99.25 ± 0.03	99.21 ± 0.02	99.24 ± 0.04	99.23 ± 0.03	99.16 ± 0.01	**99.32 ± 0.03**	99.23 ± 0.04	99.3 ± 0.02	99.25 ± 0.03	99.24 ± 0.02
japanese-vowels	96.75 ± 0.39	96.5 ± 0.33	96.52 ± 0.32	95.97 ± 0.55	91.36 ± 0.46	98.22 ± 0.41	96.46 ± 0.33	**98.23 ± 0.43**	96.56 ± 0.38	96.42 ± 0.3
jm1	81.93 ± 0.39	81.83 ± 0.49	**81.95 ± 0.52**	81.9 ± 0.37	81.66 ± 0.33	81.83 ± 0.53	81.9 ± 0.57	80.6 ± 0.83	81.86 ± 0.51	81.86 ± 0.47
magic	87.86 ± 0.45	87.76 ± 0.42	87.71 ± 0.58	87.75 ± 0.57	87.15 ± 0.55	**88.16 ± 0.5**	87.8 ± 0.54	86.93 ± 0.47	87.84 ± 0.39	87.65 ± 0.65
mnist	95.19 ± 0.13	95.0 ± 0.17	95.2 ± 0.1	94.91 ± 0.16	90.89 ± 0.08	**97.45 ± 0.07**	94.89 ± 0.22	97.38 ± 0.08	95.08 ± 0.14	94.87 ± 0.13
mozilla	95.36 ± 0.44	95.37 ± 0.39	95.36 ± 0.41	95.42 ± 0.37	94.76 ± 0.41	95.52 ± 0.23	95.18 ± 0.41	**95.57 ± 0.25**	95.33 ± 0.36	95.25 ± 0.41
nursery	99.13 ± 0.16	98.81 ± 0.19	99.15 ± 0.19	99.09 ± 0.24	97.02 ± 0.22	99.75 ± 0.08	98.16 ± 0.13	**99.78 ± 0.1**	99.08 ± 0.23	98.8 ± 0.07
postures	95.05 ± 0.18	94.32 ± 0.21	94.94 ± 0.33	94.81 ± 0.16	84.13 ± 0.19	95.73 ± 0.05	94.09 ± 0.14	**96.92 ± 0.08**	94.73 ± 0.2	94.21 ± 0.13
statlog	91.04 ± 0.18	90.88 ± 0.23	90.85 ± 0.15	90.9 ± 0.09	88.87 ± 0.14	91.44 ± 0.25	90.95 ± 0.12	**91.55 ± 0.25**	91.2 ± 0.34	90.89 ± 0.16

For datasets without a dedicated train/test split, the average accuracy and its standard deviation over the cross-validation folds are reported. For datasets with a dedicated train/test split, the experiments are repeated with 5 different random seeds, and the average accuracy and its standard deviation over these repetitions are reported. A dash '–' indicates that a method did not produce any model that fits the memory constraint. Each entry is rounded to the second digit after to decimal point. Each row represents one dataset, and each column is one method. Larger is better. The best method is marked in bold

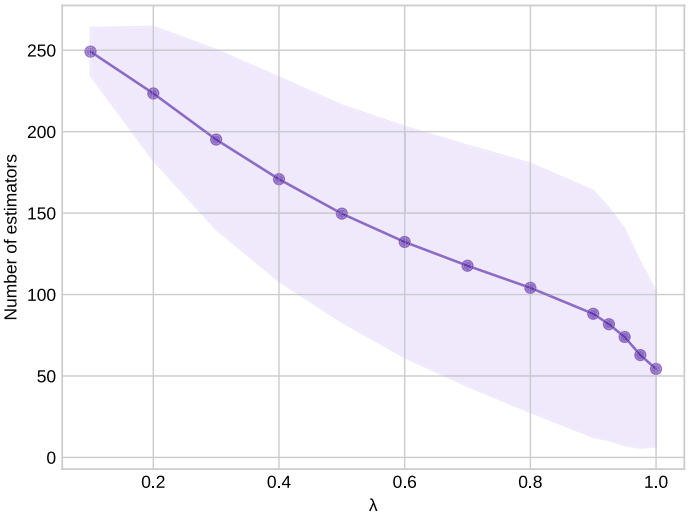

Fig. 2 Average number of estimators across all datasets and configuration of L1 + LR for different λ values. The error band shows the standard deviation

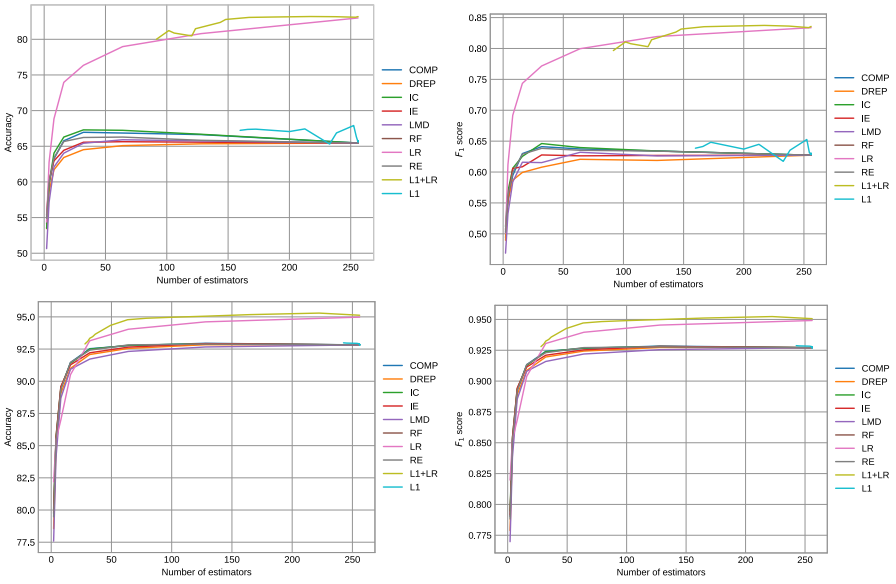

Fig. 3 Average accuracy (left column) and the average F_1 score (right column) across a different number of estimators for the chess dataset (top row, average is computed over the cross-validation folds) and eeg dataset (bottom row, average is computed over the cross-validation folds)

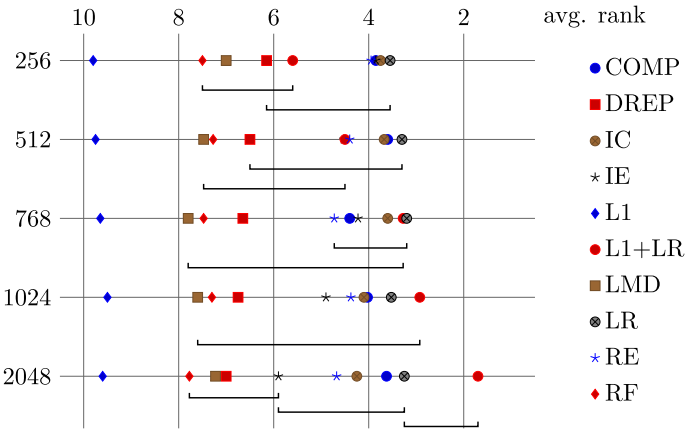

Fig. 4 2 CD-diagram for the accuracy and {256, 512, 768, 1024, 2048} KB memory constraints. For all statistical tests, $p = 0.95$ was used. More to the right (lower rank) is better. The Methods in connected cliques are statistically similar

indicating that λ was better fitting for the problem. Contrary to before, we find that regular leaf-refinement does not perform well for smaller ensembles on this dataset and is outperformed by L1 + LR for ensembles with 200 trees or less.

Similar to the previous section, we want to give a more statistical overview of our findings using CD diagrams. To do so, we expand them into two-dimensional CD diagrams where we apply memory constraints for each level on the y-axis. In the first level, we apply very restrictive constraints, only allowing for models below 256 KB, and plot the average rank of each method. This will likely result in small ensembles of small trees. On the next level, we double the amount of memory allowed to 512 KB and again plot the average rank similar to a 'regular' CD diagram. We repeat this process for all constraints and plot 5 levels with {256, 512, 768, 1024, 2048} KB constraints. Figure 4 shows the CD diagram for accuracy. As indicated by the previous discussion, all methods are relatively close to each other if only limited memory is available. LR is the best method, followed by IC, COMP, RE, RE, L1 + LR, DREP, LMD, RF, and L1 for 256 KB. As discussed previously, L1 on its own is the worst method for all memory constraints. Going to 512 KB constraints, we see that the methods begin to differentiate more but keep their relative ranking. For 765 KB constraints, L1 + LR starts to move up the ranks, now ranking second place, and for 1024 KB and for 2048 KB constraints, it becomes the best method, ranking first place. Figure 5 shows the CD diagram for the F_1 score. The overall plot is similar to Fig. 4: If limited memory is available, then it becomes more difficult to distinguish the performance of single methods, whereas, with more memory available, the average ranks seem to differentiate more. Moreover, L1 is the worst method overall, whereas LR is the best method for 256 and 512 KB constraints, and L1 + LR is the best method for 1024–2048 KB constraints. For 768 KB constraints, there is no clear winner, although LR seems to rank slightly better than L1 + LR.

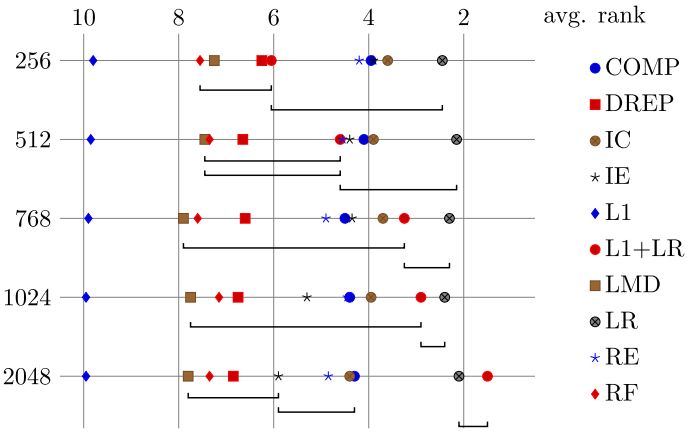

Fig. 5 2 CD-diagram for the F_1 score and {256, 512, 768, 1024, 2048} KB memory constraints. For all statistical tests, $p = 0.95$ was used. More to the right (lower rank) is better. The methods in connected cliques are statistically similar

4.3 Case-study for the PhyNetLab

To showcase the effectiveness of our approach, we will now compare the performance of ensemble pruning, and leaf-refinement in the context of the PhyNetLab warehouse (Masoudinejad et al. 2018). The PhyNetLab is a hardware test platform for the evaluation and analysis of IoT-based warehouses. It consists of small, ultra-low power, energy-neutral devices called PhyNodes that are placed on storage boxes inside the warehouse. The nodes are connected to various access points and form a wireless sensor network. Each node measures the current light intensity, the current temperature, its acceleration as well as the WiFi signal strength to the access points in the warehouse. The goal is to estimate the current position of each node and, thereby, allow for efficient routing and detection of storage boxes in the warehouse. While machine learning is ideally suited for such a task, the challenge lies in the deployment of models. The PhyNode has a MSP430 MCU with a total of 64 KB of Ferroelectric Random Access Memory (FRAM) available, of which 48 KB are accessible by the compact instruction set. Roughly one-third of this memory is already used for the operating system and drivers, leaving about 30 KB of memory for the top-level application, including the model. Subtracting an additional top-level application code of around 10 KB leaves roughly 20 KB for the localization model (Masoudinejad et al. 2018). Therefore, our goal is to find the best localization model that still fits into the remaining 20 KB.

During 42 experiments conducted at various light and temperature levels, a total of 41,431 measurements at 31 different locations inside the warehouse have been taken. Each measurement consists of the acceleration (X,Y,Z) of the box, the current temperature, the current light intensity, as well as the WiFi signal strength to 3 different access points and a unique identifier for each box. During earlier experiments, we noted that the acceleration can have a huge impact on the performance because, in some experiments, the boxes would not be leveled, introducing biases into the acceleration. Hence, the model would over-fit on this feature, although by design, the acceleration

of a (standing) box should not impact the performance of the classification. Hence, we ignore the acceleration in this experiment. To further reduce overfitting against specific environmental properties (e.g., a particular shiny or warm day), we train the models on the data from the first 41 experiments and test them on the last experiment. The resulting training data has $N_{train} = 40{,}444$ samples with $d = 6$ features, $C = 31$ classes, and the test set contains $N_{test} = 987$ test samples[2].

Recall that the model must fit into 20 KB of memory. The size of the implemented model is highly dependent on the specific implementation and can vary across models, MCUs, and implementations. Hence, we perform a two-step process to find good models that fit into 20 KB of RAM: First, we train small models that approximately fit into the memory of the PhyNode. To do so, we estimate the size of the model during training by again counting the total number of nodes n_{total} in all trees inside the forest and then by computing the size via $(17+4\cdot C)\cdot n_{total}$ as outlined above. In the second step, we FastInference[3] to generate the implementation of these models, automatically compile them and remove all models that result in an overflow during the compilation, thereby leaving only models that can actually be deployed to the PhyNode. FastInference is a model compiler that generates model- and CPU-specific inference code for various machine-learning models such as Decision Trees and Random Forests. To do so, FastInference runs in four steps (c.f. Fig. 6): In the first step, the model is loaded from a file into an internal representation. Then, the model is optimized, e.g., by converting floats to a fixed-point quantization, by pruning decision trees[4] etc. Third, the user chooses a backend that determines the target CPU's properties, such as cache size, available memory, etc., as well as the desired implementation. With this information, FastInference re-structures the optimized model such that it can be expressed through a combination of different code templates, which is then realized by a template engine in step 4. The output of this operation is the `C++` inferencing code of an optimized model that can be easily integrated into the compilation toolchain for deployment. FastInference offers two different types of tree implementations, namely native trees that iterate over a static array of nodes using a while-loop and if-else trees that decompose the DT into its if-else structure (c.f. Buschjäger et al. 2018). Unfortunately, if-else trees result in very large code sizes and hence would use too much memory during compilation. Therefore, we chose native trees for this experiment. Some additional pre-processing was required to make the models fit into 20 KB: Recall that there are $C = 31$ classes and, hence, a tree with 16 leaf nodes requires 2 KB to store the class probabilities in leaf nodes if a `float` variable is used. To reduce the memory consumption, we, therefore, employed a fixed-point quantization that scales each probability by a factor of 10,000 and rounds it down towards the next integer. In this way, the probabilities in each leaf node can be stored within a 2 Byte `short` variable, effectively halving the size. This operation is also implemented in FastInference, and we could not detect any change in the accuracy with this quantization.

[2] The data is available under http://phynetlab.com/.

[3] https://github.com/sbuschjaeger/fastinference.

[4] Technically ensemble pruning can be seen as a part of this pre-processing, and we implemented an adapter that integrates the PyPruning library into FastInference.

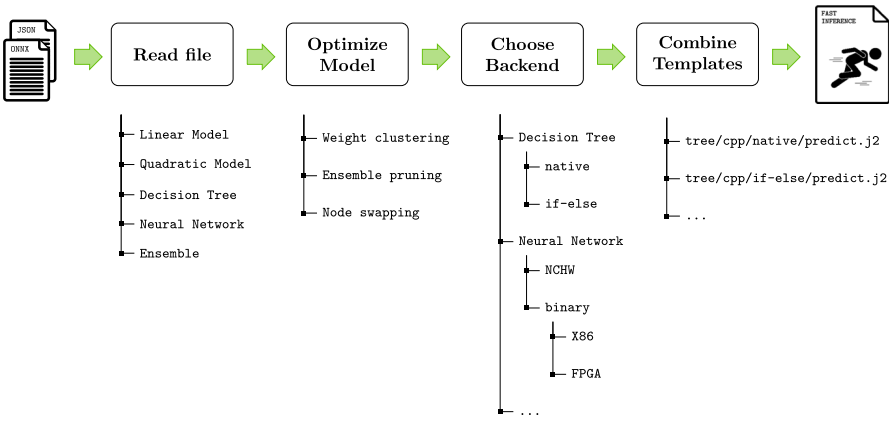

Fig. 6 Workflow of the FastInference model compiler

Table 9 Accuracy (rounded to the second decimal digit) and F_1 score (rounded to the fourth decimal digit) of the best model per method that can still fit into the memory of the PhyNode

	COMP	DREP	IC	IE	L1	L1 + LR	LMD	LR	RE	RF
Accuracy (%)	68.39	51.32	56.50	56.80	62.50	**71.04**	61.28	68.09	63.21	56.40
F_1	0.5757	0.5180	0.4266	0.4383	0.5234	**0.5758**	0.5300	0.5627	0.5163	0.4220

The best model is marked in bold

In a series of pre-experiments, we determined reasonable ranges for the hyperparameters of each algorithm so that the estimated model size is below 24 KB. Similar to before, we train a base Random Forest with $M = 256$ trees and $n_l \in \{4, 8, 12\}$ leaf nodes. Each pruning method is tasked to select $K \in \{2, 4, 8\}$ trees. For DREP, we used $\rho \in \{0.25, 0.3, \ldots, 0.5\}$. For L1 and L1 + LR, we minimized the MSE over 20 epochs with the Adam optimizer using $\alpha = 0.01$, $|\mathcal{B}| = 1024$ and $\lambda \in \{1.0465, 1.0466, \ldots, 1.047\}$. Table 9 shows the accuracy and F_1 score for the best models that could still fit in the PhyNode. As can be seen, L1 + LR offers the best predictive accuracy as well as the best F_1 score, highlighting the usefulness of our approach. Moreover, we found that the accuracy seems to vary a lot between the different methods. For example, DREP is the worst method with 51.32% accuracy, whereas L1 + LR is nearly 20 percentage points better with an accuracy of 71.04%. Given that all models are derived from the RF, these large differences seem surprising to us, but we could not find any errors in our evaluation pipeline. In particular, we made sure that all methods receive the same base forest so that no re-training of the forest would occur.

5 Conclusion

Ensemble algorithms are among the state-of-the-art in many machine learning applications. With the ongoing integration of ML models into everyday life, the deployment

and continuous application of models become more and more important issues. By today's standard, large Random Forests are trained for the best performance which can challenge the resources of small devices and sometimes make deployment impossible. Various techniques have been proposed in the literature that try to reduce the memory consumption of tree ensembles while potentially increasing their performance. In this paper, we studied two common techniques namely ensemble pruning and leaf-refinement. Ensemble pruning removes unnecessary classifiers from the ensemble to reduce the overall resource consumption while potentially improving its accuracy. Leaf-refinement, on the other hand, refines the probability estimates of the trees inside the ensemble by minimizing a global loss. In this paper, we combined both approaches into a single objective and presented an efficient algorithm to optimize it. Our L1 + LR method performs pruning by minimizing an L_1-regularized loss via proximal gradient descent while refining the probability estimates in the leaf nodes at the same time. In a series of 13,200 experiments on 20 publicly available datasets, we showed that L1 + LR has the statistically significant best accuracy and the statistically significant best F_1 score compared with 8 state-of-the-art methods. Moreover, we detailed how these algorithms behave under different memory constraints. We found that if only a very limited amount of memory is available, then L1 + LR and leaf-refinement behave similarly offering better performance than ensemble pruning. If more memory is available, then L1 + LR seems to dominate over vanilla LR making it the overall best choice. Last, we highlighted the usefulness of our approach in a case study using the PhyNetLab. We discussed how to train, prune, and implement small tree ensembles using the FastInference tool and showed how to effectively deploy our models to ultra-low power devices such as the PhyNode.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10618-023-00921-z.

Acknowledgements Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", DFG project number 124020371, SFB project A1, http://sfb876.tu-dortmund.de. Part of this research 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. https://www.ml2r.de/.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Akash PS, Kadir M, Ali AA, Tawhid MNA, Shoyaib M (2019) Introducing confidence as a weight in random forest. In: 2019 international conference on robotics, electrical and signal processing techniques (ICREST). IEEE, pp 611–616

Barros RC, de Carvalho ACPLF, Freitas AA (2015) Decision-tree induction. Springer, Cham, pp 7–45. https://doi.org/10.1007/978-3-319-14231-9_2

Biau G (2012) Analysis of a random forests model. J Mach Learn Res 13(Apr):1063–1095

Biau G, Scornet E (2016) A random forest guided tour. TEST 25(2):197–227

Branco S, Ferreira AG, Cabral J (2019) Machine learning in resource-scarce embedded systems, fpgas, and end-devices: a survey. Electronics 8(11):1289

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (2000) Some infinity theory for predictor ensembles. Technical report, Technical Report 579, Statistics Dept. UCB

Breiman L (2001) Random forests. Mach Learn. https://doi.org/10.1023/A:1010933404324

Brown G, Wyatt JL, Tino P (2005) Managing diversity in regression ensembles. JMLR. https://doi.org/10.1097/IYC.0000000000000008

Buschjäger S, Morik K (2017) Decision tree and random forest implementations for fast filtering of sensor data. IEEE Trans Circuits Syst I Regul Pap 65(1):209–222

Buschjäger S, Morik K (2021) There is no double-descent in random forests. CoRR arXiv:2111.04409

Buschjäger S, Chen K, Chen J, Morik K (2018) Realization of random forest for real-time evaluation through tree framing. In: ICDM, pp 19–28. https://doi.org/10.1109/ICDM.2018.00017

Cavalcanti GD, Oliveira LS, Moura TJ, Carvalho GV (2016) Combining diversity measures for ensemble pruning. Pattern Recogn Lett 74:38–45

Choudhary T, Mishra V, Goswami A, Sarangapani J (2020) A comprehensive survey on model compression and acceleration. Artif Intell Rev 53(7):5113–5155

Cortes C, Mohri M, Syed U (2014) Deep boosting. In: Proceedings of the thirty-first international conference on machine learning (ICML 2014)

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Denil M, Matheson D, De Freitas N (2014) Narrowing the gap: random forests in theory and in practice. In: International conference on machine learning (ICML)

Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42

Giacinto G, Roli F, Fumera G (2000) Design of effective multiple classifier systems by clustering of classifiers. In: Proceedings 15th international conference on pattern recognition. ICPR-2000, vol 2. IEEE, pp 160–163

Guo H, Liu H, Li R, Wu C, Guo Y, Xu M (2018) Margin & diversity based ordering ensemble pruning. Neurocomputing 275:237–246

Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20(8):832–844

Jiang W, Nie F, Huang H (2015) Robust dictionary learning with capped l1-norm. In: Twenty-fourth international joint conference on artificial intelligence

Jiang Z, Liu H, Fu B, Wu Z (2017) Generalized ambiguity decompositions for classification with applications in active learning and unsupervised ensemble pruning. In: 31st AAAI conference on artificial intelligence, AAAI 2017, pp 2073–2079

Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: Bengio Y, LeCun Y (eds) 3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, conference track proceedings. arXiv:1412.6980

Koltchinskii V et al (2002) Empirical margin distributions and bounding the generalization error of combined classifiers. Ann Stat 30(1):1–50

Kumar A, Sindhwani V (2015) Near-separable non-negative matrix factorization with l1 and Bregman loss functions. In: Proceedings of the 2015 SIAM international conference on data mining. SIAM, pp 343–351

Kumar A, Goyal S, Varma M (2017) Resource-efficient machine learning in 2 kb ram for the internet of things. In: International conference on machine learning. PMLR, pp 1935–1944

Lazarevic A, Obradovic Z (2001) Effective pruning of neural network classifier ensembles. In: IJCNN'01, vol 2. IEEE, pp 796–801

Li N, Yu Y, Zhou Z-H (2012) Diversity regularized ensemble pruning. In: ECML PKDD. Springer, pp 330–345
Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2016) Pruning filters for efficient convnets. arXiv:1608.08710
Li J, Cheng K, Wang S, Morstatter F, Trevino RP, Tang J, Liu H (2017) Feature selection: A data perspective. ACM Comput Surv: CSUR 50(6):1–45
Louppe G, Geurts P (2012) Ensembles on random patches. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 346–361
Lu Z, Wu X, Zhu X, Bongard J (2010) Ensemble pruning via individual contribution ordering. In: Proceedings of the ACM SIGKDD, pp 871–880
Lucchese C, Nardini FM, Orlando S, Perego R, Silvestri F, Trani S (2018) X-cleaver: learning ranking ensembles by growing and pruning trees. ACM Trans Intell Syst Technol: TIST 9(6):1–26
Margineantu DD, Dietterich TG (1997) Pruning adaptive boosting. In: ICML, vol 97, pp 211–218
Martínez-Muñoz G, Suárez A (2004) Aggregation ordering in bagging. In: Proceedings of the IASTED, pp 258–263
Martínez-Muñoz G, Suárez A (2006) Pruning in ordered bagging ensembles. In: ICML, pp 609–616
Martínez-Muñoz G, Hernández-Lobato D, Suárez A (2008) An analysis of ensemble pruning techniques based on ordered aggregation. IEEE Trans Pattern Anal Mach Intell 31(2):245–259
Masoudinejad M, Ramachandran Venkatapathy AK, Tondorf D, Heinrich D, Falkenberg R, Buschhoff M (2018) Machine learning based indoor localisation using environmental data in phynetlab warehouse. In: Smart SysTech 2018; European conference on smart objects, systems and technologies, pp 1–8
Oshiro TM, Perez PS, Baranauskas JA (2012) How many trees in a random forest? In: International workshop on machine learning and data mining in pattern recognition. Springer, pp 154–168
Parikh N, Boyd S (2014) Proximal algorithms. Found Trends Optim 1(3):127–239
Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) Pytorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems 32, pp 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830
Ravi KB, Serra J (2017) Cost-complexity pruning of random forests. arXiv:1703.05430
Ren S, Cao X, Wei Y, Sun J (2015) Global refinement of random forest. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 723–730
Schapire RE, Freund Y (2012) Boosting: foundations and algorithms. The MIT Press, Cambridge
Shahhosseini M, Hu G (2020) Improved weighted random forest for classification problems. In: International online conference on intelligent decision science. Springer, pp 42–56
Shahhosseini M, Hu G, Pham H (2022) Optimizing ensemble weights and hyperparameters of machine learning models for regression problems. Mach Learn Appl 7:100251
Shotton J, Sharp T, Kohli P, Nowozin S, Winn J, Criminisi A (2013) Decision jungles: compact and rich models for classification. In: NIPS'13 proceedings of the 26th international conference on neural information processing systems, pp 234–242
Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc Ser B (Methodol) 58(1):267–288
Tsoumakas G, Partalas I, Vlahavas IP (2009) An ensemble pruning primer. In: Okun O, Valentini G (eds) Applications of supervised and unsupervised ensemble methods, Studies in computational intelligence, vol 245. Springer, pp 1–13
Zhang Y, Burer S, Street WN (2006) Ensemble pruning via semi-definite programming. J Mach Learn Res 7(Jul):1315–1338
Zhou Z-H (2012) Ensemble methods: foundations and algorithms. CRC Press, Boca Raton. https://doi.org/10.1201/b12207
Zyblewski P, Woźniak M (2019) Clustering-based ensemble pruning and multistage organization using diversity. In: Pérez García H, Sánchez González L, Castejón Limas M, Quintián Pardo H, Corchado Rodríguez E (eds) Hybrid artificial intelligent systems. Springer, Cham, pp 287-298

Zyblewski P, Woźniak M (2020) Novel clustering-based pruning algorithms. Pattern Anal Appl 23(3):1049–1058