Alexander Grabner, BSc

Loss-Specific Training of Memory Efficient Random Forests for Super-Resolution

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Telematik

submitted to

Graz University of Technology

Supervisor
Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof
Institute for Computer Graphics and Vision

Advisors
Dipl.-Ing. Georg Poier, BSc
Dipl.-Ing. Michael Opitz, BSc
Dipl.-Ing. Dr.techn. Samuel Schulter, BSc
Institute for Computer Graphics and Vision

Graz, Austria, April 2016

Abstract

Super-Resolution (SR) addresses the problem of image upscaling by reconstructing high-resolution (HR) images from low-resolution (LR) images. Recently Random Forest (RF) approaches have shown state of the art accuracy in single image *SR* while almost achieving real time capable speed. However, existing *RF* approaches for *SR* have a large memory footprint, since complex models are required to achieve high performance.

This limits the practical utilization of *RFs* for real world *SR* applications. Especially mobile devices like smartphones or tablets demand memory efficient solutions, since they are limited in resources like RAM and flash storage.

In this work, we present three novel methods for constructing *RFs* with reduced model size: Global Refinement of Alternating Decision and Regression Forests (ADRFs+GR), Additive Global Refinement (AGR) and Intermediate Refined Random Forests (IRRFs). These methods construct *RFs* with low complexity under a global training objective. Due to global optimization, we achieve improved fitting power for *RFs* with low model size. In particular, our methods combine and extend recent approaches on loss-specific training of *RFs* and training of memory efficient *RFs*. In contrast to previous works, we train *RFs* with globally optimized structure and globally optimized prediction models.

We evaluate our proposed methods for standard machine learning tasks and single image *SR*. Our methods show significantly reduced model size while achieving competitive performance compared to state of the art *RF* approaches. Additionally, our training approach is significantly faster than other approaches, which reduce the model size of *RFs* without compromising on accuracy.

Kurzfassung

Super-Resolution (SR) behandelt das Problem der Hochskalierung von Bildern durch Rekonstruktion von Bildern mit hoher Auflösung aus Bildern mit niedriger Auflösung. Vor Kurzem haben Random Forest (RF) basierte Ansätze für Einzelbild *SR* Qualität auf dem neuesten Stand der Technik gezeigt und beinahe echtzeitfähige Geschwindigkeit erreicht. Allerdings weisen bestehende *RF* basierte Ansätze für *SR* einen hohen Speicherbedarf auf, da komplexe Modelle benötigt werden, um hohe Genauigkeit zu erzielen.

Dieser Umstand schränkt den praktischen Nutzen von *RFs* für *SR* Anwendungen in der realen Welt ein. Besonders mobile Geräte wie Smartphones oder Tablets benötigen speichereffiziente Lösungen, da ihre Ressourcen wie RAM oder Flash Speicher beschränkt sind.

In dieser Arbeit präsentieren wir drei neue Methoden, um *RFs* mit reduzierten Speicheranforderungen zu trainieren: Global Refinement of Alternating Decision and Regression Forests (ADRFs+GR), Additive Global Refinement (AGR) und Intermediate Refined Random Forests (IRRFs). Diese Methoden erzeugen *RFs* mit geringer Komplexität unter Berücksichtigung einer globalen Zielfunktion. Durch die globale Optimierung erzielen wir verbesserte Ergebnisse für *RFs* mit geringer Modellgröße. Im Speziellen kombinieren und erweitern unsere Methoden kürzlich vorgestellte Ansätze für fehlerspezifisches Training von *RFs* und Training von speichereffizienten *RFs*. Im Gegensatz zu existierenden Methoden, trainieren wir *RFs* mit global optimierter Struktur und global optimierten Vorhersagemodellen.

Wir evaluieren unsere vorgestellten Methoden für Standardaufgaben im Bereich des maschinellen Lernens und Einzelbild *SR*. Unsere Methoden erzeugen *RFs* mit signifikant reduzierter Modellgröße, deren Genauigkeit konkurrenzfähig zu *RF* basierten Ansätzen auf dem neuesten Stand der Technik ist. Zusätzlich ist unser Trainingsansatz signifikant schneller als andere Ansätze, die *RFs* mit reduzierten Speicheranforderungen erzeugen ohne Abstriche in Bezug auf die Qualität zu machen.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Contents

List of Figures

List of Tables

List of Acronyms

ADF	Alternating Decision Forest
ADRF	Alternating Decision and Regression Forest
ADRF+GR	Global Refinement of Alternating Decision and Regression Forest
AGR	Additive Global Refinement
ARF	Alternating Regression Forest
CNN	Convolutional Neural Network
DT	Decision Tree
FLOP	Floating-Point Operation
GP	Global Pruning
GR	Global Refinement
HR	high-resolution
IRRF	Intermediate Refined Random Forest
LR	low-resolution
MR	Misclassification Rate
PSNR	Peak Signal-to-Noise Ratio
RF	Random Forest
RMSE	Root Mean Squared Error
SR	Super-Resolution

CHAPTER 1

Introduction

Super-Resolution (SR) addresses the problem of high quality image upscaling. *SR* is an active research area, since today many devices like smartphones, tablets, notebooks or TVs feature high resolution displays. These devices often have to deal with low resolution content which results in a degraded user experience. Popular tasks like web browsing, image messaging or photo streaming suffer from poor image quality due to limited data bandwidth, processing power or memory capacity. Therefore, image upscaling methods have to be accurate, fast and memory efficient. However, existing *SR* approaches do not meet all of these requirements.

Single image *SR* is a subdomain of image reconstruction which addresses the problem of enhancing image resolution [30, 37, 71]. Starting from a single low-resolution (LR) input image, a visually pleasing high-resolution (HR) output image is estimated. *SR* methods perform image upscaling without losing the sharpness of the original *LR* image. This upscaling is nontrivial, because one pixel in the *LR* input image has to account for multiple pixels in the *HR* output image. Therefore, *SR* is an ill-posed problem for which no unique solution exists.

Many approaches for populating the pixels of *HR* output images have been proposed. The most popular class of *SR* algorithms are interpolation methods [24, 46, 51, 72]. Bicubic interpolation [54] is the standard approach for image upscaling in image editing applications like Adobe Photoshop® or GIMP®. Interpolation methods are fast and memory efficient, but lack accuracy.

SR methods based on machine learning [5] techniques show significantly improved accuracy compared to interpolation methods [4, 12, 97, 98, 99, 100]. In contrast to interpolation methods, these methods are often slow. However, recently machine learning methods have shown state of the art results in terms of accuracy while almost achieving real time capable speed [16, 22, 23, 45, 55, 56, 93, 94].

Among these methods are Random Forest (RF) approaches [88]. *RF* approaches for *SR* are among the fastest machine learning approaches and achieve high accuracy. However, existing *RF* approaches have a large memory footprint, since they require complex models for high performance. This limits the practical utilization of *RFs* for real world *SR* applications.

One reason for this memory inefficiency is that *RFs* [8] are based on ensembles of Decision Trees (DTs) [74]. The node count of balanced *DTs* grows exponentially with the depth. One extra level of depth doubles the total node count. For *RFs* with 100 *DTs* and a maximum depth of 25 [77], the total node count is 3.36 billion assuming balanced *DTs*. This is particularly problematic, since *RFs* perform best with a high number of deep *DTs* in most scenarios. Especially on mobile devices, large *RFs* easily exceed memory limitations.

Reducing the total node count of *RFs* is a popular approach for increasing the memory efficiency. Two strategies for reducing the total node count of *RFs* are pre-pruning [29, 35] and post-pruning [35, 75, 76]. Other methods replace *DTs* with more memory efficient structures [70, 91]. However, significant improvements in memory efficiency are prone to result in reduced accuracy.

One approach to compensate for reduced accuracy is loss-specific optimization of *RFs*. Experiments show that the accuracy of *RFs* for *SR* is improved by loss-specific optimization which corresponds to constructing *RFs* under a global training objective [88]. Standard *RFs* do not explicitly optimize a global loss [89, 90]. Each *DT* of a *RF* is trained independently and greedily minimizes its local training objective. However, the final result of a *RF* is computed by combining the predictions of the individual *DTs*. The combination of the locally optimal *DT* predictions is not guaranteed to result in a globally optimal prediction. The final model structure is not considered during training. Ideally, the loss function is evaluated on the final output of the *RF* [77]. In contrast, the loss function implied by standard *RFs* is an average over the losses of the individual *DTs*. As a result, the training of standard *RFs* is not directly guidable towards the optimization of a specific loss function.

To overcome this limitation, a number of methods have been proposed which make the individual *DTs* of *RFs* aware of each other [3, 49, 57, 79, 81]. These approaches focus on improving the performance of *RFs* by sharing information between the individual *DTs* and construct complementary *DT* ensembles [89, 90].

Global Refinement (GR) of *RFs* [77] is a recently proposed *RF* approach which addresses both model size reduction and loss-specific optimization of *RFs*. *GR* is a promising candidate for reducing the model size of *RFs* for *SR*. However, the presented training algorithm is prohibitively slow for large problems like *SR*. Additionally, *GR* is limited to constant leaf prediction models, while state of the art *RF* approaches for *SR* use linear leaf prediction models.

To address these issues, we present three new algorithms which construct complementary *DT* ensembles with reduced model size. Our methods are inspired by Alternating Decision and Regression Forests (ADRFs) [89, 90] and *GR* of *RFs* [77]. *ADRFs* grow the structure of *RFs* under a global training objective, while *GR* relearns the leaf prediction models of pre-trained *RFs* under a global training objective. We combine and extend the ideas of both approaches. In contrast to previous works, we train *RFs* with globally optimized structure and globally optimized prediction models. Additionally, we present an extension to *GR* which supports the refinement of linear prediction models.

We use our novel methods to construct *RFs* with low complexity under a global training objective. Due to global optimization, we achieve improved fitting power for *RFs* with low model size while significantly reducing the training time compared to the training approach presented in *GR* of *RFs* [77]. We provide a systematic evaluation of our methods on standard machine learning tasks for classification and regression. Additionally, we compare the performance of our proposed methods to state of the art *RF* approaches.

Finally, we show that our methods are applicable to *SR*. Our methods show significantly reduced model size while achieving competitive performance compared to state of the art *RF* approaches. In contrast to previous works, we present a *SR* approach which is accurate, fast and memory efficient. Additionally, our training approach is significantly faster than other approaches, which reduce the model size of *RFs* without compromising on accuracy.

The remainder of this work is structured as follows: In Chapter 2 we present preliminaries and discuss related works. Next, we present our novel training methods in Chapter 3. In Chapter 4 we evaluate our methods and compare them to existing *RF* approaches. We perform experiments on standard machine learning tasks and single image *SR*. In Chapter 5 we summarize our results and draw conclusions.

CHAPTER 2

Preliminaries

In this chapter we present multiple tree-based approaches to supervised machine learning. We start with an introduction to machine learning. Next, we cover Decision Trees (DTs) and Random Forests (RFs) which are ensembles of randomized *DTs*. Additionally, we present two *RF* extensions, Alternating Decision and Regression Forests (ADRFs) and Global Refinement (GR) of *RFs*. Finally, we briefly discuss different related works. The methods presented in this chapter build the foundation for the contribution of this work.

2.1 Machine Learning

Machine learning [5] is a discipline of computer science which explores the prediction of future outcomes based on knowledge obtained from past observations [34]. A formal definition is provided by Mitchell [66]: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"*. However, machine learning is not only targeted at the development of learning systems, but also studies human learning processes and explores application independent learning methods [65]. In this work we focus on supervised offline machine learning [33] to address classification and regression tasks with *RFs* [61].

2.2 Decision Trees

Decision Trees (DTs) [74] are learners which follow a divide and conquer strategy. *DTs* make predictions based on a number of hierarchical decisions. These decisions are organized in the structure of a tree. In this work we focus on binary trees. The model

of a DT is shown in Figure 2.1. Each node in a DT has one parent node – except for the root node – and zero or two child nodes. Nodes which have two children are split nodes, whereas nodes which do not have any children are leaf nodes. Each split node holds a split parameter set $\boldsymbol{\Theta}$, which is used to decide to which of the two child nodes an arriving data sample is routed. Each leaf node holds a prediction model m, which is used to make predictions for arriving data samples.

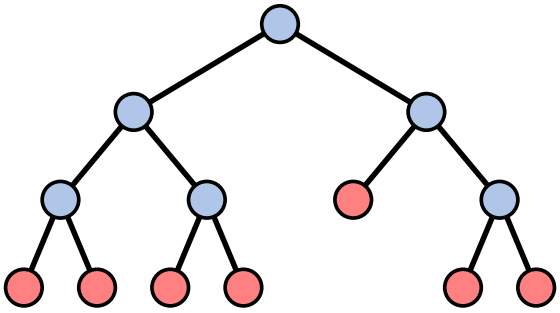


Figure 2.1: An illustration of a DT. Split nodes are colored blue, while leaf nodes are colored red. Each split node holds a split parameter set $\boldsymbol{\Theta}$. Each leaf node holds a prediction model m.

2.2.1 Testing

Testing a DT corresponds to making predictions for previously unseen data. Figure 2.2 shows the evaluation of a DT given a new data sample $\mathbf{x}$. The data sample is sent to the root node of the DT, from where it is passed down the tree. Starting from the root node each visited split node evaluates a binary split function

$$\sigma(\mathbf{x}, \boldsymbol{\Theta}) \in \{0, 1\} \tag{2.1}$$

based on the data sample $\mathbf{x}$ and a split parameter set $\boldsymbol{\Theta}$ which is stored at the specific split node. The outcome of this function determines whether the data sample is sent to the left (0) or right (1) child node. The data sample is routed down the tree from node to node until it ends up in a leaf node. The prediction model m stored at this leaf node is evaluated to generate $\hat{\mathbf{y}}_t$, the prediction of the DT.

2.2.2 Training

Training a DT corresponds to constructing a DT based on a training data set. This involves growing the tree structure, finding a split parameter set for each split node and calculating a prediction model for each leaf node. The training procedure is essentially the same for classification and regression tasks [9], except for the optimization of split functions and the calculation of prediction models.

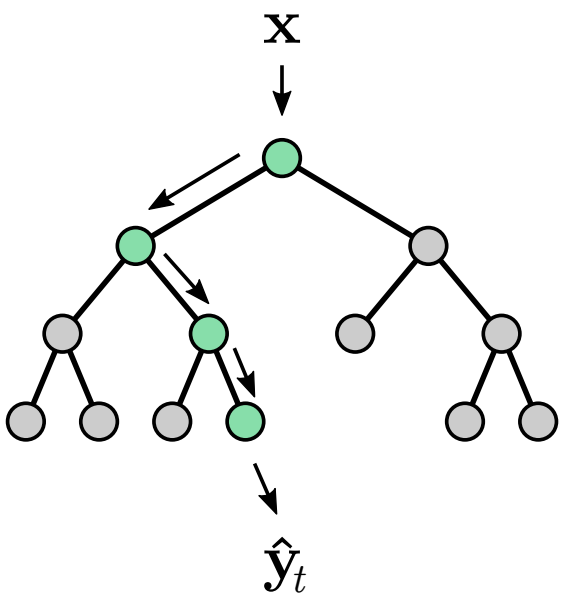


Figure 2.2: An illustration of making a prediction for a previously unseen data sample $\mathbf{x}$ using a DT. The data sample $\mathbf{x}$ is sent to the root of the DT, from where it is passed down the tree until it reaches a leaf node. The prediction model m stored at this leaf node is evaluated to obtain $\hat{\mathbf{y}}_t$, the prediction of the DT.

The training data set $S = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ consists of pairs composed of an input $\mathbf{x}_n \in \mathcal{X}$ and a desired output $\mathbf{y}_n \in \mathcal{Y}$. The input $\mathbf{x}_n$ is also referred to as feature vector, the desired output $\mathbf{y}_n$ is also referred to as ground truth. The input space $\mathcal{X}$ and the output space $\mathcal{Y}$ can be arbitrary, e.g., class labels, continuous variables, strings or graphs [20, 58]. In this work we focus on $\mathcal{X} \subseteq \mathbb{R}^F$ with $F \in \mathbb{N}$ and $\mathcal{Y} \subseteq \mathbb{N}$ for classification or $\mathcal{Y} \subseteq \mathbb{R}^G$ with $G \in \mathbb{N}$ for regression.

Growing the tree structure is concerned with recursively splitting the training data set S into disjoint subsets in a greedy manner. Each node splits the arriving training data into two subsets by evaluating a binary split function (see Section 2.2.2.1) and creates two new child nodes. One subset is sent to the left child node, the other subset is sent to the right child node. This means that the entire training data set is distributed among all nodes in one depth level of the DT. In depth level zero the entire training data set is associated with the root node. In depth level one a fraction of the training data set goes to the left child of the root node, while the rest is sent to the right child. This can be seen as a partitioning of the input space $\mathcal{X}$. The deeper a DT, the more fine-grained the partitioning and the smaller the subsets. The partitioning is greedily continued until a stopping criterion is met (see Section 2.2.2.2). When a stopping criterion is met, a node is turned into a leaf node and a prediction model for this leaf node is calculated using the arriving training data (see Section 2.2.2.3). The training ends when the recursive splitting for all branches of the DT stopped.

2.2.2.1 Split Functions

In order to split the training data each node evaluates a binary split function $\sigma(\mathbf{x}, \boldsymbol{\Theta})$ (see Equation 2.1) for each arriving training sample. In this work we focus on simple thresholding functions in the form of unary split functions

$$\sigma(\mathbf{x}, \boldsymbol{\Theta}) = \begin{cases} 0 & \text{if } \mathbf{x}[\Theta_{feat1}] < \Theta_{thresh} \\ 1 & \text{otherwise} \end{cases} \tag{2.2}$$

and binary split functions

$$\sigma(\mathbf{x}, \boldsymbol{\Theta}) = \begin{cases} 0 & \text{if } \mathbf{x}[\Theta_{feat1}] - \mathbf{x}[\Theta_{feat2}] < \Theta_{thresh} \\ 1 & \text{otherwise} \end{cases}, \tag{2.3}$$

where one feature dimension or the difference of two feature dimensions of $\mathbf{x}$ is compared against a threshold. In this case the split parameter set $\boldsymbol{\Theta}$ consists of the threshold Θ_{thresh} and the indices of the observed feature dimensions Θ_{feat1} and Θ_{feat2} with $\Theta_{feat1} \neq \Theta_{feat2}$. One strategy to find a split parameter set $\boldsymbol{\Theta}$ for a node is random parameter selection [17, 44]. At each node a number of candidate split parameter sets $\{\boldsymbol{\Theta}_k\}_{k=1}^{K}$ with $K \in \mathbb{N}$ is proposed at random. Each of these candidate sets is evaluated and the split parameter set

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}_k} E(S, \boldsymbol{\Theta}_k) \quad \forall k \in [1, K] \tag{2.4}$$

which produces the best split on the arriving training data according to an objective function $E(S, \boldsymbol{\Theta})$ is selected [15]. The objective function $E(S, \boldsymbol{\Theta})$ measures the quality of a split with respect to specific properties of the training data. In this work we focus on an objective function in the form

$$E(S, \boldsymbol{\Theta}) = \sum_{i \in \{L,R\}} |S^i| \cdot H(S^i). \tag{2.5}$$

This objective function calculates the cost of a split by evaluating a compactness measure $H(S)$ for each of the two subsets generated by a split [27]. The two subsets S^L and S^R are implicitly dependent on the split parameter set $\boldsymbol{\Theta}$ (see Equation 2.2 and 2.3). The total cost of a split is computed as the weighted sum of the two compactness results, where $|S^i|$ identifies the size of the specific subset. For classification, we focus on the Shannon entropy

$$H(S) = -\sum_{c \in \mathcal{Y}} p(c|S)\ log(p(c|S)), \tag{2.6}$$

where $p(c|S)$ is the probability of a label given a data set [9]. Other possible choices for $H(S)$ considering classification include the Gini impurity and the Towing splitting

rule [82, 83]. For regression, we focus on the variance of the training data ground truth

$$H(S) = \frac{1}{|S| - 1} \sum_{i=1}^{|S|} \|\mathbf{y}_i - \bar{\mathbf{y}}\|_2^2, \tag{2.7}$$

where $|S|$ is the size of the data set and $\bar{\mathbf{y}}$ is the empirical mean of the data set ground truth [9]. Other possible choices for $H(S)$ considering regression include the differential entropy and the absolute deviation [9, 13]. Compactness measures may not only be defined on the training data ground truth, but also on the features [27, 84]. For regression, it is possible to compute $H(S)$ as the weighted sum of the variance of the ground truth and the variance of the features.

2.2.2.2 Stopping Criteria

The partitioning of the input space $\mathcal{X}$ via split nodes is greedily continued until a stopping criterion is met [2, 63]. This is also known as early stopping or pre-pruning [29, 35]. Possible criteria for turning a node into a leaf node include: a) a maximum tree depth is reached; b) the number of training samples arriving at a node is below a predefined threshold; c) the split function parameter set $\boldsymbol{\Theta}^*$ does not satisfy predefined requirements, e.g., the split would send all samples to one of the two child nodes or the split would only add a negligible performance gain; d) the error on the training data is below a predefined threshold; e) the training data arriving at a node is pure, e.g., all samples have the same ground truth. Multiple stopping criteria can be employed together.

2.2.2.3 Prediction Models

The prediction model is a function that makes a prediction for samples falling into a leaf [9]. The parameters of this function are computed from the training data arriving at the leaf. The prediction model m may be a constant, linear or non-linear function [27, 84]. In this section we focus on constant prediction models. For constant prediction models, the prediction for each sample falling into a specific leaf will be the same. In this case, it is possible to precompute the prediction of a leaf from the arriving training data. For classification, a constant prediction label can be selected as the most frequented label among the ground truth labels of the arriving training data. A constant class distribution vector can be calculated as the normalized histogram

$$m : \hat{\mathbf{y}}_t[c] = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{I}[\mathbf{y}_i = c] \qquad \forall\, c \in \mathcal{Y} \tag{2.8}$$

of the ground truth labels of the arriving training data, where $\mathbb{I}[\cdot]$ is an indicator function which is 1 when the condition is satisfied and 0 otherwise. $|S|$ is the size of the arriving training data set. For regression, a constant prediction can be computed as the empirical

mean of the ground truth of the arriving training data

$$m : \hat{\mathbf{y}}_t = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbf{y}_i. \tag{2.9}$$

2.3 Random Forests

Random Forests (RFs) [8] are ensembles of randomized *DTs*. Each *DT* of the ensemble is independent. *RFs* improve upon single *DTs* in terms of accuracy on previously unseen data, a property called generalization [1, 18, 43, 44]. *DTs* are powerful learners that can fit the training data perfectly when fully grown. However, growing deep *DTs* introduces a high risk of overfitting which results in low generalization [9]. The decomposition of the generalization error into bias, variance and noise shows that a single deep *DT* has low bias and high variance [21]. An ensemble of *DTs* can reduce variance while maintaining low bias [33, 69, 83].

2.3.1 Testing

Figure 2.3 shows the evaluation of a *RF* given a previously unseen data sample $\mathbf{x}$. During testing each *DT* of the *RF* is evaluated (see Section 2.2.1) and the predictions $\hat{\mathbf{y}}_t$ of the individual *DTs* are combined to obtain $\hat{\mathbf{y}}$, the prediction of the *RF*. One option for computing $\hat{\mathbf{y}}$ is the arithmetic mean

$$\hat{\mathbf{y}} = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{y}}_t \tag{2.10}$$

of the different tree predictions $\hat{\mathbf{y}}_t$ [8], where T is the number of *DTs* in the ensemble. This approach is suitable when $\hat{\mathbf{y}}_t$ is a class distribution vector or a continuous variable. If $\hat{\mathbf{y}}_t$ is a class label, $\hat{\mathbf{y}}$ can be selected as the most frequented label among the labels predicted by the various *DTs*.

2.3.2 Training

Training a *RF* corresponds to constructing T distinct *DTs*. Randomness is injected into the training of each individual *DT* to create different trees. In addition to random parameter selection (see Section 2.2.2), *RFs* employ Bagging [7]. Bagging which is short for bootstrap aggregation is an ensemble technique [69]. Each *DT* is trained on a new training data set which is created by randomly drawing samples from the original training data set following a uniform distribution. The size of the new training data set can be smaller than the original. The sampling can be done with or without replacement. Another method for injecting randomness into the training of *DTs* is subsampling at node level [53], e.g., subsampling the training data set arriving at a node before the evaluation

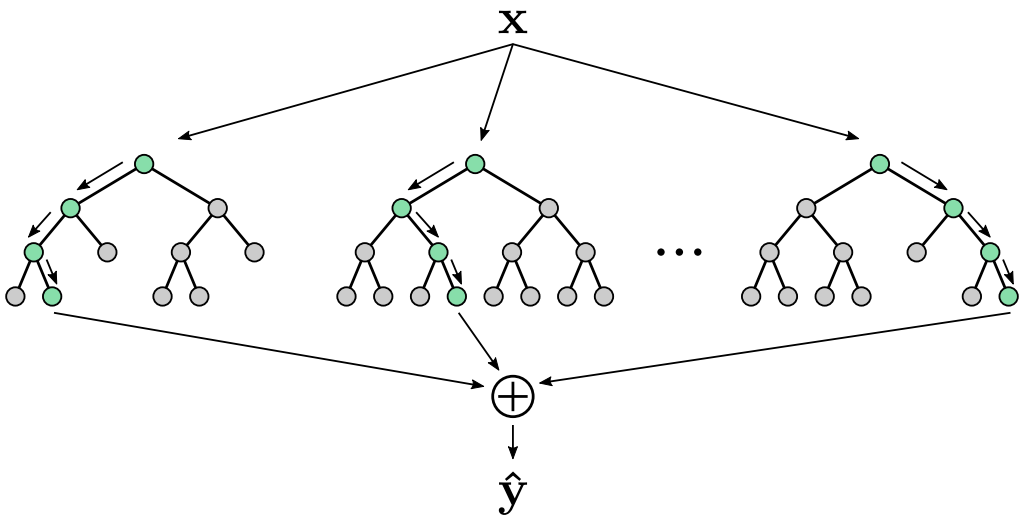


Figure 2.3: An illustration of making a prediction for a previously unseen data sample $\mathbf{x}$ using a *RF*. The data sample $\mathbf{x}$ is sent to the root of each *DT*, from where it is passed down until it reaches a leaf node. The prediction model m stored at this leaf node is evaluated to obtain $\hat{\mathbf{y}}_t$, the prediction of the *DT*. The predictions of the individual *DTs* are combined to obtain $\hat{\mathbf{y}}$, the prediction of the *RF*.

of the candidate split parameter sets $\{\boldsymbol{\Theta}_k\}_{k=1}^{K}$ or before the calculation of the prediction model m. Randomized training is essential for the performance of *RFs*. Breiman [8] shows that the generalization error of *RFs* depends on the strength and correlation of the individual *DTs*. A low generalization error is obtained by high strength and low correlation. The injection of randomness into the training can decrease the correlation of the individual *DTs* while maintaining strength. Breiman [8] also shows that *RFs* do not overfit when more *DTs* are added to the ensemble, as long as the correlation between the individual *DTs* is low.

2.4 Alternating Decision and Regression Forests

ADRFs [89, 90] modify the training procedure of standard *RFs* (see Section 2.3) by incorporating ideas from Boosting [86]. *ADRFs* integrate the minimization of a global loss function directly into the tree growing process. In contrast to Boosted Trees [31], where *DTs* are added sequentially to the model, *ADRFs* iteratively increase the depth of *DTs*, while the number of *DTs* remains constant.

2.4.1 Boosting

Boosting [86] is an ensemble technique. Multiple weak learners are combined to form a strong learner. A weak learner is a predictor which only performs slightly better than random guessing, e.g., a *DT* stump [31]. In contrast to *RFs*, boosting methods iteratively

add predictors to the model. Each new predictor is trained to compensate for the error of the model up to this point. The prediction of the ensemble is computed as the weighted sum of the different weak learner predictions.

Gradient Boosting [32] is a variant of Boosting that interprets Boosting as performing gradient decent in function space. Gradient Boosting can be used to optimize any differentiable loss function. The main idea is that each new predictor applies a step in the steepest decent direction. The steepest decent direction is computed by the negative gradient of a differentiable loss function given the current state of the model. Gradient Boosting uses shrinkage to reduce the contribution of new predictors to the existing model. Experiments show that small learning rates result in improved generalization, but more predictors have to be trained to achieve similar accuracy [32, 42]. For *ADRFs*, shrinkage is omitted, because small learning rates result in deep *RFs* with large memory demands.

2.4.2 Stage-Wise Training

The main difference of *ADRFs* compared to standard *RFs* (see Section 2.3) is the replacement of the greedy *DT* training procedure by a stage-wise training scheme. Figure 2.4 shows the iterative stage-wise training scheme. The entire *RF* is trained in a breadth-first manner. Instead of greedily training all *DTs* independent of each other, the depth of the *RF* is iteratively increased stage by stage. One stage corresponds to one level of depth of the *RF*. Each iteration extends the *RF* by one extra level of depth, which is equivalent to splitting all leaf nodes of the current model. Increasing the depth of the *RF* can be seen as adding a new predictor in the sense of Boosting.

The stage-wise training scheme produces a fully functional *RF* in each iteration. These intermediate *RFs* are used to evaluate the performance of the current model. The results are used to guide the training of the next stage towards a solution that makes up for the error of the *RF* in its current state. Similar to Gradient Boosting [32], *ADRFs* rely on the negative gradients

$$-\mathbf{g}_d(\mathbf{x}) = -\left[\frac{\partial \mathcal{L}(\mathbf{y}, F(\mathbf{x}))}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{d-1}(\mathbf{x})} \tag{2.11}$$

of a differentiable loss function $\mathcal{L}(\cdot)$ to influence the training of the next stage. The loss function is evaluated on $F_{d-1}(\mathbf{x})$, the prediction of the *RF* trained up to depth $d-1$. The training of *ADRFs* alternates between updating the global training objective and growing a new stage of the *RF*.

2.4.3 Alternating Decision Forests

Alternating Decision Forests (ADFs) [90] are targeted at classification tasks. Similar to Adaboost [31], *ADFs* introduce an importance weight for each training sample. These

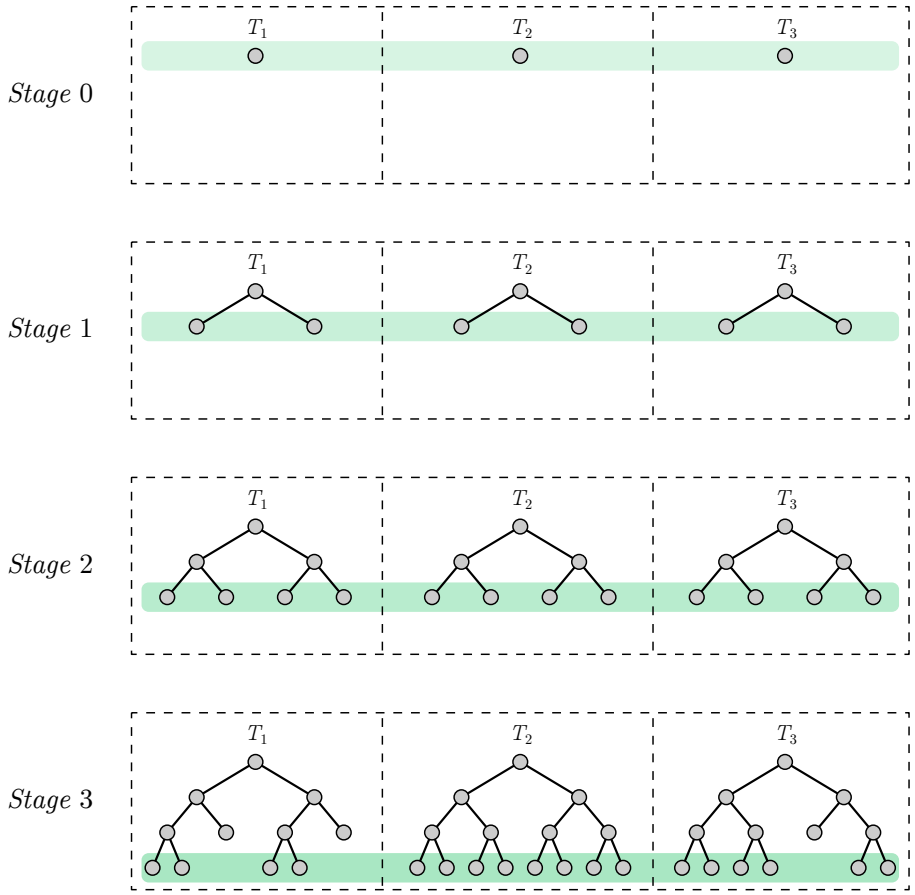


Figure 2.4: An illustration of the stage-wise training scheme of *ADRFs*. Adding a new stage corresponds to splitting all leaf nodes in the current model. This may not be possible for all nodes due to different stopping criteria which results in unbalanced trees.

importance weights are updated after the training of each stage. The weight updates are computed from the negative gradients $-\mathbf{g}_d(\mathbf{x})$ of a differentiable loss function (see Equation 2.11) which is defined over the classification margin

$$margin(\mathbf{x}) = F(\mathbf{y}_{GT}|\mathbf{x}) - \max_{\mathbf{y} \neq \mathbf{y}_{GT}} F(\mathbf{y}|\mathbf{x}). \tag{2.12}$$

The classification margin is the difference between the class probability of the ground truth and the highest class probability apart from the ground truth. A classification margin equal to 1 corresponds to a perfect prediction for a sample. The probability of the ground truth is 1, while the probability of any other class is 0. On the other side, a margin equal to -1 implies that the probability of the ground truth label is 0. The classification margin is a performance indicator for the current prediction. Training samples with a margin close to 1 receive a low importance weight. In contrast, training samples with a margin close to -1 receive a high importance weight. The training of the next stage focuses on

samples with a high importance weight. *ADFs* globally optimize the splits of the *DTs* in a way that the combination of the different tree predictions yields less misclassification on the training data. *ADFs* do not modify the prediction models. Experiments show that the *DTs* of *ADFs* are less correlated than the *DTs* of standard *RFs* [87]. This results in improved generalization.

2.4.4 Alternating Regression Forests

Alternating Regression Forests (ARFs) [89] are targeted at regression tasks. *ARFs* modify the training data before adding a new stage. The ground truth in the original training data set $S = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ is replaced with the negative gradients of a differentiable loss function (see Equation 2.11). This gives the intermediate training data set $S_{\mathbf{g}_d} = \{(\mathbf{x}_n, -\mathbf{g}_d(\mathbf{x}_n))\}_{n=1}^{N}$. One possible choice for the differentiable loss function is the squared loss

$$\mathcal{L}(\mathbf{y}, F(\mathbf{x})) = \frac{1}{2}(\mathbf{y} - F(\mathbf{x}))^2 \tag{2.13}$$

which reduces the calculation of the negative gradients to

$$-\mathbf{g}_d(\mathbf{x}) = \mathbf{y} - F_{d-1}(\mathbf{x}). \tag{2.14}$$

$F_{d-1}(\mathbf{x})$ is the prediction of the *ARF* trained up to depth $d-1$. Each new stage is trained on these pseudo-targets or residuals instead of the ground truth. Therefore, the predictions of the leaf nodes correct the error of the previous stages instead of making independent predictions. This means that the predictions of all stages $\hat{\mathbf{y}}^d$ must be summed up to obtain the final prediction $\hat{\mathbf{y}}$. Since the path to each leaf in a binary tree is unique, it is possible to precompute the summation of the different stage predictions $\hat{\mathbf{y}}_t^d$ within a *DT* during training. This can be formalized as

$$\begin{aligned}
\hat{\mathbf{y}} &= \sum_{d=0}^{D_{max}} \hat{\mathbf{y}}^d \\
&= \sum_{d=0}^{D_{max}} \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{y}}_t^d \\
&= \frac{1}{T} \sum_{t=1}^{T} \sum_{d=0}^{D_{max}} \hat{\mathbf{y}}_t^d \\
&= \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{y}}_t.
\end{aligned} \tag{2.15}$$

T is the number of *DTs* and D_{max} is the maximum depth of the *ARF*. For each leaf all prediction models along the path from the root to the leaf are summed up. The resulting prediction model is stored at the corresponding leaf node. Thus, the testing procedure and the model size are the same as for standard *RFs*.

2.5 Global Refinement of Random Forests

GR of *RFs* [77] relearns the leaf prediction models of existing *RFs*. Starting from a pre-trained *RF* the leaf nodes of all *DTs* are simultaneously retrained by optimizing a global loss function. The refinement procedure is applicable to *RFs* for classification and regression using constant prediction models. Additionally, Ren *et al.* [77] present a pruning method which reduces model size while maintaining high accuracy.

2.5.1 Global Refinement

In [77] the prediction process $F(\mathbf{x})$ of a *RF* is formulated as single matrix multiplication

$$\hat{\mathbf{y}} = F(\mathbf{x}) = W\Phi(\mathbf{x}), \tag{2.16}$$

where W is referred to as the leaf matrix and $\Phi(\mathbf{x})$ as the indicator vector. The structure of W and $\Phi(\mathbf{x})$ is shown in Figure 2.5. Each column of the leaf matrix W corresponds to the prediction model of one leaf node. The number of rows of W equals the output dimensionality. The number of columns of W equals the total number of leaf nodes of the *RF*. The indicator vector $\Phi(\mathbf{x})$ is a binary embedding which gives information about which leaf nodes a sample falls into. The number of rows of $\Phi(\mathbf{x})$ equals the number of columns of W. This means that there is a row for each leaf of the *RF*. Each binary entry of $\Phi(\mathbf{x})$ indicates whether the sample $\mathbf{x}$ falls into the leaf identified by the specific row (1) or not (0). It is possible to stack the indicator vectors of multiple samples together

$$\Phi(\mathbf{X}) = \begin{bmatrix} \Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_2) & \cdots & \Phi(\mathbf{x}_N) \end{bmatrix}, \tag{2.17}$$

where the number of columns of $\Phi(\mathbf{X})$ equals the number of samples N.

Given the training data set $S = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ the indicator matrix $\Phi(\mathbf{X})$ is derived from the existing *RF* and the training data features. *GR* minimizes the objective function

$$\min_W \frac{1}{2}\|W\|_F^2 + \frac{C}{N}\sum_{n=1}^{N} \mathcal{L}(\mathbf{y}_n, \hat{\mathbf{y}}_n)) \tag{2.18}$$

by calculating the optimal leaf matrix W. $\mathcal{L}(\cdot)$ is a convex loss function defined on the training data ground truth $\mathbf{y}$ and the prediction $\hat{\mathbf{y}}$ (see Equation 2.16). The Euclidean norm of the leaf matrix W is minimizes to reduces the risk of overfitting [92]. The parameter C controls the tradeoff between the regularization term and the data term. Convex optimization algorithms can be used to find the global minimum of this objective function [6]. The refined leaf prediction models in W are used to overwrite the existing prediction models of the *RF*.

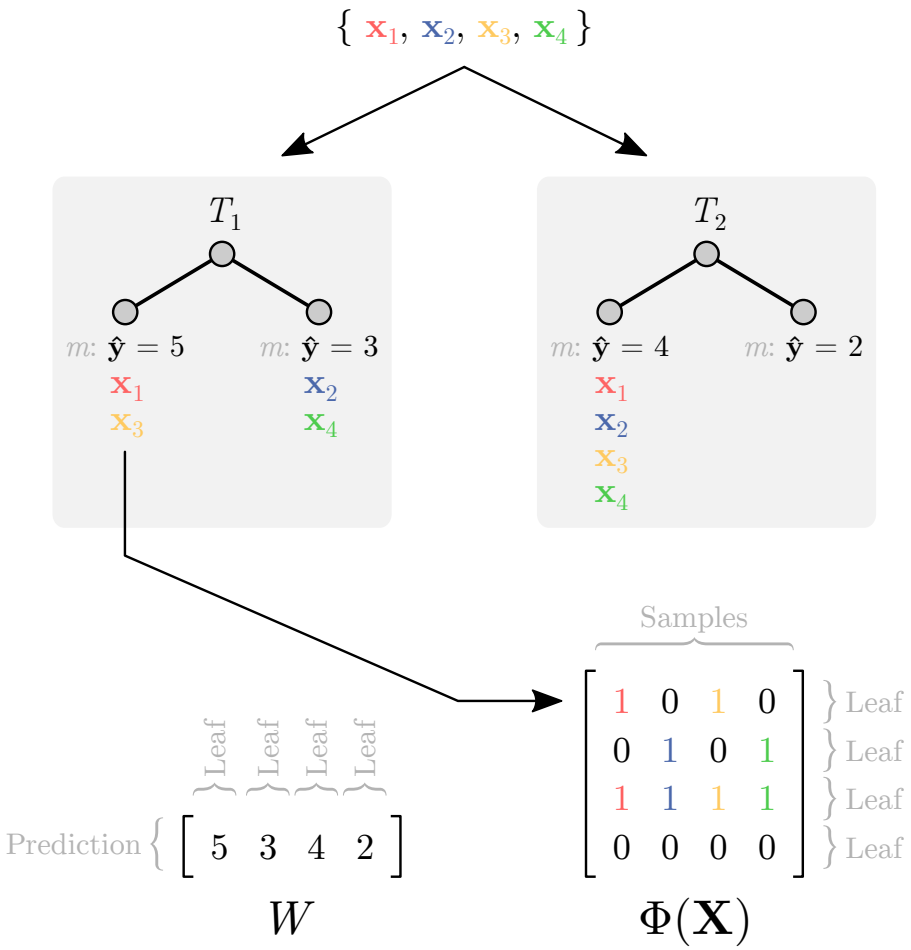


Figure 2.5: An illustration of the structure of the leaf matrix W and the indicator matrix $\Phi(\mathbf{X})$. Four data samples $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_4$ are sent to the root of each DT, from where they are passed down until they reach a leaf node. Considering T_1, the samples $\mathbf{x}_1$ and $\mathbf{x}_3$ fall into the left child node. This leaf node is associated with the first row of the indicator matrix $\Phi(\mathbf{X})$, therefore, the first and third entry of this row are set to 1. The prediction models of all leaf nodes are horizontally concatenated to form the leaf matrix W.

2.5.2 Global Pruning

Although the objective function utilizes regularization techniques, *GR* has a high risk of overfitting. Therefore, [77] additionally presents Global Pruning (GP). *GP* iteratively reduces the total number of nodes of a *RF* to avoid overfitting. Traditional post-pruning methods [35, 75, 76] independently merge the leaf nodes of individual *DTs*. *GP* merges the leaf nodes of a *RF* using a global optimization strategy. *GP* consists of two steps. First, *GR* is applied to the *RF*. Second, all mergeable leaf nodes of the *RF* are identified and a significance measurement for each leaf pair is calculated. The significance is the summation of the Euclidean norms of the two prediction models. The smaller the significance the lower the contribution of the leaves. Therefore, a fraction of the least significant leaf pairs is

merged. Next, *GR* is reapplied to the pruned *RF*. As a result, the algorithm alternates between refining and pruning the *RF* until a stopping criterion is met. *GP* stops when the *RF* achieves the best accuracy on a validation set or a certain model size is reached.

2.6 Related Works

In this section we briefly discuss related works on tree-based machine learning approaches which either reduce the model size or explicitly optimize a global loss function. Our work differs from these methods, since we reduce the model size of *RFs* and explicitly optimize a global loss function.

2.6.1 Decision Jungles

Decision Jungles [91] build upon generic rooted directed acyclic graphs instead of binary trees. This opens the possibility for multiple paths ending in the same leaf node. During training nodes with similar properties are merged. Decision Jungles uses the same objective function to jointly split and merge nodes. Similar to *RFs*, each directed acyclic graph is trained independently of each other. Decision Jungles show improved memory efficiency and generalization compared to *RFs*. Experiments show that for specific tasks, Decision Jungles achieve competitive performance compared to *RFs* in terms of accuracy with a significantly reduced total node count [91].

2.6.2 Random Ferns

Random Ferns [70] simplify *RFs* by dropping the hierarchical split node model. The same sequence of split functions is applied to each data sample. In contrast to *RFs*, Random Ferns directly index a leaf node using the binary sequence obtained by evaluating split functions in a predefined order. This results in improved memory efficiency and a speedup for training and testing. For certain applications like keypoint recognition, Random Ferns show competitive performance compared to *RFs* in terms of accuracy [60].

2.6.3 Dynamic Random Forests

Dynamic Random Forests [3] use a Boosting [86] approach to construct *RFs*. Dynamic Random Forests sequentially add *DTs* to the *RF*. Each new *DT* is trained on a new weighted training data set. This training data set is generated by Bagging. Additionally, an importance weight is assigned to each bagged sample to guide the training of the next *DT*. In contrast to Boosting, the weights for each new *DT* are calculated according to the entire *RF* under construction. Due to the sequential training, *DTs* are not independent of each other. The training of multiple *DTs* is not parallelizable.

2.6.4 Decision and Regression Tree Fields

Decision and Regression Tree Fields [50, 68] combine *RFs* and conditional random fields. In contrast to *RFs*, each leaf node stores the parameters of a local quadratic energy function. Each DT is associated with a factor type of a conditional random field. All local energy functions are combined to obtain the overall energy function which is minimized using a closed form solution. It is possible to jointly learn the structure of the *DTs* and the leaf parameters optimizing any differentiable loss function [49].

2.6.5 Globally Optimal Fuzzy Decision Trees

Globally Optimal Fuzzy Decision Trees [79] modify *DTs* to minimize a specified loss function. The binary splits of *DTs* are replaced with fuzzy sigmoidal splits. This means a sample takes all possible paths within a DT instead of a single branch. A samples ends up in each leaf node with a specific membership probability. The final prediction of a DT is made jointly by all leaf nodes instead of one leaf node. The training algorithm transforms a pre-trained DT into a fuzzy DT and then applies backpropagation in combination with an optimization algorithm to refine the fuzzy split parameters according to a specified loss function.

2.6.6 Deep Neural Decision Forests

Deep Neural Decision Forests [57] combine *RFs* with the representational learning of deep Convolutional Neural Networks (CNNs). Similar to Globally Optimal Fuzzy Decision Trees (see Section 2.6.5), Deep Neural Decision Forests employ fuzzy probabilistic splits and use an optimization algorithm which performs backpropagation on existing *DTs*. In contrast to Globally Optimal Fuzzy Decision Trees, Deep Neural Decision Forests retrain the predictions of the leaf nodes and optimize a RF instead of a single DT. The training algorithm uses a dropout strategy to update individual *DTs* and to optimize a global loss function defined over the entire RF.

2.6.7 Relating Cascaded Random Forests to Deep Convolutional Neural Networks

Richmond *et al.* [81] present a mapping from cascaded *RFs* to deep *CNNs* and an approximate mapping back. In [81] this mapping is used to intelligently initialize a *CNN* with a greedily trained cascaded RF. After the refinement of the parameters, the *CNN* is mapped back to a cascaded RF. The back mapped cascaded RF shows improved performance compared to the greedily trained cascaded RF and accelerated evaluation compared to the Convolutional Neural Network.

CHAPTER 3

Loss-Specific Training of Random Forests

In this chapter we present three new methods for constructing Random Forests (RFs) which optimize a global loss function over all Decision Trees (DTs). The methods combine the ideas of Alternating Decision and Regression Forests (ADRFs) (see Section 2.4) and Global Refinement (GR) (see Section 2.5). Additionally, we present an extension to *GR* which supports the refinement of linear prediction models. This extension can be used in combination with the three presented training algorithms.

3.1 Global Refinement of Alternating Decision and Regression Forests

The most basic combination of *GR* and *ADRFs* is to apply a *GR* to an Alternating Decision Forest (ADF) or an Alternating Regression Forest (ARF). It is possible to perform the leaf prediction model optimization of *GR* on top of any *RF*. Therefore, we apply *GR* to a *RF* which was grown using the alternating training procedure of *ADRFs* (see Section 2.4). There are no difficulties in combining the two approaches, since they are performed sequentially and independent of each other. We call this method Global Refinement of Alternating Decision and Regression Forests (ADRFs+GR). Figure 3.1 shows the pseudo code of *ADRFs+GR* in comparison to other approaches. *RFs* trained by *ADRFs+GR* feature optimized splits (*ADRFs*) and optimized prediction models (*GR*). The possibility of a naive combination of *GR* and *ADRFs* is suggested in [77].

3.2 Additive Global Refinement

GR discards the existing prediction models of a *RF* and replaces them with new prediction models. This refinement strategy shows potential for improvement, since it does not take advantage of the existing prediction models of a *RF*, but simply ignores them. In contrast to *GR*, we propose to use the existing prediction models as a starting point. We perform a global leaf prediction model optimization to calculate prediction models which compensate for the error of the existing prediction models. In this way, we improve the existing prediction models of a *RF* instead of relearning them from scratch. We call this refinement technique Additive Global Refinement (AGR).

AGR can be seen as a variant of *GR* which performs a Gradient Decent [6] step on an existing *RF*. The leaf prediction model optimization is used to apply a step in the steepest decent direction. The steepest decent direction is computed by the negative gradient of a differentiable loss function given the current state of the model. Figure 3.1 shows the pseudo code of *AGR* in comparison to other approaches. Similar to *ARFs* (see Section 2.4.4), *AGR* modifies the training data before new prediction models are calculated. The ground truth in the original training data set $S = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ is replaced with the negative gradients of a differentiable loss function (see Equation 2.11). This gives the intermediate training data set $S_{\mathbf{g}_d} = \{(\mathbf{x}_n, -\mathbf{g}_d(\mathbf{x}_n))\}_{n=1}^{N}$. The gradient decent step is implemented by optimizing

$$\min_{W} \frac{1}{2}\|W\|_F^2 + \frac{C}{N}\sum_{n=1}^{N} \mathcal{L}(-\mathbf{g}_d(\mathbf{x}_n), \hat{\mathbf{y}}_n), \tag{3.1}$$

a modified version of the objective function presented in *GR* (see Equation 2.18). In contrast to Equation 2.18, the convex loss function $\mathcal{L}(\cdot)$ is evaluated on the negative gradients $-\mathbf{g}_d(\mathbf{x})$ (see Equation 2.11) and the prediction $\hat{\mathbf{y}}$ (see Equation 2.16). All other parameters remain the same. The prediction models obtained by *AGR* correct the error of the existing prediction models instead of making an independent prediction. This means that the new prediction models and the existing prediction models are summed up to obtain the refined prediction models. Therefore, the new prediction models in W are added to the existing prediction models of the *RF*, instead of replacing them (see Equation 2.15). *AGR* is applicable to *RFs* for regression and *ARFs*.

3.3 Intermediate Refined Random Forests

The stage-wise training scheme of *ADRFs* produces a fully functional *RF* in each iteration. These intermediate *RFs* are used to evaluate the performance of the current model and guide the training of the next stage. We propose to refine each of these intermediate *RFs*. We integrate multiple global leaf prediction model optimizations directly into the training of *ADRFs*. In this way, growing the structure of *DTs* and refinement are interweaved.

We call this method Intermediate Refined Random Forests (IRRFs). In contrast to single refinement approaches which relearn the leaf prediction models of pre-trained *RFs*, *IRRFs* refine the prediction models of each stage of *ADRFs* during construction. The stages are trained sequentially and each new stage corrects the error of the previous stages. Therefore, intermediate refinements influence the training of the next stage. As a result, the refinement of intermediate *RFs* optimizes the leaf prediction models as well as the structure of the individual *DTs*.

The training of *ADRFs* alternates between updating the global training objective and growing a new stage of the *RF*. In contrast to this, the training of *IRRFs* cycles through four phases in each iteration. The first two phases are identical to *ADRFs*. First, the global training objective is updated. Second, all leaf nodes of the current *RF* are split and prediction models for the newly emerged leaf nodes are calculated locally using the training data arriving at the node. Third, the global training objective is updated again. Fourth, a global refinement technique is applied to the current *RF*.

For regression, we apply *AGR* to each intermediate *RF* during the construction of *ARFs*. Figure 3.1 shows the pseudo code of *IRRFs* for regression in comparison to other approaches. In this case, we choose *AGR* over *GR* for the refinement. The reason for this is that *AGR* improves the existing prediction models of a *RF* instead of relearning them from scratch like *GR*. Additionally, *AGR* better complements the additive local leaf prediction models of *ARFs* in this context.

For classification, the interweavement of growing the structure of *DTs* and refinement shows incompatibilities. While it is possible to apply *GR* to each intermediate *RF* during the construction of *ADFs*, updating the global training objective is difficult in this case. The reason for this is that refined *RFs* for classification predict SVM decision values [11], but the loss functions of *ADFs* are designed for classification margins computed from normalized class probabilities [90]. Although there are techniques for mapping SVM decision values to class probability estimates, e.g., soft-max [5] or platt-scaling [73], we observe low performance compared to standard *RFs*. Alternatively, the Hinge loss is compatible with SVM decision values, but the selection of the threshold for this loss function is unclear and experiments show that the performance of *ADFs* with Hinge loss is inferior to standard *RF* [90]. Because of this, we focus on *IRRFs* for regression.

3.4 Global Refinement for Linear Prediction Models

GR for linear prediction models is an extension to *GR* (see Section 2.5.1) which supports the optimization of linear leaf prediction models. In [77] *GR* only covers constant prediction models. This means that the prediction for each sample falling into a specific leaf will be the same. (see Section 2.2.2.3). Linear prediction models compute the prediction as a function of the features of a sample [27]. In this work we focus on linear

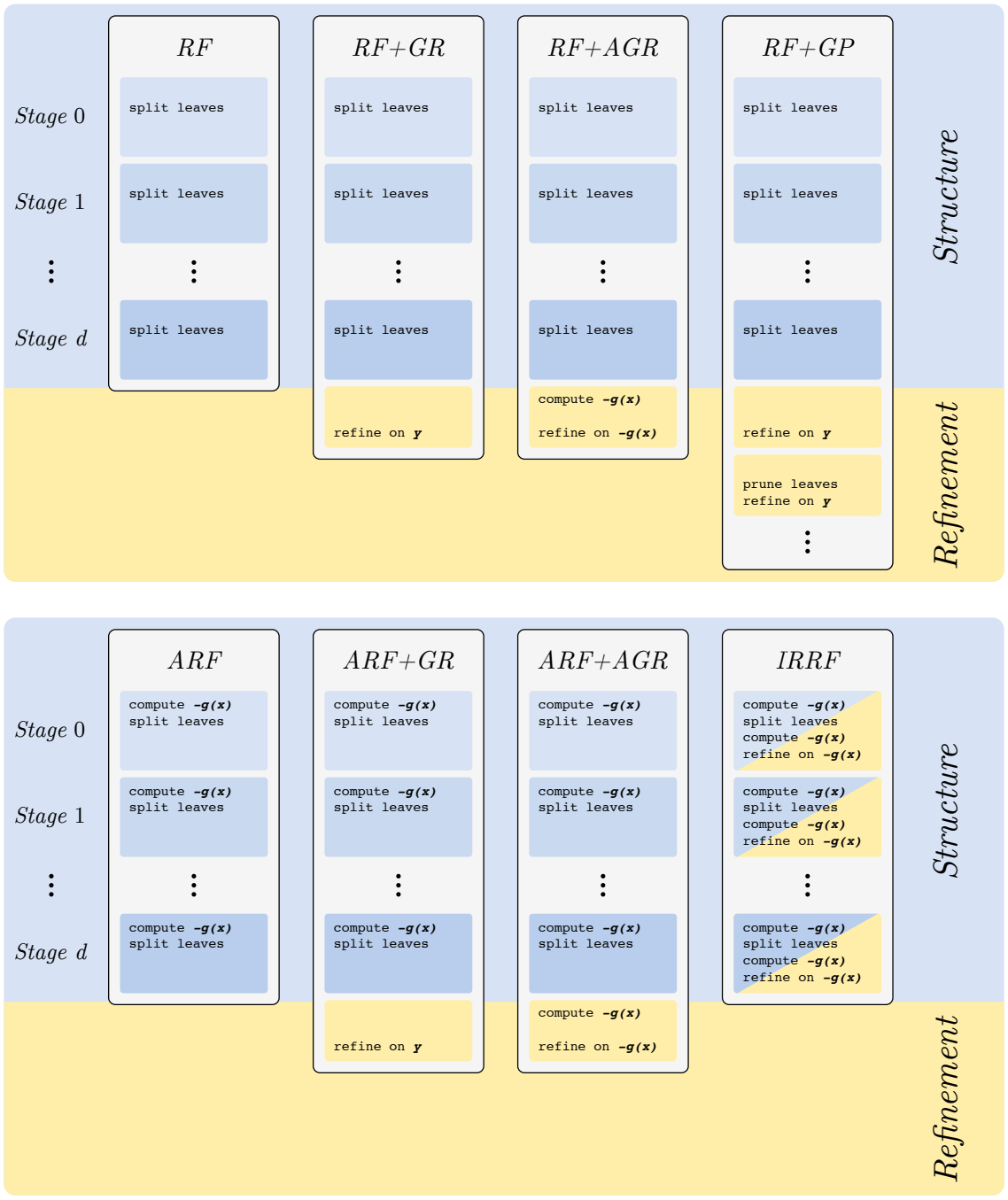


Figure 3.1: A comparison of different training algorithms for *RFs* targeted at regression. We assume a stage-wise growing scheme which is applicable to all methods. The pseudo code highlights the differences of the various approaches. Different algorithms for growing the structure of *DTs* are combined with different refinement techniques. The code snippets associated with growing structure of *DTs* are colored blue, while code snippets associated with refinement are colored yellow. $\mathbf{y}$ identifies the training data ground truth. In contrast, $-\mathbf{g}(\mathbf{x})$ identifies the pseudo targets computed from the training data features and ground truth.

prediction models for regression in the form

$$m : \hat{\mathbf{y}}_t(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{F} w_i x_i, \tag{3.2}$$

where $\mathbf{w}$ is a weight vector and $\mathbf{x}$ the feature vector of a sample. F is the number of features or the dimensionality of $\mathbf{x}$. *GR* for linear prediction models finds optimal weights $\mathbf{w}$ for each leaf node by optimizing the same objective function as *GR* (see Equation 2.18). However, the prediction of the *RF* which is a parameter of the objective function is computed differently. We use the same notation as *GR* (see Equation 2.16)

$$\hat{\mathbf{y}}(\mathbf{x}) = F(\mathbf{x}) = W\Phi_{LIN}(\mathbf{x}), \tag{3.3}$$

but replace the indicator vector $\Phi(\mathbf{x})$ with the feature indicator vector $\Phi_{LIN}(\mathbf{x})$. The structure of W and $\Phi_{LIN}(\mathbf{x})$ is shown in Figure 3.2. In contrast to *GR*, W horizontally concatenates the weight vectors of all leaf nodes. The feature indicator matrix $\Phi_{LIN}(\mathbf{x})$ is a container for the feature vector $\mathbf{x}$ which also gives information about which leaf nodes a sample falls into. Each entry of $\Phi_{LIN}(\mathbf{x})$ is associated with one weight of W. This means that each leaf node is associated with F entires of $\Phi_{LIN}(\mathbf{x})$. F is the dimensionality of $\mathbf{x}$. If the sample $\mathbf{x}$ falls into a leaf, the associated F entries of $\Phi_{LIN}(\mathbf{x})$ are set to the corresponding features of $\mathbf{x}$. Otherwise, the associated F entries are set to zero. It is possible to stack the feature indicator vectors of multiple samples together

$$\Phi_{LIN}(\mathbf{X}) = \begin{bmatrix} \Phi_{LIN}(\mathbf{x}_1) & \Phi_{LIN}(\mathbf{x}_2) & \cdots & \Phi_{LIN}(\mathbf{x}_N) \end{bmatrix}, \tag{3.4}$$

where the number of columns of $\Phi_{LIN}(\mathbf{X})$ equals the number of samples N.

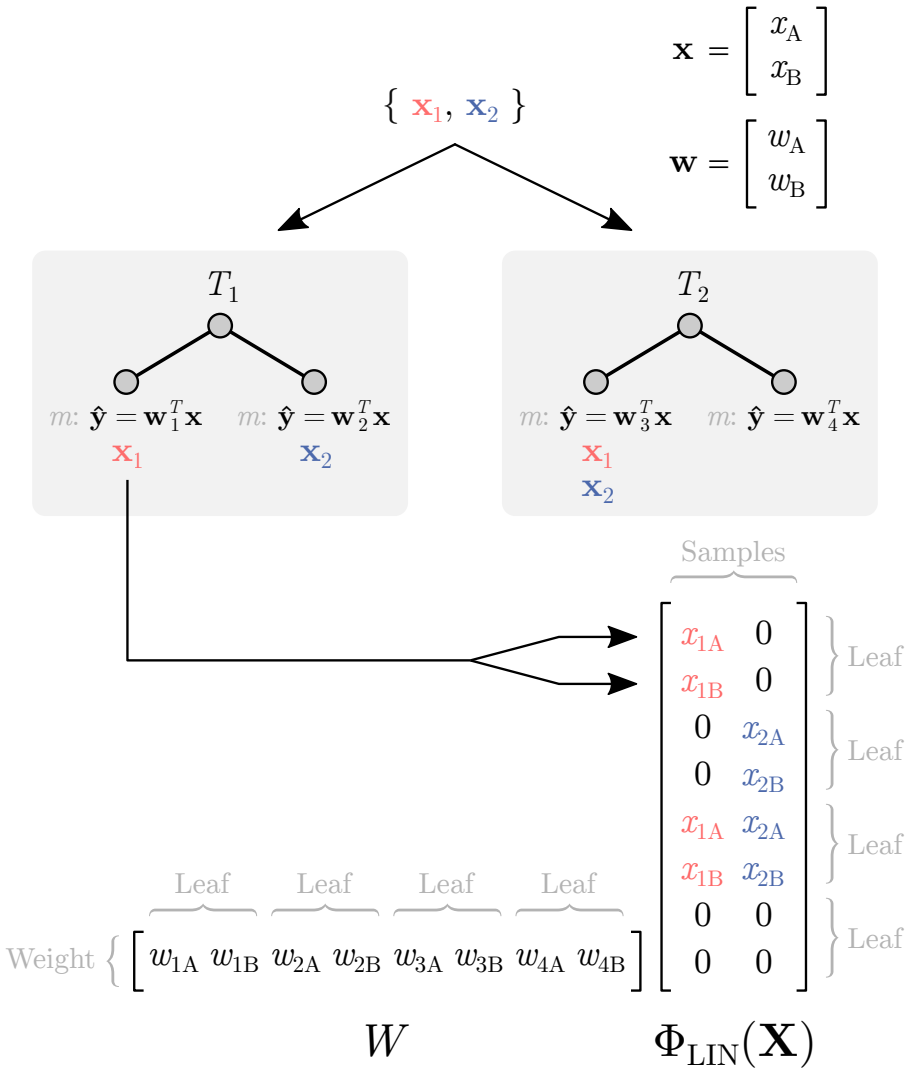


Figure 3.2: An illustration of the structure of the leaf matrix W and the feature indicator matrix $\Phi_{LIN}(\mathbf{X})$. The two data samples $\mathbf{x}_1$ and $\mathbf{x}_2$ are sent to the root of each DT, from where they are passed down until they reach a leaf node. Considering T_1, the sample $\mathbf{x}_1$ falls into the left child node. This leaf node is associated with the first and second row of the feature indicator matrix $\Phi_{LIN}(\mathbf{X})$, therefore, the first entry of the two rows is set the corresponding feature values x_{1A} and x_{1B}. The weight vectors of all leaf nodes are horizontally concatenated to form the leaf matrix W.

CHAPTER 4

Evaluation

In this chapter we evaluate the methods presented in this work. We compare our proposed methods (see Chapter 3) against existing approaches (see Chapter 2). We perform experiments on standard machine learning tasks and single image Super-Resolution (SR). Our experiments emphasize regression, but also present results for classification. We evaluate different parameters and analyze specific properties of various Random Forest (RF) approaches.

4.1 Standard Machine Learning Tasks

In this section we evaluate different *RF* approaches for standard machine learning tasks. Standard machine learning tasks are publicly available data sets for different applications [62]. These data sets can be used to benchmark different machine learning algorithms. In this work we focus on data sets for classification and regression.

For data sets which do not provide a dedicated split into training and test samples, we randomly split the data set into 60% training samples and 40% test samples [77, 89]. We use standard performance evaluation metrics for classification and regression [10]. We perform 5 independent runs for each experiment [89, 90]. All Figures and Tables report the mean and the standard deviation of the independent runs.

We combine different algorithms for growing the structure of Decision Trees (DTs) with different refinement techniques. We report the performance of different methods on all data sets. Additionally, we evaluate different parameters which are common to all evaluated methods. In this case, we fix all parameters except for one and vary this parameter within a specified range. The reason for this is that different methods perform best with different parameters. Our experiments show that the performance of some

methods is strongly affected by specific parameters, while other methods are insensitive to the same parameters.

For parameter evaluation, we do not present results for each data set. A detailed parameter evaluation for all data sets goes beyond the scope of this work. Therefore, we report results on a single data set. For each parameter, we present results on a different data set to provide an unbiased evaluation. The reported results are representative for the behavior across different data sets. Unless otherwise stated, the observed behavior is reproducible on other data sets.

Our implementation is written in MATLAB® and based upon the code of Dollár *et al.* [19] and Schulter *et al.* [88]. For the implementation of refinement techniques we use LIBLINEAR [26]. We use Eigen [39] to accelerate specific implementation steps.

4.1.1 Classification

The evaluated methods for classification are summarized in Table 4.1. We evaluate two non-refinement approaches (*RF* and Alternating Decision Forest (ADF)) and three refinement approaches. We employ Global Refinement (GR) for the leaf prediction model optimization.

We do not present results for Intermediate Refined Random Forest (IRRF), because this method shows incompatibilities for classification. Additionally, we do not present results for *ADF+GP*, since Global Pruning (GP) has a number of disadvantages which we discuss in Section 4.1.2.8. However, we evaluate *RF+GP*, since this method shows state of the art performance for different machine learning tasks in terms of *RF* approaches [77]. In most experiments our proposed methods achieve competitive performance compared to *RF+GP*.

Methods - Classification

	Training	
Name	Structure	Refinement
RF	RF	-
RF+GR	RF	GR
RF+GP	RF	GP
ADF	ADF	-
ADF+GR	ADF	GR

Table 4.1: Evaluated methods for classification.

We evaluate the different methods on 6 data sets for classification. The properties of the data sets are summarized in Table 4.2. All 6 data sets represent multiclass classification problems.

Data Sets - Classification

Data Set	Train	Test	Features	Classes
char74k	66 707	7 400	64	62
letter	16 000	4 000	16	26
mnist	60 000	10 000	784	10
pendigits	7 494	3 498	16	10
protein	14 895	6 621	357	3
usps	7 291	2 007	256	10

Table 4.2: Evaluated data sets for classification.

We evaluate the performance by computing the Misclassification Rate (MR) between the ground truth and the prediction [10, 77, 90]. The lower the *MR*, the better the performance. Unless otherwise stated, we use the parameters in Table 4.3 for all experiments.

Default Parameters - Classification

Category	Name	Value
Complexity	Number of *DTs*	50
	Threshold minCount	1
	Threshold minChild	1
Randomness	Bagging Ratio	1.0
	Bagging Replacement	no
	Subsampling at Node Level	25
	Feature Subsampling	$F/2$
Splits	Split Function	Unary + Binary
	Split Compactness Measure	Shannon Entropy
Predictions	Leaf Prediction Model	constant
ADF	ADF Loss	Tangent Loss [90]
GR	Refinement Classification	Crammer and Singer [14]
GP	Pruning Iterations	300
	Pruning Ratio	0.1
	Pruning Strategy	Least Significance

Table 4.3: Default parameters for classification. The role of the different parameters is explained and evaluated in the individual subsections. Unless otherwise stated, these parameters are used for all classification experiments.

4.1.1.1 Overall Results

In this experiment we compare the performance of different methods on all evaluated data sets. We use the same parameters (see Table 4.3) for each method to obtain an unbiased comparison. However, different methods perform best with different parameters. The most

important parameter concerning accuracy on previously unseen data is the maximum tree depth D_{max} (see Section 2.2.2.2). Therefore, we evaluate multiple D_{max} for each method. In this experiment we vary D_{max} in the range [0,20] and select the best performing D_{max}. We report the *MR* and the corresponding D_{max} for each method. Table 4.4 presents experimental results for different methods on all evaluated data sets.

Overall Results - Classification

			Methods				
Data Set	Property	Scale	RF	RF+GR	RF+GP	ADF	ADF+GR
char74k	*MR*	$\cdot 10^{-1}$	1.63 ± 0.01	1.60 ± 0.01	1.58 ± 0.01	1.62 ± 0.01	1.59 ± 0.01
	D_{max}		20	20	20	20	20
mnist	*MR*	$\cdot 10^{-2}$	3.37 ± 0.05	2.97 ± 0.02	2.88 ± 0.03	3.18 ± 0.07	2.74 ± 0.05
	D_{max}		20	16	20	20	16
letter	*MR*	$\cdot 10^{-2}$	3.01 ± 0.30	2.75 ± 0.11	2.65 ± 0.14	2.85 ± 0.10	2.84 ± 0.12
	D_{max}		18	14	18	20	14
pendigits	*MR*	$\cdot 10^{-2}$	2.91 ± 0.14	2.50 ± 0.08	2.50 ± 0.06	2.67 ± 0.06	2.52 ± 0.13
	D_{max}		14	10	16	12	12
protein	*MR*	$\cdot 10^{-1}$	4.11 ± 0.01	4.03 ± 0.01	3.94 ± 0.01	4.13 ± 0.02	3.96 ± 0.02
	D_{max}		20	16	20	20	18
usps	*MR*	$\cdot 10^{-2}$	5.73 ± 0.26	5.66 ± 0.08	5.43 ± 0.09	5.58 ± 0.14	5.63 ± 0.09
	D_{max}		20	8	14	20	8

Table 4.4: Overall results for classification. For each data set and each *SR* factor, the three best performing methods are highlighted in shades of green.

On most data sets *RF+GP* shows the best performance. *RF+GR* and *ADF+GR* achieve a similar level of performance compared to *RF+GP*. We observe that refinement approaches outperform non-refinement approaches on almost all data sets for the best performing D_{max}.

Interestingly, refinement approaches perform best at lower D_{max}, while non-refinement approaches perform best at higher D_{max}. *RFs* with lower D_{max} correspond to simpler models. Simpler models have multiple advantages including reduced model size, faster training and faster testing.

Another interesting characteristic are the relearned leaf prediction models of refinement approaches. For non-refinement approaches, each leaf nodes stores a class probability distribution vector. The entries of this vector represent the probability of each class and the sum of all class probabilities equals 1. In contrast to that, for refinement approaches, each leaf node stores a vector of SVM decision values [11]. In this case, the prediction of a single *DT* is more difficult to interpret compared to a class probability distribution vector. As for non-refined *RFs*, the final prediction is made jointly by all *DTs*. However, the final prediction does not represent a normalized probability either.

4.1.2 Regression

The evaluated methods for regression are summarized in Table 4.5. We evaluate two non-refinement approaches (*RF* and Alternating Regression Forest (ARF)) and six refinement approaches. We employ *GR* and Additive Global Refinement (AGR) for the leaf prediction model optimization. *IRRF* is a special refinement approach as growing the structure of *DTs* and refinement are interweaved.

Methods - Regression

	Training	
Name	Structure	Refinement
RF	RF	-
RF+GR	RF	GR
RF+AGR	RF	AGR
RF+GP	RF	GP
ARF	ARF	-
ARF+GR	ARF	GR
ARF+AGR	ARF	AGR
IRRF	ARF/AGR interweaved	

Table 4.5: Evaluated methods for regression.

We do not present results for *ARF+GP*, since *GP* has a number of disadvantages which we discuss in Section 4.1.2.8. However, we evaluate *RF+GP*, since this method shows state of the art performance for different machine learning tasks in terms of *RF* approaches [77]. In most experiments our proposed methods achieve competitive performance compared to *RF+GP*.

We evaluate the different methods on 22 data sets for regression. The properties of the data sets are summarized in Table 4.6. All 22 data sets have an output dimensionality of one which means the output is a continuous valued scalar.

We evaluate the performance by computing the Root Mean Squared Error (RMSE) between the ground truth and the prediction [10, 77, 89]. The lower the *RMSE*, the better the performance. The magnitude of the *RMSE* is different for each data set. Therefore, we report the corresponding *RMSE* scale factor for each data set.

Unless otherwise stated, we use the parameters in Table 4.7 for all experiments. We use the combined variance of the training data ground truth and features as a compactness measure (see Section 2.2.2.1). However, we also present results for other compactness measures.

In addition to constant prediction models, we evaluate linear prediction models for regression (see Section 3.4). In this case, we perform locally linear regression at each leaf node using the arriving training data. We use Tikhonov regularization [92] to compute the weights for linear prediction models and employ a constant intercept term [67].

Data Sets - Regression

Data Set	Train	Test	Features
abalone	239	159	8
ailerons	7 154	6 596	40
auto_mpg	235	157	7
auto_price	95	64	15
breast_cancer	116	78	32
california_housing	12 384	8 256	8
cart_delve	24 461	16 307	10
cpu_act	4 915	3 277	21
cpu_small	4 915	3 277	12
delta_ailerons	4 277	2 852	5
delta_elevators	5 710	3 807	6
diabetes	26	17	2
elevators	8 752	7 847	18
friedman_delve	24 461	16 307	10
housing	304	202	13
kinematics	4 915	3 277	8
machine	125	84	6
pol	5 000	10 000	48
pyrimidines	44	30	27
servo	100	67	4
stock_airplanes	570	380	9
triazines	112	74	60

Table 4.6: Evaluated data sets for regression.

For regression, we apply min-max feature scaling to each data set [41]. The range of each feature dimension is rescaled to $[0, 1]$. This prevents features with large numeric ranges from dominating other features if the variance of the input space is included in the evaluation of split parameter sets [85]. The preprocessing also improves the numeric stability of linear prediction models.

4.1.2.1 Overall Results

In this experiment we compare the performance of different methods on all evaluated data sets. We use the same parameters (see Table 4.7) for each method to obtain an unbiased comparison. However, different methods perform best with different parameters. The most important parameter concerning accuracy on previously unseen data is the maximum tree depth D_{max} (see Section 2.2.2.2). Therefore, we evaluate multiple D_{max} for each method. In this experiment we vary D_{max} in the range [0,20] and select the best performing D_{max}. We report the *RMSE* and the corresponding D_{max} for each method. Table 4.8 and 4.9 present experimental results for different methods on all evaluated data sets.

On most data sets non-refinement approaches (*RF* or *ARF*) show the best performance. On some data sets refinement approaches (*RF+GP*, *ARF+AGR* or *IRRF*)

Default Parameters - Regression

Category	Name	Value
Complexity	Number of *DTs*	50
	Threshold minCount	1
	Threshold minChild	1
Randomness	Bagging Ratio	1.0
	Bagging Replacement	no
	Subsampling at Node Level	25
	Feature Subsampling	$F/2$
Splits	Split Function	Unary + Binary
	Split Compactness Measure	Variance (GT + F)
Predictions	Leaf Prediction Model	constant
	Tikhonov Regularization Term	0.05
ARF	ARF Loss	Squared Loss
GR	Refinement Regularization	0.05
	Refinement Loss	L2 Loss
GP	Pruning Iterations	300
	Pruning Ratio	0.1
	Pruning Strategy	Least Significance

Table 4.7: Default parameters for regression. The role of the different parameters is explained and evaluated in the individual subsections. Unless otherwise stated, these parameters are used for all regression experiments.

show the best performance. More importantly, the performance of all evaluated methods is approximately the same for most data set.

The key finding of this experiment is that non-refinement approaches and refinement approaches show a similar level of performance, but refinement approaches achieve this level of performance with significantly lower D_{max}. Across all data sets the average D_{max} leading to best performance is 12 for non-refinement approaches and 5 for refinement approaches. *RFs* with lower D_{max} correspond to simpler models. Simpler models have multiple advantages including reduced model size, faster training and faster testing. In the following subsection we discuss this phenomenon in detail.

4.1.2.2 Maximum Depth

In the following experiments we show the effect of the maximum tree depth D_{max} on different methods. D_{max} directly controls the model complexity. The model size increases exponentially with D_{max} (see Section 4.1.2.11). We present results on the data set *elevators*. This data set has a medium number of training samples (8 752) and a medium number of features (18). Previous works show that *ARFs* significantly improve upon standard *RFs* on this data set [89].

Overall Results - Regression - Part 1

Data Set	Property	Scale	Methods							
			RF	RF+GR	RF+AGR	RF+GP	ARF	ARF+GR	ARF+AGR	IRRF
abalone	*RMSE*	$\cdot 10^{0}$	2.38 ± 0.01	2.51 ± 0.04	2.51 ± 0.05	2.42 ± 0.07	2.38 ± 0.03	2.51 ± 0.08	2.49 ± 0.06	2.47 ± 0.10
	D_{max}		4	2	1	3	4	2	2	2
ailerons	*RMSE*	$\cdot 10^{-4}$	1.74 ± 0.01	1.80 ± 0.01	1.80 ± 0.01	1.80 ± 0.01	1.68 ± 0.01	1.77 ± 0.01	1.76 ± 0.01	1.80 ± 0.01
	D_{max}		20	4	4	4	14	4	4	3
auto_mpg	*RMSE*	$\cdot 10^{0}$	3.14 ± 0.03	3.17 ± 0.05	3.16 ± 0.03	3.14 ± 0.06	3.10 ± 0.03	3.22 ± 0.08	3.19 ± 0.02	3.26 ± 0.04
	D_{max}		7	4	4	6	6	4	4	2
auto_price	*RMSE*	$\cdot 10^{3}$	1.63 ± 0.09	2.01 ± 0.11	1.60 ± 0.08	1.84 ± 0.7	1.71 ± 0.08	1.99 ± 0.12	1.72 ± 0.10	1.78 ± 0.14
	D_{max}		7	3	6	3	6	2	4	3
breast_cancer	*RMSE*	$\cdot 10^{1}$	3.29 ± 0.01	3.53 ± 0.14	3.57 ± 0.03	3.31 ± 0.17	3.28 ± 0.01	3.45 ± 0.18	3.49 ± 0.13	3.54 ± 0.16
	D_{max}		2	1	1	1	2	1	1	1
california_housing	*RMSE*	$\cdot 10^{4}$	5.53 ± 0.01	5.73 ± 0.02	5.65 ± 0.02	5.56 ± 0.02	5.31 ± 0.01	5.66 ± 0.03	5.56 ± 0.02	5.98 ± 0.04
	D_{max}		20	9	9	12	20	7	9	7
cart_delve	*RMSE*	$\cdot 10^{0}$	1.01 ± 0.01	1.00 ± 0.01	0.99 ± 0.01	0.94 ± 0.01	1.01 ± 0.01	1.00 ± 0.01	0.98 ± 0.01	1.00 ± 0.01
	D_{max}		10	4	3	6	9	4	3	3
cpu_act	*RMSE*	$\cdot 10^{0}$	2.54 ± 0.02	2.68 ± 0.01	2.61 ± 0.01	2.65 ± 0.01	2.41 ± 0.01	2.63 ± 0.01	2.49 ± 0.02	2.62 ± 0.03
	D_{max}		20	5	8	8	20	4	8	5
cpu_small	*RMSE*	$\cdot 10^{0}$	2.89 ± 0.01	3.15 ± 0.06	3.03 ± 0.01	3.06 ± 0.05	2.78 ± 0.03	3.05 ± 0.05	2.90 ± 0.03	3.08 ± 0.02
	D_{max}		18	6	8	8	20	4	8	6
delta_ailerons	*RMSE*	$\cdot 10^{-4}$	1.66 ± 0.01	1.64 ± 0.02	1.63 ± 0.01	1.65 ± 0.01	1.64 ± 0.01	1.66 ± 0.01	1.65 ± 0.01	1.65 ± 0.01
	D_{max}		12	3	4	5	10	3	3	3
delta_elevators	*RMSE*	$\cdot 10^{-3}$	1.45 ± 0.01	1.48 ± 0.01	1.47 ± 0.01	1.46 ± 0.01	1.44 ± 0.01	1.47 ± 0.01	1.47 ± 0.01	1.47 ± 0.01
	D_{max}		12	2	3	4	9	2	2	2

Table 4.8: Overall results for regression. For each data set, the three best performing methods are highlighted in shades of green.

Overall Results - Regression - Part 2

Data Set	Property	Scale	Methods							
			RF	RF+GR	RF+AGR	RF+GP	ARF	ARF+GR	ARF+AGR	IRRF
diabetes	*RMSE*	$\cdot 10^{-1}$	6.02 ± 0.19	6.07 ± 0.06	5.86 ± 0.02	5.74 ± 0.37	5.67 ± 0.15	5.95 ± 0.14	5.66 ± 0.17	6.31 ± 0.01
	D_{max}		5	2	2	5	4	2	4	1
elevators	*RMSE*	$\cdot 10^{-3}$	3.15 ± 0.01	2.98 ± 0.04	2.87 ± 0.03	2.77 ± 0.04	2.76 ± 0.01	2.95 ± 0.04	2.73 ± 0.02	2.96 ± 0.09
	D_{max}		18	6	6	12	20	6	12	6
friedman_delve	*RMSE*	$\cdot 10^{0}$	1.50 ± 0.01	1.35 ± 0.06	1.30 ± 0.03	1.32 ± 0.02	1.17 ± 0.01	1.23 ± 0.01	1.22 ± 0.01	1.30 ± 0.01
	D_{max}		20	6	7	8	20	6	7	5
housing	*RMSE*	$\cdot 10^{0}$	3.69 ± 0.03	4.09 ± 0.21	3.68 ± 0.21	4.04 ± 0.15	3.72 ± 0.04	3.98 ± 0.13	3.52 ± 0.19	3.59 ± 0.15
	D_{max}		11	4	7	4	11	4	6	6
kinematics	*RMSE*	$\cdot 10^{-1}$	1.26 ± 0.01	1.22 ± 0.02	1.16 ± 0.02	1.21 ± 0.02	1.10 ± 0.01	1.21 ± 0.03	1.11 ± 0.01	1.38 ± 0.04
	D_{max}		18	8	9	9	19	8	10	6
machine	*RMSE*	$\cdot 10^{1}$	3.53 ± 0.04	3.91 ± 0.16	3.71 ± 0.08	3.44 ± 0.13	3.94 ± 0.25	3.89 ± 0.25	4.05 ± 0.24	4.66 ± 0.32
	D_{max}		11	6	6	7	8	7	4	6
pol	*RMSE*	$\cdot 10^{0}$	6.24 ± 0.07	6.04 ± 0.05	5.94 ± 0.07	5.86 ± 0.11	6.41 ± 0.05	6.27 ± 0.13	6.17 ± 0.08	9.06 ± 0.47
	D_{max}		20	10	10	14	20	9	12	8
pyrimidines	*RMSE*	$\cdot 10^{-2}$	5.63 ± 0.31	6.02 ± 0.14	5.47 ± 0.17	5.78 ± 0.31	5.05 ± 0.08	6.33 ± 0.30	5.35 ± 0.13	5.12 ± 0.08
	D_{max}		10	2	5	2	10	2	5	3
servo	*RMSE*	$\cdot 10^{-1}$	4.50 ± 0.17	4.35 ± 0.42	3.43 ± 0.35	3.54 ± 0.23	4.16 ± 0.20	4.37 ± 0.29	3.29 ± 0.06	3.16 ± 0.19
	D_{max}		7	4	4	5	10	4	5	6
stock_airplane	*RMSE*	$\cdot 10^{0}$	0.69 ± 0.01	1.18 ± 0.01	0.70 ± 0.02	1.15 ± 0.02	0.73 ± 0.02	1.18 ± 0.03	0.72 ± 0.01	0.90 ± 0.02
	D_{max}		18	3	10	4	14	3	8	9
triazines	*RMSE*	$\cdot 10^{-1}$	1.25 ± 0.04	1.40 ± 0.06	1.32 ± 0.05	1.23 ± 0.04	1.24 ± 0.05	1.36 ± 0.08	1.32 ± 0.04	1.44 ± 0.11
	D_{max}		6	3	3	8	6	2	2	2

Table 4.9: Overall results for regression. For each data set, the three best performing methods are highlighted in shades of green.

Constant Prediction Models

Figure 4.1 presents results for different D_{max} using constant prediction models. In this experiment we vary D_{max} in the range [0,20].

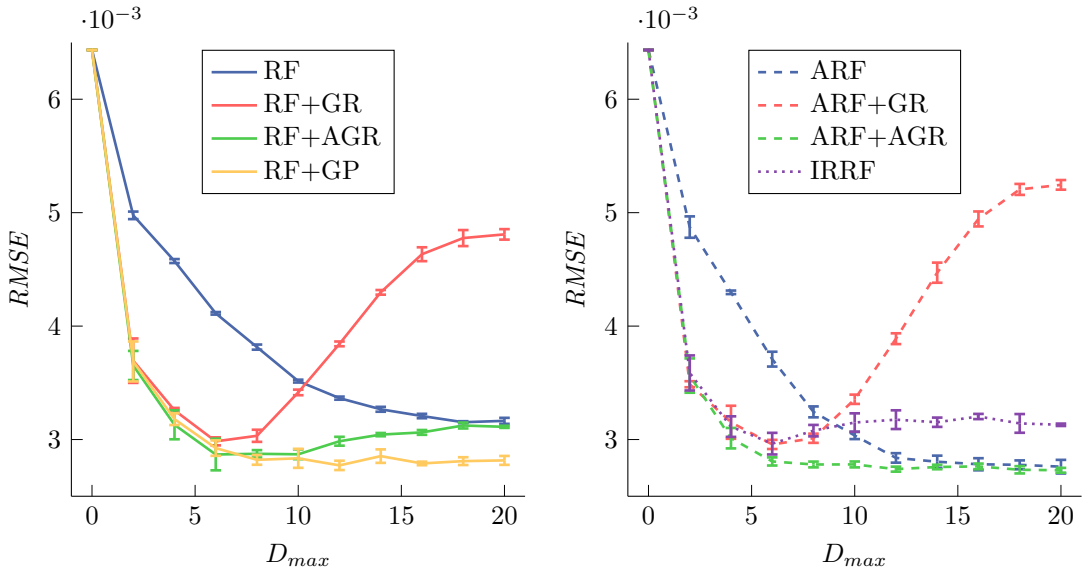


Figure 4.1: Evaluation of the maximum tree depth for D_{max} for constant prediction models.

For low depths ($D_{max} \leq 4$), refinement approaches significantly outperform non-refinement approaches. All refinement approaches perform similar for low depths. For low depths, the performance of refinement techniques is independent of the underlying DT growing scheme (RF or ARF). $IRRF$ is on par with single refinement approaches for low depths.

For high depths ($D_{max} \geq 10$), non-refinement approaches are competitive to refinement approaches. GR shows severe overfitting for high depths. AGR shows reduced overfitting compared to GR for high depths. However, AGR cannot improve upon the underlying DT growing scheme (RF or ARF) for high depths. $IRRF$ shows overfitting for high depths. GP does not show overfitting.

Figure 4.1 shows that non-refinement approaches perform best with deep DTs. Deep DTs correspond to complex models with large model sizes. In contrast to that, refinement approaches perform best with shallow DTs. Refinement approaches cannot improve the performance as D_{max} increases. Instead, refinement techniques like GR show severe overfitting for high depths. AGR and GP reduce overfitting, but fail to improve the performance for high depths.

Linear Prediction Models

Figure 4.2 presents results for different D_{max} using linear prediction models. In this experiment we vary D_{max} in the range [0,10].

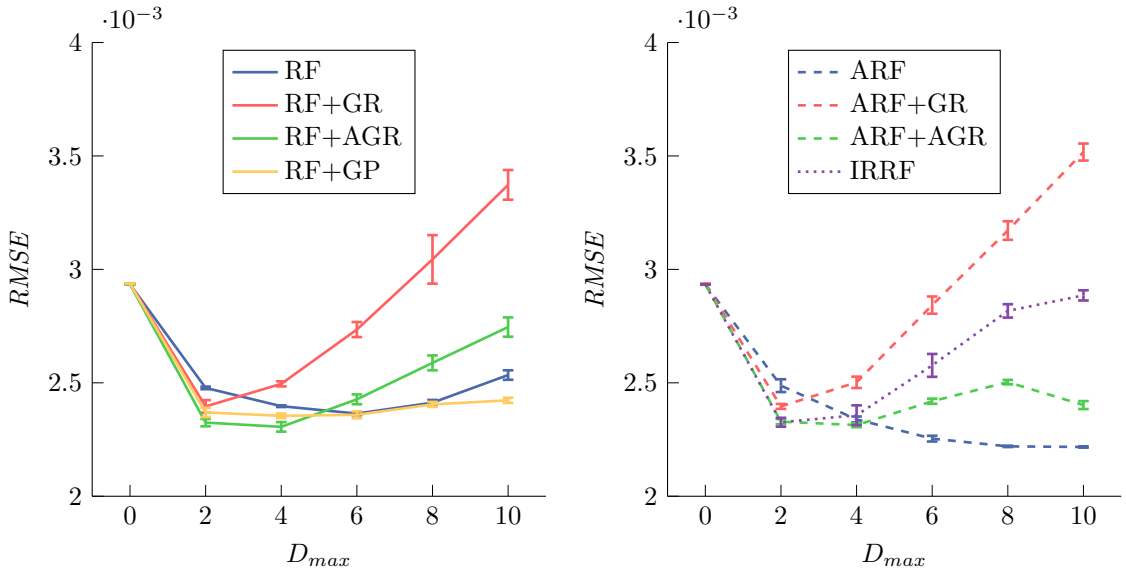


Figure 4.2: Evaluation of the maximum tree depth for D_{max} for linear prediction models.

We observe the same behavior as for constant prediction models. Refinement approaches outperform non-refinement approaches for low D_{max}, but show overfitting for high D_{max}. Due to the increased fitting power of linear prediction models the behavior is intensified. Overfitting already occurs at medium depths ($D_{max} \geq 4$). For high depths ($D_{max} \geq 8$), even non-refinement approaches show overfitting. Only *ARF* is robust to overfitting.

ARF+AGR shows a special behavior. Overfitting starts at medium depths ($D_{max} \geq 4$), but decreases again for high depths ($D_{max} \geq 10$). We hypothesize that one reason for this is that *ARF+AGR* performs a refinement which targets residuals obtained by evaluating an *ARF*. These residuals have low norm as deep *ARFs* almost perfectly fit the training data. The *RMSE* of *ARFs* on the test data set saturates for high D_{max}, but the *RMSE* on the training data set continuously decreases. *AGR* overfits the residuals, but the contribution of the refinement to the existing prediction models is negligible, due to the low norm. The higher D_{max}, the lower the norm. *ARF+AGR* benefits from the strong fitting power and robustness to overfitting of deep *ARFs*. The described behavior is also observable in Figure 4.3 and Figure 4.4.

Constant versus Linear Prediction Models

Next, we compare the performance of constant and linear prediction models. Figure 4.3 presents results for one non-refinement approach (*RF*) and one refinement approach (*ARF+AGR*) on the data set *elevators*. In this experiment we vary D_{max} in the range [0,20].

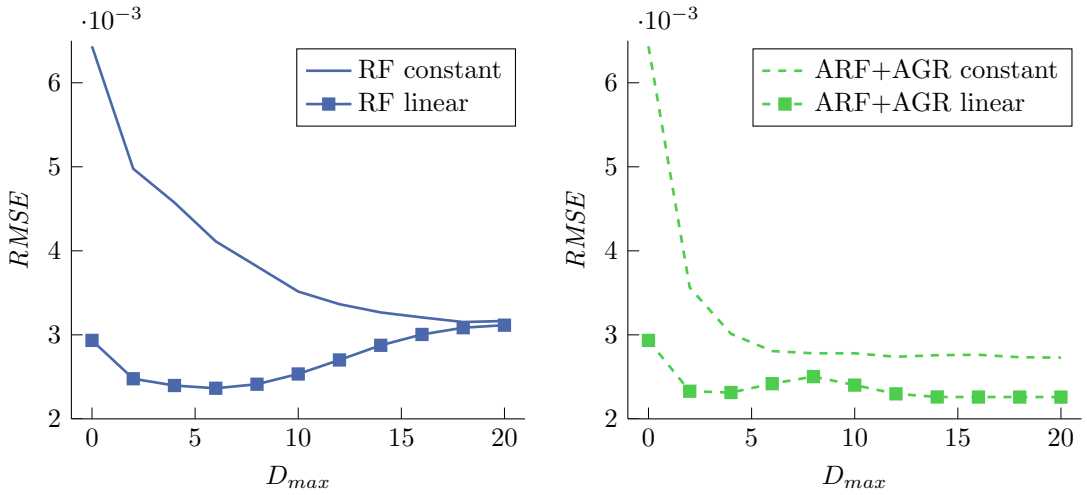


Figure 4.3: Comparison of constant and linear prediction models on the data set *elevators*. The observed data set has a feature dimensionality of 18.

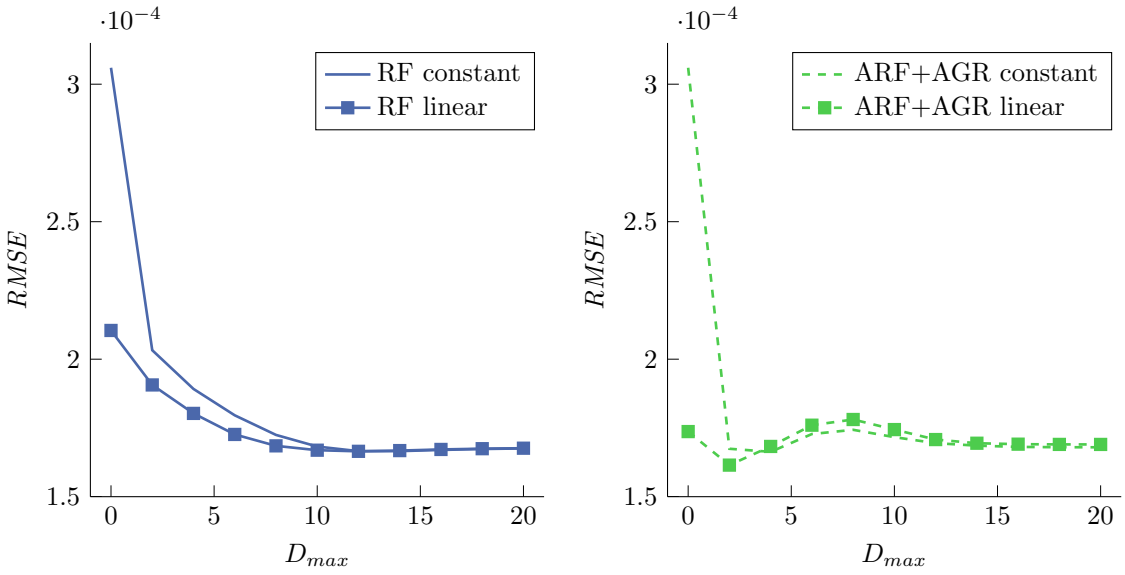


Figure 4.4: Comparison of constant and linear prediction models on the data set *delta_ailerons*. The observed data set has a feature dimensionality of 5.

Linear prediction models outperform constant prediction models in this experiment. In fact, simple linear regression at $D_{max} = 0$ performs better than *RF* with constant prediction models at $D_{max} = 20$. Linear prediction models benefit from a high number of features. The observed data set *elevators* has a feature dimensionality of 18.

The improvement achieved by linear prediction models is different for other data sets. The performance gap vanishes if the feature dimensionality is low. Figure 4.4 presents results on the data set *delta_ailerons*. This data set has a feature dimensionality of 5. The performance of constant prediction models is on par with linear prediction models for $D_{max} \geq 2$ on this data set.

4.1.2.3 Early Stopping

In this experiment we show the effect of early stopping apart from explicitly restricting D_{max} on different methods. During the training of *DTs* the greedy partitioning of the training data is stopped if: i) the number of training samples arriving at a node is below a predefined threshold (*minCount*); ii) the size of one of the two subsets generated by a split is below a predefined threshold (*minChild*).

These techniques indirectly control the model complexity. High values for *minCount* and *minChild* result in reduced model size. In contrast, restricting D_{max} gives more precise control over the model complexity as the resulting *DTs* have $2^{D_{max}+1} - 1$ nodes at most.

We present results on the data set *stock_airplane*. This data set has a medium number of training samples (570) and a low number of features (9). Figure 4.5 presents results for different *minCount* and *minChild* thresholds. We do not restrict D_{max} in this experiment.

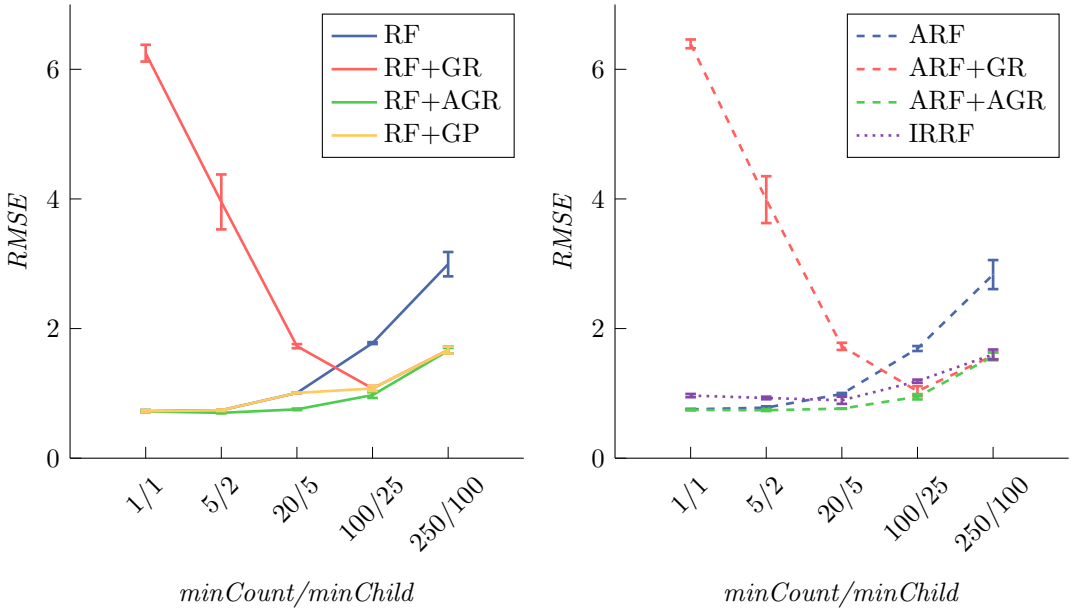


Figure 4.5: Evaluation of Early Stopping.

Non-refinement approaches perform best with fully grown *DTs* (*minCount*=1 and *minChild*=1) which correspond to complex models with large model sizes. The performance of non-refinement approaches decreases as both thresholds increase.

Refinement approaches perform significantly better than non-refinement approaches for high thresholds which correspond to less complex models with smaller model sizes. Refinement approaches show only slightly improved performance as the model complexity increases. For simple models, all refinement approaches perform equally.

For complex models, non-refinement approach and refinement approaches which reduce overfitting perform equally. *GR* shows severe overfitting for fully grown *DTs*.

4.1.2.4 Number of Decision Trees

In this experiment we show the effect of the number of *DTs* of an ensemble on different methods. Similar to D_{max} (see Section 4.1.2.2), the number of *DTs* directly controls the model complexity. We point out that the model size increases exponentially with D_{max}, but only linearly with the number of *DTs* (see Section 4.1.2.11).

We present results on the data set *cpu_small*. This data set has a medium number of training samples (4 915) and a medium number of features (12). Previous works show that *ARFs* and *RF+GP* significantly improve upon standard *RFs* on this data set [77, 89]. Figure 4.6 presents results for different numbers of *DTs*. In this experiment we vary the number of *DTs* in the range [1,500]. We use constant prediction models and set D_{max} to 5. At this depth refinement techniques do not overfit. The selected D_{max} is specific to the data set *cpu_small*.

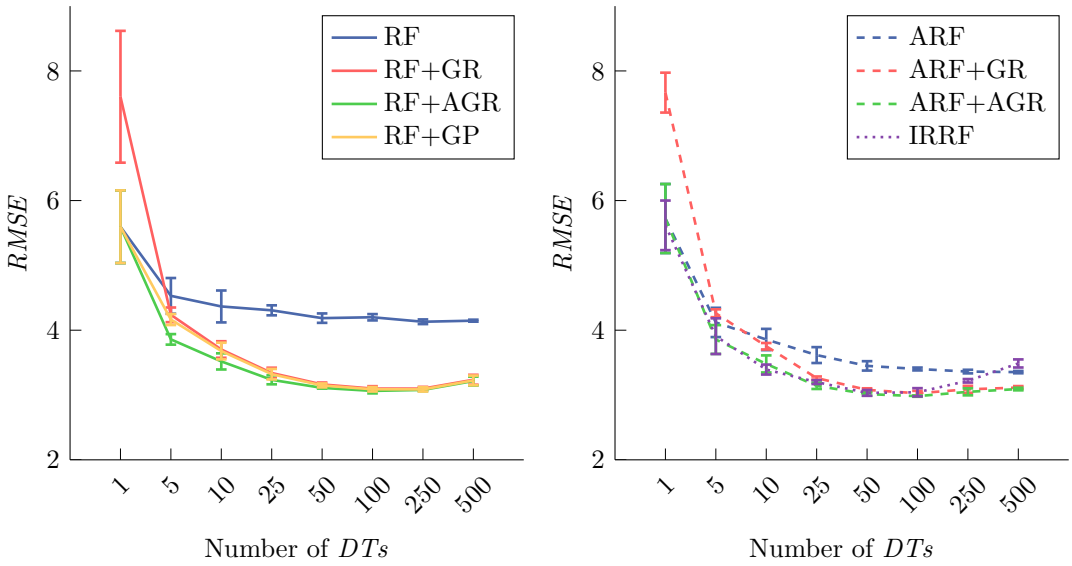


Figure 4.6: Evaluation of the number of *DTs*.

All methods show low performance when the number of *DTs* is small. As the number of *DTs* rises, the performance increases, but the effect saturates at some point. The experiment confirms the proposition of Breiman [8] which states that *RFs* do not overfit when more *DTs* are added to the ensemble, as long as the correlation between the individual *DTs* is low.

In contrast to that, refinement approaches show small overfitting, when the number of *DTs* is too high. We hypothesize that one reason for this is that the support vector regression performed by LIBLINEAR is more prone to overfitting if the input feature dimensionality of the refinement problem is high, due to the curse of dimensionality [25]. The higher the number of *DTs*, the higher the input feature dimensionality of the refinement problem.

Figure 4.6 shows that the standard deviation decreases as the number of *DTs* rises. This experiment confirms that ensembles of *DTs* reduce variance compared to a single *DT* (see Section 2.3).

4.1.2.5 Randomness

In the following experiments we show the effect of randomness on different methods. We investigate multiple parameters, which control the amount of randomness injected into the training. We use multiple techniques for injecting randomness into the training. We employ Bagging, feature subsampling and subsampling at node level (see Section 2.3).

We present results on the data set *ailerons* (see Table 4.6). This data set has a medium number of training samples (7 154) and a high number of features (40). Previous works show that *ARFs* and *RF+GP* significantly improve upon standard *RFs* on this data set [77, 89]. In the following experiments we use constant prediction models and set D_{max} to 4. At this depth refinement techniques do not overfit. The selected D_{max} is specific to the data set *ailerons*.

Bagging

Each *DT* is trained on a different data set which is generated by Bagging [7]. We report results for Bagging with and without replacement and different Bagging ratios. The Bagging ratio denotes the size of the bagged data set relative to the original data set. A low Bagging ratio corresponds to more decorrelated *DTs*. Additionally, a low Bagging ratio decreases the training time.

Figure 4.7 and 4.8 present results for different Bagging ratios and replacement strategies. In this experiment we vary the Bagging ratio in the range [0.25,1].

All evaluated methods are insensitive to the selected Bagging ratio. Additionally, there is no performance difference between Bagging with or without replacement. One reason for this is that we employ subsampling at node level which decreases the influence of Bagging on the performance. Subsampling at node level is analyzed later in this subsection.

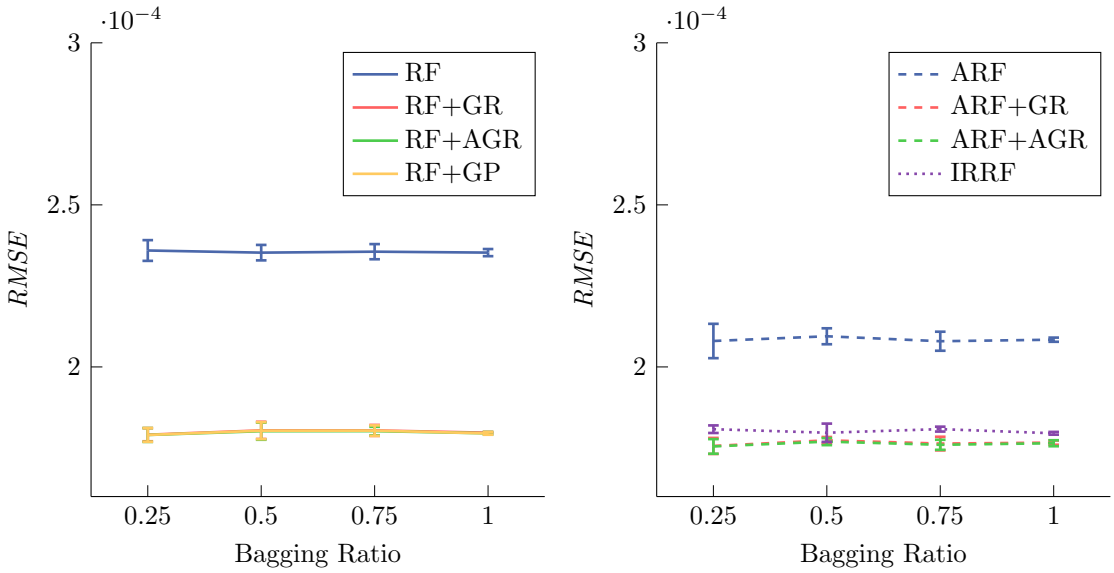


Figure 4.7: Evaluation of Bagging without replacement.

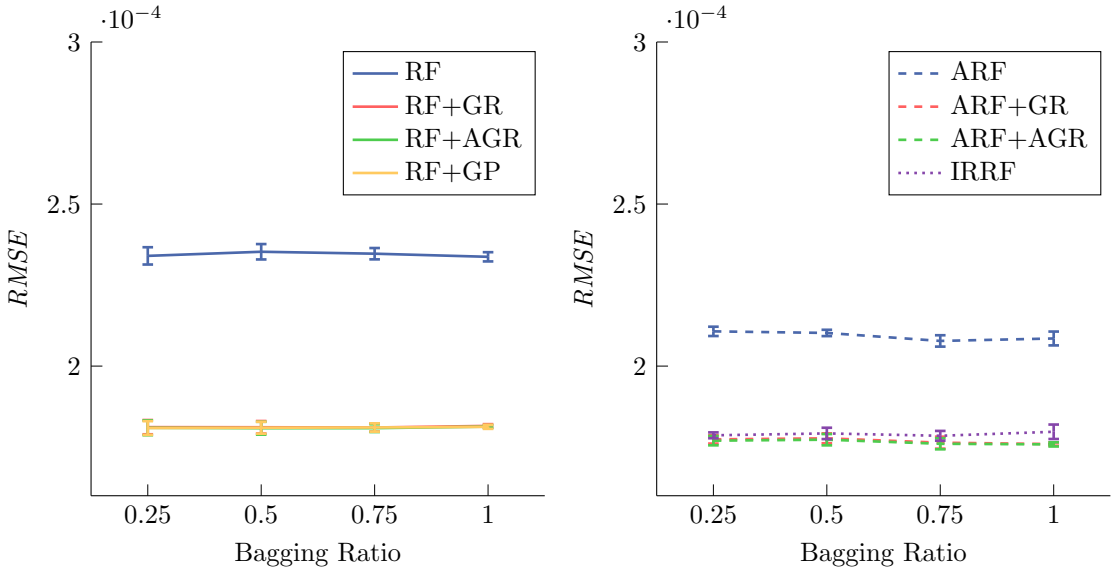


Figure 4.8: Evaluation of Bagging with replacement.

Due to the low D_{max} of 4, we do not obtain fully grown *DTs* for low Bagging ratios. For a Bagging ratio of 0.25, the mean number of samples arriving at a leaf node is 100. As the performance of all methods is not affected by Bagging in our configuration, we can employ low Bagging ratios to speed up the training as long as the number of *DTs* is sufficiently high.

Feature Subsampling

At each split node the feature dimensionality of the arriving training data is subsampled. Therefore, the training of the split ignores certain features. A low number of subsampled features corresponds to more decorrelated *DTs*. Additionally, a low number of subsampled features decreases the training time, because the number of evaluated split parameter sets is dependent of the number of analyzed features in our implementation.

Figure 4.9 presents results for different feature subsampling values. In this experiment we vary the number of features used to train a split in the range [1,40]. The feature dimensionality of the observed data set *ailerons* is 40.

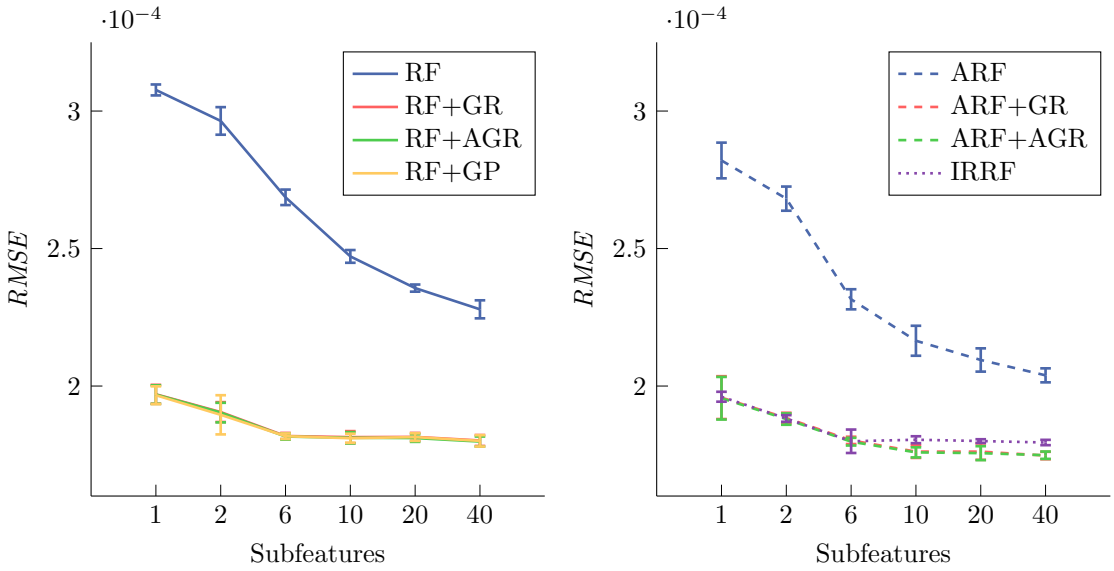


Figure 4.9: Evaluation of feature subsampling.

Non-refinement approaches like *RFs* and *ARFs* benefit from a high number of analyzed features. Refinement approaches are less sensitive to this parameter, but perform better for a high number of analyzed features.

A high number of features corresponds to a less randomized selection of the thresholded feature dimension. Analyzing a high number of features increases the training time of splits and may increase the correlation of the individual *DTs*.

Subsampling at Node Level

At each split node the number of arriving training samples is subsampled if it is above a predefined threshold. The subsampled training data is used to optimize the split performed by the node. A low number of subsamples corresponds to more decorrelated *DTs*. Additionally, a low number of subsamples decreases the training time.

Figure 4.10 presents results for different subsampling thresholds at node level. In this experiment we vary the number of subsamples used to train a split in the range [5,10 000]. The number of training samples of the observed data set *ailerons* is 7 154.

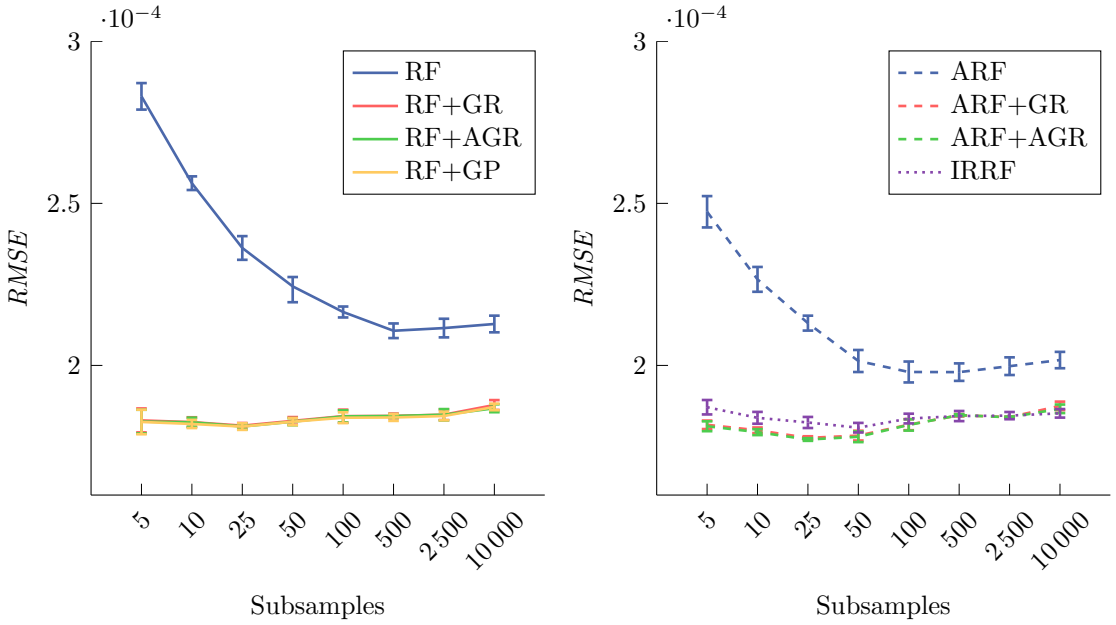


Figure 4.10: Evaluation of subsampling at node level.

Non-refinement approaches like *RFs* and *ARFs* benefit from a high number of subsamples. This means that the threshold selection for each split is more precise. Refinement approaches are insensitive to this parameter.

If the subsampling threshold is too high, the performance of all methods slightly decreases. One reason for this is that the individual *DTs* are more correlated if a high number of subsamples is analyzed at each node. A low number of subsamples corresponds to a more randomized selection of the split threshold.

We observe that the accuracy of splits increases as the number of arriving training samples decreases. As a result, the splits in early depth levels are more randomized.

Interaction of Feature Subsampling and Subsampling at Node Level

To conclude the evaluation of randomness, we present results on the interaction of feature subsampling and subsampling at node level. Both parameters influence the training of splits. We present results for one non-refinement approach (*RF*) and one refinement approach (*ARF+AGR*).

Table 4.10 and 4.11 present results for different feature subsampling and subsampling at node level thresholds. In this experiment we vary the number of subsampled features in the range [2,40] and the number of subsamples in the range [10,10 000].

Subfeatures \ Subsamples	10	25	100	500	2500	10000
2	2.99 ± 0.02	2.94 ± 0.03	2.92 ± 0.03	2.93 ± 0.04	2.95 ± 0.04	2.95 ± 0.03
6	2.79 ± 0.06	2.66 ± 0.05	2.54 ± 0.05	2.55 ± 0.04	2.56 ± 0.05	2.57 ± 0.03
10	2.70 ± 0.05	2.51 ± 0.05	2.35 ± 0.02	2.32 ± 0.03	2.32 ± 0.06	2.34 ± 0.05
20	2.55 ± 0.03	2.36 ± 0.03	2.16 ± 0.02	2.10 ± 0.01	2.11 ± 0.01	2.12 ± 0.01
40	2.52 ± 0.04	2.25 ± 0.02	2.06 ± 0.01	2.03 ± 0.01	2.05 ± 0.01	2.08 ± 0.01

Table 4.10: Interaction of feature subsampling and subsampling at node level for *RF*. The reported results show the *RMSE* on the test data set with scale $\cdot 10^{-4}$.

Subfeatures \ Subsamples	10	25	100	500	2500	10000
2	1.89 ± 0.04	1.87 ± 0.03	1.93 ± 0.05	1.95 ± 0.03	1.99 ± 0.03	2.01 ± 0.04
6	1.81 ± 0.03	1.78 ± 0.02	1.81 ± 0.02	1.85 ± 0.02	1.85 ± 0.02	1.86 ± 0.01
10	1.79 ± 0.02	1.78 ± 0.02	1.78 ± 0.02	1.82 ± 0.02	1.83 ± 0.01	1.84 ± 0.01
20	1.77 ± 0.02	1.76 ± 0.02	1.77 ± 0.02	1.81 ± 0.01	1.82 ± 0.01	1.83 ± 0.02
40	1.77 ± 0.01	1.76 ± 0.01	1.76 ± 0.02	1.82 ± 0.01	1.85 ± 0.01	1.88 ± 0.01

Table 4.11: Interaction of feature subsampling and subsampling at node level for *ARF+AGR*. The reported results show the *RMSE* on the test data set with scale $\cdot 10^{-4}$.

RF performs best, when the number of features and the number of subsamples is high. *ARF+AGR* performs best, when the number of features is high, but the number of subsamples is low. Both non-refinement and refinement approaches benefit from a high number of analyzed feature dimensions.

In contrast to refinement approaches, non-refinement approaches benefit from a high subsampling threshold at node level. We hypothesize that one reason for this is the low D_{max} of 4 in this experiment. The average number of samples arriving at each leaf is 400. Therefore, non-refinement approaches prefer *DTs* with accurate splits to have samples with as little as possible variance at each leaf.

For refinement approaches, the leaf model optimization focuses on specific leaves from few *DTs*, while the contribution of leaves from other *DTs* is low. Therefore, refinement approaches are more robust to outliers in leaf nodes.

The lowest randomness is injected if all feature dimension (40) and all arriving subsamples (10 000) are analyzed. In this case, the individual *DTs* are almost identical. However, there is little standard deviation in the results, because we additionally observe a number of randomly selected binary features at each split node.

For standard *RFs*, which employ Bagging and feature subsampling, Breiman [8] recommends to set the number of analyzed feature dimensions to $\sqrt{F}$, where F is the feature dimensionality. Our experiments show that it is advisable to analyze more feature dimension when subsampling at node level is additionally employed [33, 48]. We propose to evaluate $F/2$ feature dimensions at each split node for this configuration.

4.1.2.6 Split Functions

In this experiment we show the effect of different split function compactness measures. We evaluate three compactness measures: i) the combined variance of the training data ground truth and features; ii) the variance of the training data ground truth; iii) random splits for which no compactness measure is computed [36].

elevators

		Compactness Measures		
Method	Property	Variance (GT+F)	Variance (GT)	Random
RF	*RMSE*	3.15 ± 0.01	3.15 ± 0.02	5.03 ± 0.12
	D_{max}	18	20	20
RF+GR	*RMSE*	2.98 ± 0.04	2.97 ± 0.03	3.59 ± 0.08
	D_{max}	6	6	8
RF+AGR	*RMSE*	2.87 ± 0.03	2.89 ± 0.04	3.50 ± 0.09
	D_{max}	6	6	8
RF+GP	*RMSE*	2.77 ± 0.04	2.76 ± 0.04	3.55 ± 0.05
	D_{max}	12	11	13
ARF	*RMSE*	2.76 ± 0.01	2.73 ± 0.04	4.28 ± 0.07
	D_{max}	20	20	20
ARF+GR	*RMSE*	2.95 ± 0.04	2.95 ± 0.05	3.58 ± 0.08
	D_{max}	6	6	7
ARF+AGR	*RMSE*	2.73 ± 0.02	2.78 ± 0.06	3.55 ± 0.11
	D_{max}	12	10	10
IRRF	*RMSE*	2.96 ± 0.09	2.99 ± 0.10	3.66 ± 0.07
	D_{max}	6	6	6

Table 4.12: Evaluation of compactness measures. The reported results show the *RMSE* on the test data set with scale $\cdot 10^{-3}$ and used D_{max}

We present results on the data set *elevators*. Table 4.12 presents results for different compactness measures. In this experiment we vary D_{max} in the range [0,20] and select the best performing D_{max}. We report the *RMSE* and the corresponding D_{max} for each method.

RFs with totally randomized splits perform significantly worse than *RFs* with splits optimized based on variance compactness measures. The best performing D_{max} is similar for all compactness measures.

There is no performance difference between variance compactness measures in this experiment. However, we employ the variance of the training data ground truth and features, since linear prediction models benefit from less feature variance between arriving training samples.

4.1.2.7 Global Refinement

In the following experiments we analyze various characteristics of *GR*. We evaluate the effect of regularization and benchmark different convex loss functions. We present results on the data set *kinematics*. This data set has a medium number of training samples (4 915) and a low number of features (8).

Regularization

In this experiment we show the effect of regularization on *GR*. In Equation 2.18 the Euclidean norm of the weights is minimized to reduce the risk of overfitting [92]. The regularization is controlled by the parameter C. High values for C correspond to low regularization.

Figure 4.11 presents results for *GR* with different C compared to *RF*. In this experiment we vary C in the range [0.001,1] and D_{max} in the range [0,14].

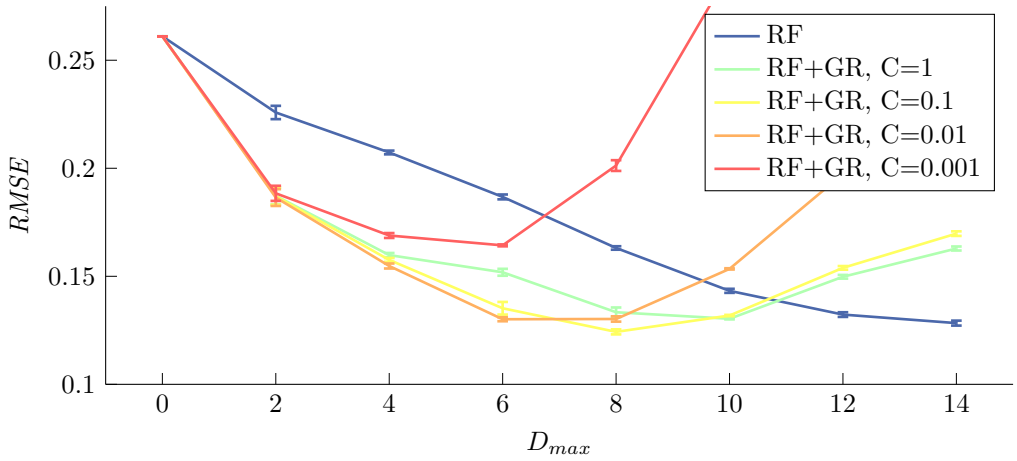


Figure 4.11: Evaluation of regularization for *GR*.

GR with low regularization ($C = 1$) is prone to overfitting for deep *DTs*. For high D_{max} the number of training samples arriving at each leaf node is low and *GR* overfits the training data.

Interestingly, *GR* with high regularization ($C = 0.001$) does not reduce overfitting for deep *DTs*. Instead, *GR* with $C = 0.001$ underfits the training data for $D_{max} \geq 4$. One reason for this is that, the data term in Equation 2.18 is almost ignored and the refinement only focuses on finding weights with low norm.

We conclude that the regularization techniques employed in Equation 2.18 are not sufficient to prevent overfitting or underfitting for deep *RFs*. As a result, non-refined *RFs* outperform *GR* for deep *DTs* independent of C.

Convex Loss Function

In [77] *GR* minimizes a convex L2 regularized optimization problem (see Equation 2.18). However, the convex loss function $\mathcal{L}(\cdot)$ is not specified. LIBLINEAR [26] implements L1 loss and L2 loss support vector regression. L1 loss is more robust to outliers, while L2 loss enforces a low error for all data samples. One reason for this is that there is less penalization for outliers using L1 loss compared to L2 loss.

Figure 4.12 presents results for *GR* with L1 loss and L2 loss compared to *RF*. In this experiment *GR* with L2 loss performs slightly better than *GR* with L1 loss. For data sets with more noise or outliers, *GR* with L1 loss may perform better.

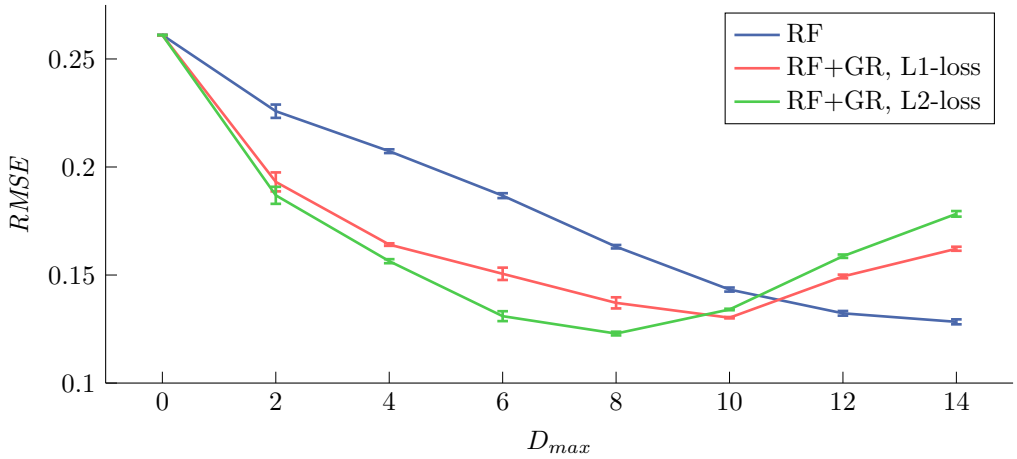


Figure 4.12: Evaluation of loss function for *GR*.

Additionally, there are LIBLINEAR extensions which support parallel L2 loss minimization [59] and incremental learning for L2 loss minimization [96]. These extensions can be used to speed up the training.

4.1.2.8 Global Pruning

In the following experiments we analyze various characteristics of *GP*. We observe the progression of *GP* over multiple iterations, evaluate different pruning parameters and benchmark different pruning strategies. We present results on the data set *elevators*.

GP needs an validation data set. Due to the limited number of available data samples in standard machine learning tasks, we use the test data set for validation. This means that *GP* is indirectly trained on the test data set and the learned predictor is biased. Despite this statistically incorrect learning, our experiments show that the performance of *GP* is not superior to other methods.

Progression of Global Pruning

Figure 4.13 shows the progression of *GP* over multiple iterations compared to *RF*. In this experiment we set D_{max} to 12 and perform 300 pruning iterations. At this depth *GR* alone already shows overfitting (see Figure 4.1).

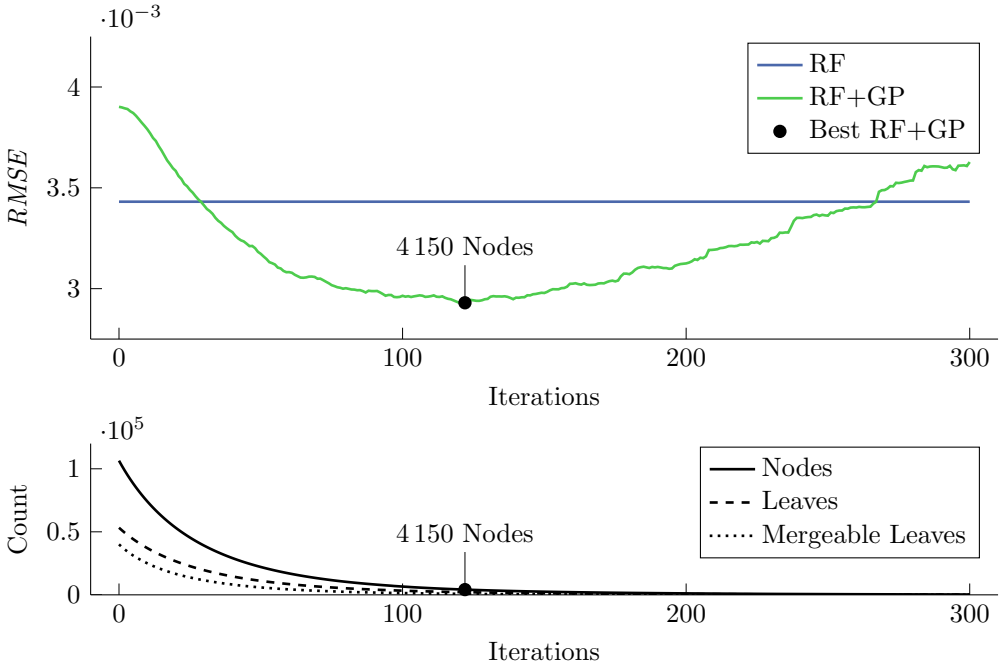


Figure 4.13: Progression of *GP* over multiple iterations.

At iteration 0 a single refinement pass is performed, but no pruning is done yet. At this point the performance of *GP* is equal to *GR* and shows overfitting compared to *RF*. In the beginning iterative *GP* significantly reduces overfitting, until iteration 90. Between iteration 90 and 150 the performance is virtually constant. After 150 iterations the performance starts to decrease and *GP* shows underfitting.

At iteration 0 the *RF* consists of 106 450 nodes. At iteration 300 the *RF* consists of 268 nodes. The best performing *RF* is obtained after 122 iterations and consists of 4 150 nodes, which corresponds to an average of 83 nodes per *DT*. This node count is comparable to a balanced *RF* with $D_{max} = 5$ (63 nodes per *DT*).

Analogous to previous experiments (see Section 4.1.2.2), *GR* performs best with simpler models, while *RF* performs best with complex models. This behavior is reproducible for different D_{max} and different data sets.

Pruning Ratio

In this experiment we show the effect of different pruning ratios. The pruning ratio determines which percentage of all fusible leaves is merged in each iteration. High pruning ratios corresponds to fast reduction in model complexity.

Figure 4.14 presents results for different pruning ratios. In this experiment we vary the pruning ratio in the range [0.05,0.5]. Other than that, we use the same parameters as in the previous experiment.

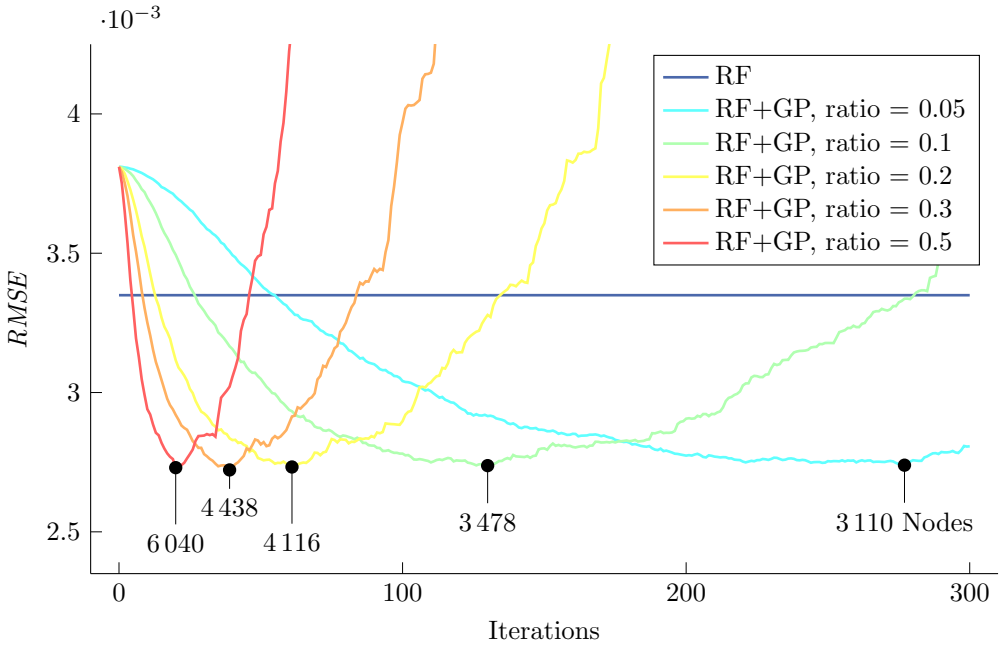


Figure 4.14: Evaluation of pruning ratio for *GP*.

We observe the same behavior across all pruning ratios. In each iteration the model is simplified. In the beginning overfitting is reduced and the performances increases. At some point the model becomes too simple and the performance starts to decrease due to underfitting.

In [77] *GP* with low pruning ratios is recommended to achieve best accuracy. We observe that different pruning ratios yield equally performing *RFs*, but smaller pruning ratios generate simpler models. Due to the smaller model reduction in each iteration, smaller pruning ratios adjust the model complexity more precisely. However, smaller pruning ratios need more iterations which corresponds to slower training.

Pruning Strategy

In this experiment we show the effect of different pruning strategies. In [77] *GP* merges the least significant leaf pairs (see Section 2.5.2). If the Euclidean norm of a prediction model is close to zero, the impact on the final result is negligible. However, this heuristic merging rule is not guaranteed to improve the performance of the model.

In this experiment we evaluate two other pruning strategies: i) merging randomly selected leaf pairs; ii) merging the most significant leaf pairs. Figure 4.15 presents results for different pruning strategies.

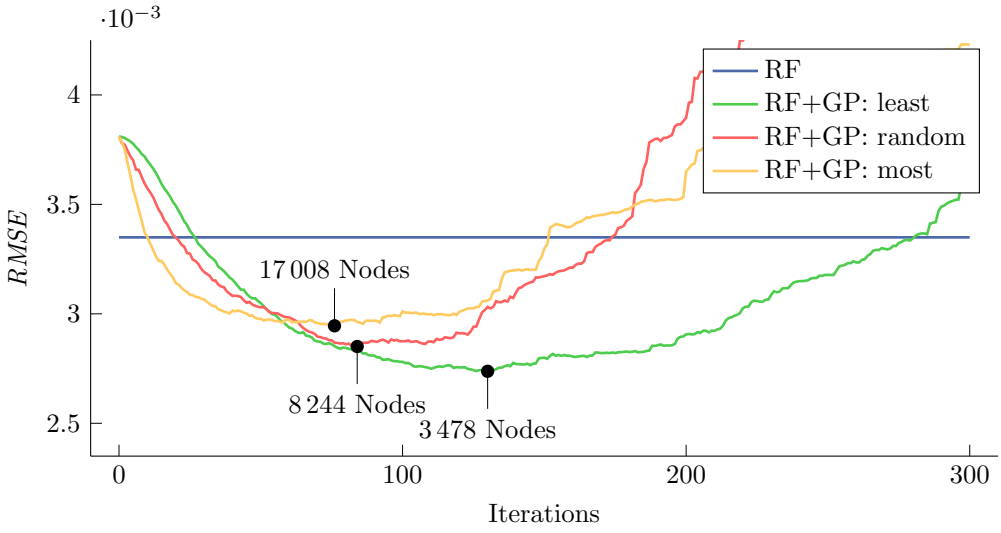


Figure 4.15: Evaluation of pruning strategies for *GP*.

We observe the same characteristic behavior for all strategies. In the beginning overfitting is reduced and the performances increases. At some point the model becomes too simple and the performance starts to decrease due to underfitting.

GP performs significantly better than *RF* for all evaluated pruning strategies . While pruning the least significant leaf pairs performs best, our two alternative pruning strategies show competitive results. In fact, the two alternative strategies outperform the merging rule presented in [77] for highly complex models in the first pruning iterations. One explanation for this phenomenon is that overfitting can be reduced by minimizing the Euclidean norm of the prediction weights [6, 92]. This corresponds to merging the most significant leaf pairs.

Pre-Pruning versus Post-Pruning

The results presented in Section 4.1.2.2 show that *GR* for low D_{max} performs similar to *GP* for high D_{max}. In this experiment we compare pre-pruned shallow *RF+GR* to post-pruned deep *RF+GP*.

First, we train *RF+GR* to a low depth, where we do not observe overfitting. Second, we train *RFs* to a higher depth and apply *GP* until the node count of the *RF* equals the node count of the shallow *RF+GR*. Table 4.13 and 4.14 present results for the data sets *elevators* and *ailerons*.

elevators

Method	D_{max}	Node Count	Training Time	*RMSE*
RF+GR	6	$5\,951 \pm 21$	0.63 ± 0.01	2.99 ± 0.06
RF+GP	8	$5\,859 \pm 45$	8.99 ± 0.15	2.93 ± 0.04
RF+GP	10	$5\,868 \pm 25$	20.18 ± 0.41	2.88 ± 0.03
RF+GP	12	$5\,900 \pm 76$	33.79 ± 1.17	2.89 ± 0.04

Table 4.13: Comparison of pre-pruned *RF+GR* and post-pruned *RF+GP* for the data set *elevators*. The reported results show the *RMSE* on the test data set with scale $\cdot 10^{-3}$.

ailerons

Method	D_{max}	Node Count	Training Time	*RMSE*
RF+GR	5	$3\,120 \pm 14$	0.88 ± 0.02	1.80 ± 0.02
RF+GP	8	$3\,101 \pm 16$	15.63 ± 0.31	1.83 ± 0.02
RF+GP	10	$3\,107 \pm 12$	31.47 ± 0.55	1.84 ± 0.01
RF+GP	12	$3\,100 \pm 19$	52.18 ± 1.37	1.84 ± 0.02

Table 4.14: Comparison of pre-pruned *RF+GR* and post-pruned *RF+GP* for the data set *ailerons*. The reported results show the *RMSE* on the test data set with scale $\cdot 10^{-4}$.

For the data set *elevators*, post-pruned deep *RF+GP* performs slightly better. In contrast to that, for the data set *ailerons*, pre-pruned shallow *RF+GR* performs slightly better. More importantly, *GR* and *GP* show a similar level of performance on both data sets.

For pruned *RFs* of $D_{max} = 12$ we observe that each *DTs* has at least one leaf with $D_{max} \geq 11$. This means that pruning tends to produce unbalanced *DTs*, which have few deep branches. The performance of these unbalanced *DTs* is on par with shallow pre-pruned *DTs* if refinement techniques are employed.

This experiment shows that the performance of refinement approaches is largely insensitive to the structure of the individual *DTs*, but highly dependent on the total number of nodes in a *RF*. Pre-pruned shallow *RF+GR* shows a similar level of performance compared to post-pruned deep *RF+GP*, but the training of pre-pruned

shallow *RF+GR* is more efficient. The higher training time of *RF+GP* is caused by the increased D_{max} and the iterative post-pruning.

In our experiments *GP* shows appealing properties. During all experiments *GP* was among the best performing methods. *GP* is robust to overfitting. *GP* produces simple *DTs* with low model size. However, *GP* also has certain drawbacks. *GP* needs a validation data set. *GP* is computationally expensive for a high number of training samples, especially if linear prediction models are used and the output dimensionality is greater than one. *GP* uses a heuristic merging rule which is not guaranteed to improve the performance of the model. Still, *GP* is an effective method for constructing *RFs* with high accuracy and low model size.

4.1.2.9 Intermediate Refined Random Forests

In this experiment we analyze the performance of *IRRF*. For low D_{max}, the performance of *IRRF* is similar to single refinement approaches. For high D_{max}, we observe overfitting for *IRRF*.

In this experiment we compare the performance of *IRRF* to *ARF+AGR*. We present results on the data set *cpu_small*. This time, we additionally report the *RMSE* on the training data set. Figure 4.16 presents results for different D_{max}.

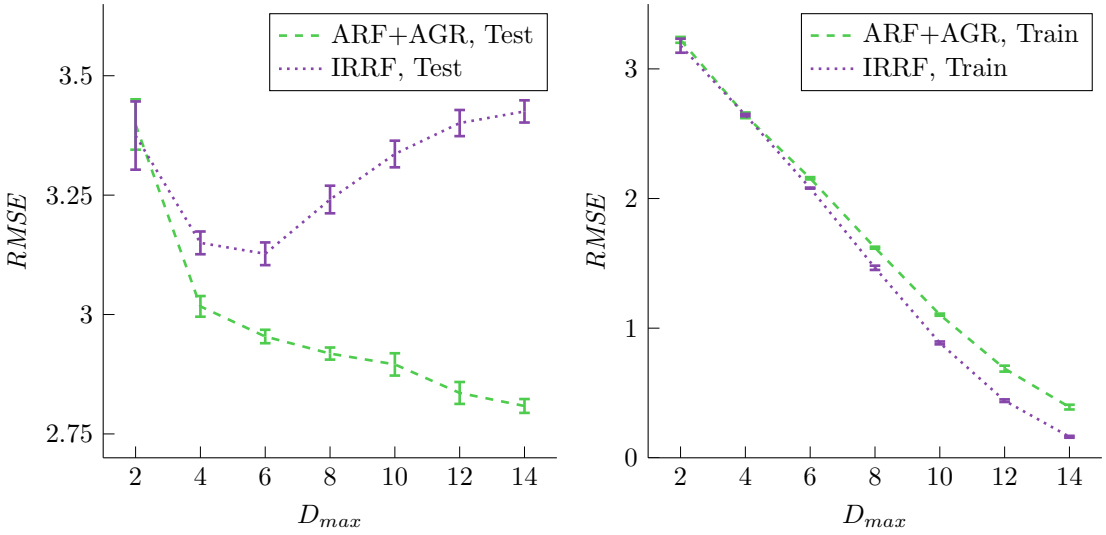


Figure 4.16: Evaluation of the *RMSE* on the training and test data set for *IRRF* and *ARF+AGR*.

IRRF achieves the highest accuracy of all evaluated methods on the training data set. *ARF+AGR* performs second best on the training data set. *IRRF* shows low generalization on the test data set for high D_{max}. *ARF+AGR* improves the performance on the test data set as D_{max} increases. One reason for this is that *ARF+AGR* benefits from the strong generalization of *ARF* at high D_{max}.

Additionally, the contribution of *AGR* is small due to the low norm of the refined prediction models. In contrast to that, we observe that the multiple refinement passes of *IRRF* generate prediction models with higher norm. As a result, *IRRF* is more prone to overfitting. The predictions of each stage build on top of the predictions of the previous stages which are not robust to overfitting.

Another reason for the low performance of *IRRF* is that the individual *DTs* have low strength compared to other methods (see Section 4.1.2.10).

4.1.2.10 Strength and Correlation

In this experiment we analyze the strength and correlation of individual *DTs* for different methods. Breiman [8] shows that the generalization error of *RFs* depends on the strength and correlation of the individual *DTs*. For regression, the residuals of a *DT* indicate the strength of a *DT*. The lower the residuals, the higher the strength.

In this experiment we compare the *RMSE* of a *RF* to the average *RMSE* of the individual *DTs* and report the correlation between the individual *DTs* for $D_{max} = 5$. Table 4.15 presents results for different methods on the data set *auto_mpg*.

auto_mpg

Method	*RMSE* (*RF*)	*RMSE* (*DTs*)	Correlation
RF	3.24 ± 0.02	3.92 ± 0.04	0.67 ± 0.02
RF+GR	3.37 ± 0.09	14.94 ± 0.58	0.10 ± 0.02
RF+AGR	3.18 ± 0.04	6.25 ± 0.16	0.25 ± 0.01
RF+GP	3.22 ± 0.05	12.91 ± 0.78	0.08 ± 0.03
ARF	3.14 ± 0.02	5.54 ± 0.08	0.33 ± 0.02
ARF+GR	3.36 ± 0.13	15.65 ± 0.68	0.10 ± 0.01
ARF+AGR	3.12 ± 0.07	7.06 ± 0.08	0.20 ± 0.01
IRRF	3.51 ± 0.12	25.90 ± 1.61	0.03 ± 0.01

Table 4.15: Evaluation of strength and correlation for different methods. The reported results show the *RMSE*.

RF shows the highest strength, but also the highest correlation. On the other side, *IRRF* shows the lowest correlation, but also the lowest strength. *ARF* shows slightly less strength, but significantly less correlation than *RF*. *AGR* further decreases the strength of individual *DTs* and further decreases the correlation. *GR* and *GP* show low correlation, but also the low strength. In this experiment the best performance is achieved by *ARF+AGR* which finds the best tradeoff between strength and correlation.

We observe that non-refinement approaches give strong, but highly correlated *DTs*. In contrast, refinement approaches give weak, but decorrelated *DTs*. This behavior is reproducible for different D_{max}.

For non-refinement approaches, the prediction of each *DT* roughly has the same magnitude. In contrast, for refinement approaches, the predictions of the individual *DTs*

have significantly different magnitudes. This means that for each test sample some *DTs* are more important than others. Within each *DT* there are some leaves which have a high contribution to the final prediction, while others have a low contribution.

4.1.2.11 Model Size

In the following experiments we analyze the model size of *RF* approaches. We report the model size with respect to specific properties of *RF* approaches and show that different methods perform best at different model sizes. We present results on the data set *elevators*.

Growth of Model Size

In this experiment we analyze the effect of D_{max} and the number of *DTs* on the model size. Figure 4.17 presents results for different D_{max} and different numbers of *DTs*. For the evaluation of D_{max}, we fix the number of *DTs* to 50 and vary D_{max} in the range [0,20]. For the evaluation of the number of *DTs*, we fix the D_{max} to 10 and vary the number of *DTs* in the range [1,100].

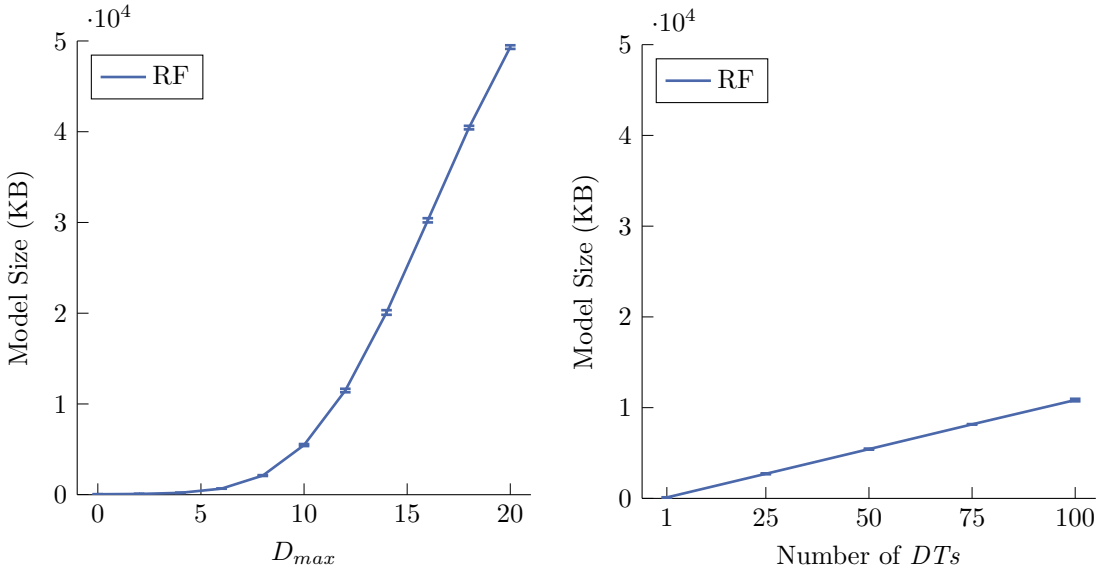


Figure 4.17: Evaluation of the model size of *RFs* as a function of D_{max} and as a function of the number of *DTs*.

The model size of *RFs* increases linearly with the number of *DTs* and exponentially with D_{max}. The exponential growth slows down for $D_{max} \geq 14$, because the number of training samples is limited. The training stops if the number of arriving training samples equals 1. Training deep *DTs* results in large memory demands. The training of many shallow *DTs* is more memory efficient than the training of few deep *DTs*.

Model Size Comparison

In this experiment we compare the model sizes of *RFs* obtained by different training methods. In Figure 4.1 we present the performance for different methods dependent on D_{max}. In this experiment we select the best performing D_{max} for each method and compare the obtained model sizes. Table 4.16 presents results for different methods.

elevators

Method	D_{max}	*RMSE*	Model Size
RF	20	3.16 ± 0.03	$49\,325 \pm 199$
RF+GR	6	2.99 ± 0.06	670 ± 8
RF+AGR	6	2.87 ± 0.07	675 ± 9
RF+GP	12	2.89 ± 0.04	668 ± 24
ARF	20	2.76 ± 0.06	$49\,291 \pm 205$
ARF+GR	6	2.96 ± 0.04	671 ± 7
ARF+AGR	6	2.81 ± 0.05	673 ± 8
IRRF	6	2.96 ± 0.10	679 ± 10

Table 4.16: Evaluation of model size for different methods. The reported results show the *RMSE* with scale $\cdot 10^{-3}$ and the model size in KB.

All methods show a similar level of performance for the best performing D_{max}. Non-refinement approaches perform best for high D_{max}. This corresponds to complex models with large model sizes. In contrast, refinement approaches perform best for low D_{max}. This corresponds to simpler models with smaller model sizes.

Due to the exponential growth of the model size with D_{max}, non-refinement approaches show large memory demands. Shallow refined *RFs* achieve similar performance compared to deep non-refined *RFs*, but show 70 times smaller models. On average the model size obtained by refinement approaches is only 1.37% of the model size obtained by non-refinement approaches in this experiment.

4.1.2.12 Runtime

In the following experiments we analyze the runtime of different methods. We focus on the evaluation of the training time, since the test time is largely independent of the used method. We report the total training time of different methods. Additionally, we report results with respect to specific properties of *RF* approaches. We present results on the data set *pol*. This data set has a medium number of training samples (5 000) and a high number of features (48).

Total Training Time

Table 4.17 presents results for total training time different methods. In this experiment we set D_{max} to 10.

pol	
Method	Training Time
RF	7.53 ± 0.11
RF+GR	7.62 ± 0.12
RF+AGR	7.70 ± 0.15
RF+GP	48.63 ± 0.97
ARF	7.76 ± 0.14
ARF+GR	7.92 ± 0.16
ARF+AGR	7.98 ± 0.17
IRRF	8.67 ± 0.21

Table 4.17: Evaluation of training time for different methods. The reported results show the training time in seconds.

Standard *RF* is the fastest training method. This is no surprise as *RF* is the simplest method and all other methods extend the standard training procedure. *ARF* is only slightly slower than *RF*. The additional costs of *ARF* are due to the computation of prediction models for each stage and updates to the global training objective. Updating the global training objective corresponds to evaluating intermediate *RFs*. Both operations have a low computational impact.

GR and *AGR* are fast. A single refinement pass is processed in 0.1 to 0.2 seconds. The contribution of a single refinement pass to the total training time is negligible. *IRRF* performs multiple refinement passes during training and is only slightly slower than non-refinement and single refinement approaches.

GP is slow. While a single refinement pass is fast, *GP* performs 300 refinement passes and pruning iterations. The large number of iterations has a significant impact on the training time. In [77] *GP* is utilized for Kinect body part classification. There, the training of the initial *RF* takes 2 days and the pruning takes almost 4 days for only 40 iterations.

Specific Training Time

In this experiment we analyze the training time for growing the structure of *RFs* and refining *RFs* with respect to specific properties. We evaluate the effect of D_{max} and the prediction model on the training time. We perform *RF+GR* using different parameters. Figure 4.18 presents results for two different D_{max} with constant and linear prediction models.

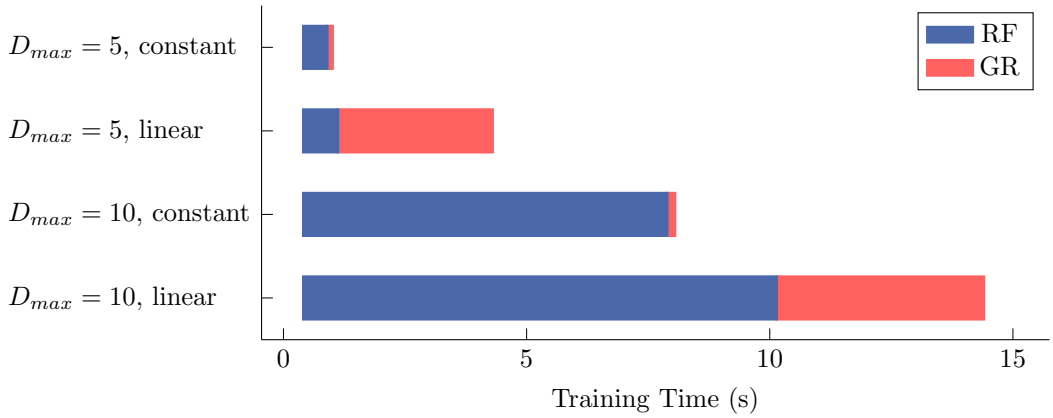


Figure 4.18: Evaluation of training time for *RF+GR*. We evaluate two different D_{max} with constant and linear prediction models.

The training time of *RF* is strongly influenced by D_{max}, but less sensitive to the prediction model. Training the structure of a *RF* is fast for low D_{max}. The higher D_{max}, the slower the training. Training *RFs* with linear prediction models is only slightly slower than training *RFs* with constant prediction models. The reason for this is that the computation of linear prediction models is more complex. However, we use closed form Tikhonov regularization to compute the weights for linear prediction models efficiently [92].

In contrast, the training time of *GR* is strongly influenced by the prediction model, but less sensitive to D_{max}. *GR* for high D_{max} is only slightly slower than *GR* for low D_{max}. *GR* is efficient for constant prediction models, but expensive for linear prediction models. One reason for this is that the computation of *GR* using LIBLINEAR is strongly influenced by the number of non-zero entries of the optimization problem, but only slightly influenced by the dimensions of the optimization problem (see Figure 2.5 and 3.2).

The structure of a *RF* accounts for the dimensions of the optimization problem. Considering constant prediction models, each column of the optimization problem has one entry for each leaf node of the *RF*. However, the number of non-zero entries per column only equals the number of *DTs*. Even for shallow *RFs* the number of leaf nodes is significantly higher than the number of *DTs*. LIBLINEAR [26] efficiently exploits this sparsity. In contrast, the optimization problem for linear prediction models has F times more entries and F times more non-zero entries, where F is the feature dimensionality.

Parallelization

In this experiment we analyze the effect of parallelization on the training time of different methods. For *RFs*, the construction of the individual *DTs* is fully parallelizable, since all *DTs* are independent of each other. For *ARFs*, the training of each stage is parallelizable. Our implementation supports parallel training for *RFs* and *ARFs* on multiple cores.

Table 4.18 presents results for parallel training of *RFs* and *ARFs*. In this experiment we set D_{max} to 10 and use a machine with 4 cores.

pol

Method	Cores	Training Time
RF	1	7.53 ± 0.11
RF	4	2.89 ± 0.07
ARF	1	7.76 ± 0.14
ARF	4	3.07 ± 0.05

Table 4.18: Evaluation of parallelization for *RFs* and *ARFs*. The reported results show the training time in seconds.

For both evaluated methods we observe a speedup by a factor of 2.5 for parallel training on 4 cores. The speedup is slightly lower for *ARFs*, since the global training objective is updated after the parallel training of each stage.

For low D_{max}, the speedup is modest, because the overhead of the parallel implementation accounts for a higher percentage of the training time. The speedup is also dependent on the number of *DTs*. In this experiment we train 50 *DTs*.

4.2 Super-Resolution

In this section we evaluate different *RF* approaches for single image *SR* [30, 37, 71]. We use the *SR* framework provided by Schulter *et al.* [88] which is based upon the code of Timofte *et al.* [93]. The framework uses bicubic interpolation [54] to obtain the desired output resolution followed by a sharpening using machine learning techniques. We exchange the provided learning algorithm with our *RF* toolbox. Before we present results, we briefly describe the *SR* pipeline of the framework.

The *SR* pipeline implements a mapping from low-resolution (LR) to high-resolution (HR) images. First, images are transformed from RGB to YCbCr color space [40, 80] which separates luminance information (Y) and color information (Cb and Cr). The reason for this is that the human visual system is most sensitive to high frequency changes in luminance [28]. Color information only plays a minor role in the human perception of sharpness. Therefore, bicubic interpolation is applied to all image channels, but sharpening is only performed on the Y channel. The framework takes a patch based approach to *SR*. Overlapping patches are extracted from the luminance channel of the upscaled *LR* image. A machine learning algorithm is used to estimate a high frequency patch for each upscaled *LR* patch. This high frequency patch corrects for the blur in the upscaled *LR* patch. Therefore, the two patches are summed up to obtain the final sharpened patch. Finally, the sharpened patches are stitched together to obtain the sharpened output image.

We use *RFs* to estimate the high frequency patches based on features extracted from the upscaled *LR* patches. For the features, we compute the first and second order derivates

of the upscaled *LR* patches and apply dimensionality reduction using PCA [38, 52]. The dimensionality reduction preserves 99.9% of the average energy [93]. This leads to around 30 features depending on the training data.

We use the training data set provided by Yang *et al.* [98] which has also been used in [22, 88, 93, 94]. This data set consists of 91 images which show natural scenes including nature, animals and people. The resolution of the images is around 0.1 megapixel. We sample a total of one million patches for training. We report the performance on two test data sets, *Set5* [4] and *Set14* [100]. These data sets have been used for benchmarking different *SR* methods in [16, 22, 23, 55, 56, 88, 93, 94]. We evaluate the performance by computing the Peak Signal-to-Noise Ratio (PSNR) [47] between the ground truth and the prediction. For selected experiments, we report the *PSNR* improvement over bicubic interpolation which we refer to as *gain* in *dB* [22]. In contrast to the performance metrics for standard machine learning tasks, higher *PSNR* or *gain* corresponds to better performance. We perform 3 independent runs for each experiment [88]. All Figures and Tables report the mean and the standard deviation of the independent runs.

Default Parameters - Super-Resolution

Category	Name	Value
Complexity	Number of *DTs*	15
	Threshold minCount	128
	Threshold minChild	64
Randomness	Bagging Ratio	1.0
	Bagging Replacement	no
	Subsampling at Node Level	512
	Feature Subsampling	$F/2$
Splits	Split Function	Unary + Binary
	Split Compactness Measure	Variance (GT + F)
Predictions	Leaf Prediction Model	linear
	Tikhonov Regularization Term	0.01
ARF	ARF Loss	Squared Loss
GR	Refinement Regularization	0.05
	Refinement Loss	L2 Loss
GP	Pruning Iterations	50
	Pruning Ratio	0.3
	Pruning Strategy	Least Significance

Table 4.19: Default parameters for *SR*. The role of the different parameters is explained and evaluated in the individual subsections. Unless otherwise stated, these parameters are used for all *SR* experiments.

Since *SR* is a regression task, we can use the same *RF* approaches as for standard machine learning tasks for regression (see Table 4.5). However, we focus on two non-refinement approaches (*RF* and *ARF*) and two refinement approaches (*ARF+GR*

and *ARF+AGR*) due to the following reasons. First, the experiments in Section 4.1.2 show that the performance of *GR* and *AGR* is largely independent of the underlying *DT* growing scheme (*RF* and *ARF*). Therefore, we do not evaluate *RF+GR* and *RF+AGR*, since their performance is resembled by *ARF+GR* and *ARF+AGR*. Second, the experiments in Section 4.1.2 show that the performance of *RF* approaches which perform multiple refinement passes (*GP* and *IRRF*) is not superior to the performance of single refinement approaches. However, for *SR*, even a single refinement pass is computationally expensive (see Section 4.2.7). Because of this, we focus on the evaluation of single refinement approaches, but provide results for multi refinement approaches in dedicated experiments.

Unless otherwise stated, we use the parameters in Table 4.19 for all experiments. In contrast to standard machine learning tasks, we use linear prediction models in all experiments. Additionally, we use higher thresholds for early stopping, analyze more subsamples at node level and adjust the regularization parameters, because of the significantly increased size of the training data set (one million samples). We perform experiments for different *SR* factors. The *SR* factor specifies the scale which is used to resize each image dimension. Considering a *SR* factor of $\times 3$, the number of pixels in the super-resolved image is 9 times the number of pixels in the *LR* image. For parameter evaluation we present results for a *SR* factor of $\times 3$ on the data set *Set5*.

4.2.1 Overall Results

In this experiment we compare the performance of *RF* approaches on *Set5* [4] and *Set14* [100]. We use the same parameters (see Table 4.19) for each method to obtain an unbiased comparison. However, different methods perform best with different parameters. The most important parameter concerning accuracy on previously unseen data is the maximum tree depth D_{max}. Therefore, we evaluate multiple D_{max} for each method. In this experiment we vary D_{max} in the range [0,12] and select the best performing D_{max}. We report the *PSNR* and the corresponding D_{max} for each method. Table 4.20 presents experimental results for different *SR* factors.

Analogous to standard machine learning tasks for regression, all evaluated methods show a similar level of performance for the best performing D_{max}. However, the best performance is achieved by deep *ARF*. We observe this phenomenon across different data sets and *SR* factors.

Compared to *ARF*, the performance of refinement approaches is slightly lower, but the best performing D_{max} is significantly lower. *RFs* with lower D_{max} correspond to simpler models. For *SR*, the main advantage of simpler models is the reduction in model size.

Overall Results - Super-Resolution

Data Set	SR Factor	Property	Methods			
			RF	ARF	ARF+GR	ARF+AGR
Set5	×2	*PSNR*	36.48 ± 0.03	36.68 ± 0.02	36.50 ± 0.02	36.55 ± 0.02
		D_{max}	12	12	6	8
	×3	*PSNR*	32.38 ± 0.02	32.54 ± 0.01	32.46 ± 0.01	32.50 ± 0.01
		D_{max}	12	12	6	10
	×4	*PSNR*	30.12 ± 0.01	30.22 ± 0.02	30.11 ± 0.01	30.13 ± 0.01
		D_{max}	12	12	6	8
Set14	×2	*PSNR*	32.23 ± 0.03	32.36 ± 0.01	32.24 ± 0.01	32.27 ± 0.02
		D_{max}	12	12	6	8
	×3	*PSNR*	28.99 ± 0.02	29.11 ± 0.02	29.05 ± 0.01	29.07 ± 0.01
		D_{max}	12	12	6	8
	×4	*PSNR*	27.20 ± 0.01	27.31 ± 0.02	27.23 ± 0.01	27.25 ± 0.01
		D_{max}	12	12	6	8

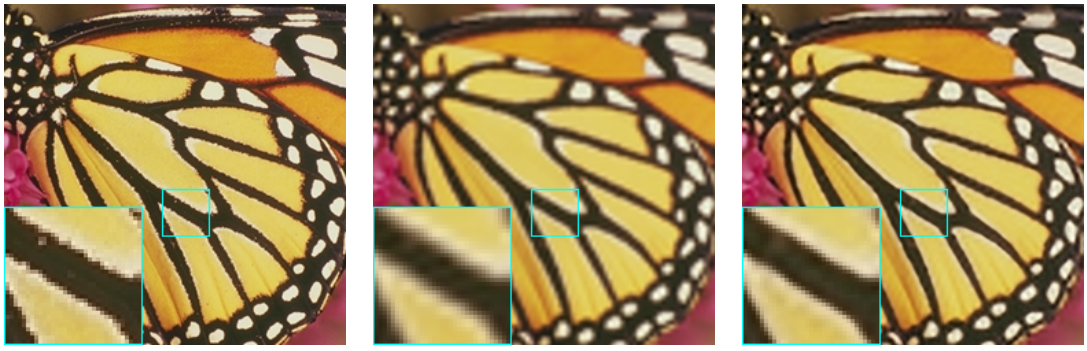
Table 4.20: Overall results for *SR*. For each data set and each *SR* factor, the three best performing methods are highlighted in shades of green.

4.2.2 Qualitative Results

Figure 4.19 and 4.20 present qualitative *SR* results for selected images from *Set5* for a *SR* factor of ×3. For both images, we observe that all evaluated methods show a similar level of performance. However, the *PSNR* improvement is different for each image. The performance of our *SR* methods is dependent on local image topology and structure.

We observe higher *gain* for images with sharp discontinuities between homogenous regions, e.g., *butterfly* (see Figure 4.19). One reason for this is that edges and ridges between homogenous regions are common elements of natural images. We can easily harvest patches for these scenarios from training images and effectively learn an appropriate mapping for them, especially since sharpening is only performed on the luminance channel.

In contrast to this, we observe lower *gain* for images with a large fraction of textured areas, e.g., *head* (see Figure 4.20). One reason for this is that different mechanisms in the *SR* framework enforce smoothness. Among these mechanisms are Tikhonov regularization [92] for local leaf prediction models, L2 regularized L2 loss for global leaf prediction model optimization, averaging of *DT* results and averaging of overlapping patches. Additionally, the texture of different materials is unique, e.g., the texture of human skin is significantly different from the texture of cloth or metal. Textures are less generic than edges between homogenous regions and, therefore, harder to learn. Given a blurred patch, the reconstruction of irregularly shaded regions with material specific patterns is difficult, even if prior knowledge is exploited. Many *SR* methods show high performance at intensifying blurred edges and ridges, but low performance at recovering fine grained textures [4, 16, 88, 93, 94, 97, 100].



(a) Ground Truth **(b)** Bicubic: 24.04 dB **(c)** RF: +2.80 dB

(d) ARF: +3.27 dB **(e)** ARF+GR: +3.12 dB **(f)** ARF+AGR: +3.23 dB

Figure 4.19: Qualitative *SR* results for *butterfly* from *Set5*. We report the *PSNR* for the bicubic upscaled image and the *gain* for different sharpening methods.

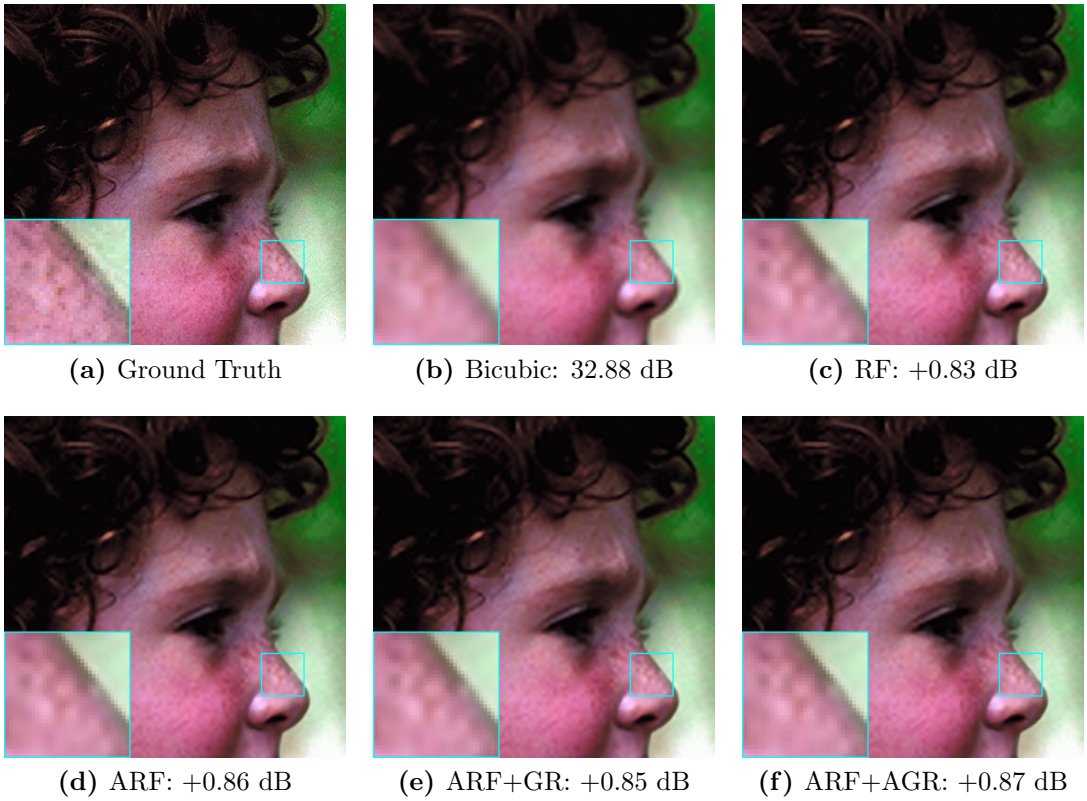
4.2.3 State of the Art

In this experiment we compare the performance of different state of the art methods on *Set5* [4] and *Set14* [100]. Table 4.21 presents experimental results for different *SR* factors.

State of the Art - Super-Resolution

		Methods					
Data Set	SR Factor	A+ [94]	ARF [88]	SRCNN [22] (ECCV)	SRCNN [23] (PAMI)	VDSR [55]	DRCN [56]
Set5	×2	36.55	36.70	36.34	36.66	37.53	37.63
	×3	32.59	32.58	32.39	32.75	33.66	33.82
	×4	30.29	30.21	30.09	30.49	31.35	31.53
Set14	×2	32.28	32.37	32.18	32.45	33.03	33.04
	×3	29.13	29.13	29.00	29.30	29.77	29.76
	×4	27.33	27.30	27.20	27.50	28.01	28.02

Table 4.21: Comparison of state of the art methods for *SR*. For each data set and each *SR* factor, the three best performing methods are highlighted in shades of green.



(a) Ground Truth **(b)** Bicubic: 32.88 dB **(c)** RF: +0.83 dB

(d) ARF: +0.86 dB **(e)** ARF+GR: +0.85 dB **(f)** ARF+AGR: +0.87 dB

Figure 4.20: Qualitative *SR* results for *head* from *Set5*. We report the *PSNR* for the bicubic upscaled image and the *gain* for different sharpening methods.

Dictionary [94] and *RF* approaches show a similar level of performance. However, both are outperformed by recent Convolutional Neural Network (CNN) approaches [23, 55, 56]. Depending on the data set DRCN [56] performs 0.6 to 1.3 dB better than *ARF*. Apart from improved performance, one advantage of *CNNs* is that they learn a representation of the input at hidden layers, while dictionary and *RF* approaches rely on handcrafted features. One disadvantage is that deep architectures may require a GPU implementation for fast training and testing [55].

The number of Floating-Point Operations (FLOPs) is an implementation independent measure for the computational workload of an algorithm. We compare the number of *FLOPs* required to super-resolve a 512×512 image for *RF* approaches and *SRCNN* [22], which achieve a similar level of performance. For *RF* approaches, D_{max} has a negligible impact on the test time. *RF* approaches require 2.75 *GFLOPs*, while *SRCNN* requires 4.77 *GFLOPs*. More advanced *CNNs* may require even more *FLOPs*. In comparison to these *SR* methods, bicubic interpolation requires 0.012 *GFLOPs* to upscale a single channel *LR* image to 512×512 pixel [64].

Alternatively, there are other ways of improving the accuracy of *SR* methods apart from using a more powerful learning algorithm. Timofte *et al.* [95] present several adjustments to the *SR* framework which boost the performance of *A+* [94] by up to 1 dB. Most of these enhancements are applicable to our *SR* approach, e.g., using more training patches, training deeper *DTs* or training a cascade of *RFs*. However, these enhancements significantly increase the training time and the model size.

4.2.4 Maximum Depth

In this experiment we show the effect of D_{max} on the performance of *RF* approaches. Figure 4.21 presents results for different methods. In this experiment we report the *gain* in dB and vary D_{max} in the range [0,12].

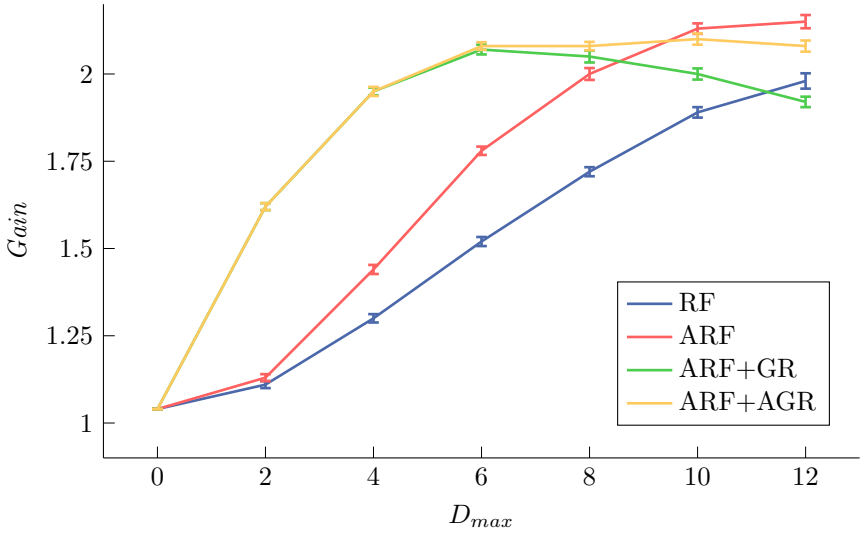


Figure 4.21: Evaluation of the maximum tree depth for D_{max} for *SR*.

We observe the same behavior as for standard machine learning tasks. Refinement approaches outperform non-refinement approaches for low D_{max}, but show overfitting for high D_{max}. The best performance is achieved by deep *ARF*.

Schulter *et al.* [88] train even deeper *ARF* ($D_{max} = 15$), but only increase the mean performance across different *SR* factors by negligible 0.02 dB compared to our *ARF* with $D_{max} = 12$.

On the other end of the range a uniform linear regression for all samples at $D_{max} = 0$ shows a considerable *gain* of 1 dB compared to the bicubic baseline. In this case, *RF* approaches resemble Global Regression [93].

Interestingly, we observe that the performance improvement of non-refinement approaches between $D_{max} = 0$ and $D_{max} = 2$ is low for *SR*. In contrast to this, the initial D_{max} increments for standard machine learning tasks show the highest performance improvement (see Section 4.1.2.2). One reason for this is the significantly larger size of

the training data set compared to standard machine learning tasks. For *SR*, we use one million training samples. At $D_{max} = 2$ a balanced *DT* has 4 leaves and the entire training data set is distributed among these leaves. Therefore, the average number of samples arriving at a node is 250 000. In our experiment lowest number of arriving samples is 25 000. At $D_{max} = 2$ the partitioning of the training data set is too coarse to fit local prediction models which significantly outperform ordinary linear regression.

4.2.5 Number of Decision Trees

In this experiment we show the effect of the number of *DTs* on the performance of *RF* approaches. Figure 4.22 presents results for different methods. In this experiment we vary the number of *DTs* in the range [1,25] and set D_{max} to 6.

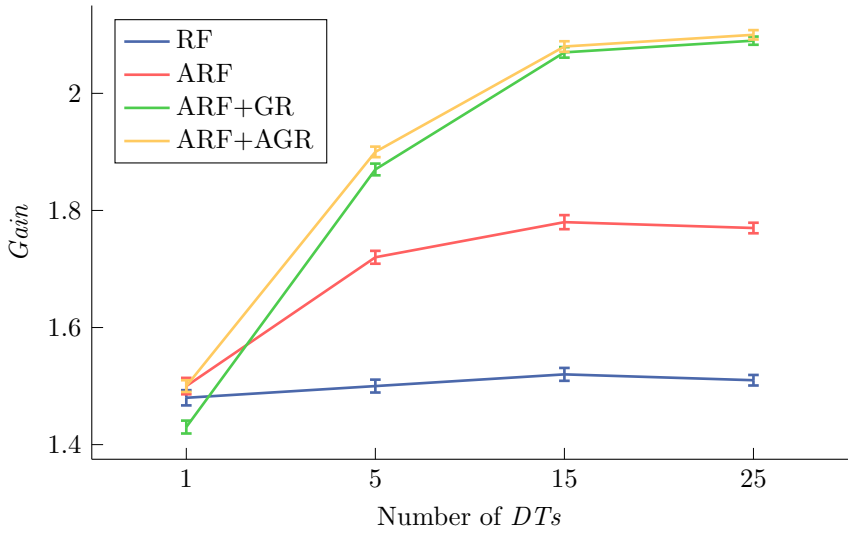


Figure 4.22: Evaluation of the number of *DTs* for *SR*.

Analogous to standard machine learning tasks for regression, all methods show low performance if the number of *DTs* is small. As the number of *DTs* rises, the performance increases, but the effect saturates at some point.

Interestingly, *RF* only slightly benefits from a higher number of *DTs* in this experiment. One reason for this is that *RF* performs best for higher D_{max}. At $D_{max} = 6$ the number of training samples arriving at each leaf is high, due to the large size of the training data set. Therefore, the partitioning is too coarse to fit accurate prediction models. As a result, the individual *DTs* show high bias. Ensembles of *DTs* can reduce variance, but not bias.

4.2.6 Model Size

In this experiment we compare the model sizes obtained by *RF* approaches. Table 4.22 presents results using the best performing D_{max} for each method.

Model Size Comparison

Method	D_{max}	*PSNR*	Model Size
RF	12	32.38 ± 0.02	447.54 ± 1.86
ARF	12	32.54 ± 0.01	446.71 ± 1.97
ARF+GR	6	32.46 ± 0.01	19.81 ± 0.00
ARF+AGR	6	32.48 ± 0.01	19.81 ± 0.00

Table 4.22: Evaluation of model size for different methods. The reported results show the *PSNR* in dB and the model size in MB.

Similar to previous experiments, all evaluated methods show a similar level of performance for the best performing D_{max}. Non-refinement approaches perform best for higher D_{max}. This corresponds to complex models with large model sizes. Refinement approaches perform best for lower D_{max}. This corresponds to simpler models with smaller model sizes.

Non-refinement approaches show high memory demands, due to the exponential growth of the model size with D_{max}. In contrast, the model size obtained by refinement approaches is more than 22 times smaller. This corresponds to 4.43% of the model size obtained by non-refinement approaches.

Interestingly, the standard deviation of the model size for refinement approaches is zero. This means that even at $D_{max} = 6$ the training algorithm constructs fully balanced *DTs*, due the large size of the training data set.

The model sizes for *SR* are significantly larger than for standard machine learning tasks, although we train a lower number of *DTs*. One reason for this is that we use linear prediction models for *SR*. For linear prediction models, the number of weights stored at each leaf equals input dimensionality times output dimensionality. In our configuration the input dimensionality is 31, because we use 30 PCA features and a constant intercept term.

In contrast to standard machine learning tasks for regression, the patch based *SR* approach is a multivariate regression problem which means that the output dimensionality is greater than one [78]. In our implementation the output dimensionality is dependent on the *SR* factor. The output dimensionality corresponds to the pixel count of a 3×3 *LR* patch which is upscaled by the *SR* factor. For a *SR* factor of ×3, we predict 9×9 patches. In this case, the output dimensionality is 81. Therefore, in this experiment each leaf stores 2 511 weights.

4.2.7 Runtime

In the following experiments we analyze the runtime of *RF* approaches. We focus on the evaluation of the training time. We report the total training time of different methods. Additionally, we present results for parallel training.

The test time required to super-resolve one image is below one second in all experiments, even if deep *RFs* are evaluated. However, the *SR* framework is not optimized for speed and there is potential for accelerating of both training and testing.

Total Training Time

In this experiment we compare the training time of *RF* approaches. Table 4.23 presents results using the best performing D_{max} for each method.

Runtime

Method	D_{max}	*PSNR*	Training Time
RF	12	32.38 ± 0.02	248 ± 13
ARF	12	32.54 ± 0.01	519 ± 21
ARF+GR	6	32.46 ± 0.01	$66\,423 \pm 931$
ARF+AGR	6	32.48 ± 0.01	$67\,112 \pm 945$

Table 4.23: Evaluation of training time for different methods. The reported results show the training time in seconds.

RF is the fastest training method. *ARF* is by a factor of 2 slower than *RF*. The additional costs of *ARF* are mainly due to updates to the global training objective after the training of each stage. Updating the global training objective corresponds to evaluating intermediate *RFs*. The evaluation of a *RF* is fast, but in this case one million training samples are evaluated. This increases the training time significantly.

Despite the reduced D_{max}, the training time of refinement approaches is high compared to non-refinement approaches. Although we only perform a single refinement pass, *ARF+GR* and *ARF+AGR* are by a factor of 130 slower than *ARF*. The training of *ARF* takes 8 to 9 minutes, while the training of *ARF+AGR* takes 18 to 19 hours.

Considering refinement approaches, the contribution of the *DT* growing algorithm to the total training time is negligible. In contrast to that, for standard machine learning tasks, the contribution of a single refinement pass to the total training time is negligible.

Our experiments show that the training time of a global leaf prediction model optimization using LIBLINEAR [26] is dependent on the number of non-zero entries of the optimization problem (see Section 4.1.2.12). This number is influenced by the number of *DTs*, the leaf prediction model and the number of training samples. In this experiment we use 15 *DTs*, linear leaf prediction models with 31 prediction weights per output dimension and one million training samples which corresponds to 450 million

non-zero entries. The large size of the training data set accounts for the major share and is the main reason for the increased training time in this case.

Additionally, LIBLINEAR only supports univariate linear regression. However, for a *SR* factor of ×3, the output dimensionality is 81 in our implementation. This means we have to solve 81 univariate linear regression problems, which all have the same input variables but different optimization targets. The optimization of a single univariate linear regression problem takes 13 to 14 minutes.

Parallelization

In this experiment we analyze the effect of parallelization on the training time of *RF* approaches. For *SR*, we focus on the parallelization of refinement techniques, since the contribution of the *DT* growing algorithm to the total training time is negligible. Table 4.24 presents results for parallel optimization of *GR* and *AGR* with $D_{max} = 6$. In this experiment we use a machine with 8 cores.

Parallel Refinement

Method	Cores	Training Time
ARF+GR	1	$66\,423 \pm 931$
ARF+GR	8	$10\,692 \pm 419$
ARF+AGR	1	$67\,112 \pm 945$
ARF+AGR	8	$10\,877 \pm 438$

Table 4.24: Evaluation of parallelization for *GR* and *AGRs*. The reported results show the training time in seconds.

Our implementation uses Multi-Core LIBLINEAR [59] which is a LIBLINEAR extensions that supports parallel L2 loss minimization of a univariate linear regression problem. We observe a speedup by a factor of 6.2 for 8 cores. Parallel refinement using Multi-Core LIBLINEAR reduces the training time of *ARF+GR* and *ARF+AGR* from 18.5 hours to 3 hours.

Another possible parallelization approach is solving the different univariate linear regression problems for each output dimension in parallel. However, parallel loops in MATLAB® are limited to input matrices with a maximum size of 2 GB. Since LIBLINEAR operates strictly on double precision floating point numbers and indices for each non-zero entry must be provided, we end up with 450 million non-zero entries which correspond to 7.2 GB of data. Therefore, we do not provide results for parallel training of the independent univariate linear regression problems.

Although we make use of parallelization, a single refinement pass is slow compared to the time needed to construct a *RF*. The training of multi refinement approaches is prohibitively slow for *SR*, e.g., the training of *RF+GP* with $D_{max} = 12$ takes more than 1.5 weeks for 50 iterations despite parallel refinement.

4.2.8 Global Pruning

In this experiment we analyze the progression of *GP* over multiple iterations. We perform 50 pruning iterations with a pruning ratio of 0.3. Figure 4.23 shows the performance of *RF+GP* ($D_{max} = 10$) over multiple iterations compared to *RF* ($D_{max} = 10$) and *RF+GR* ($D_{max} = 6$).

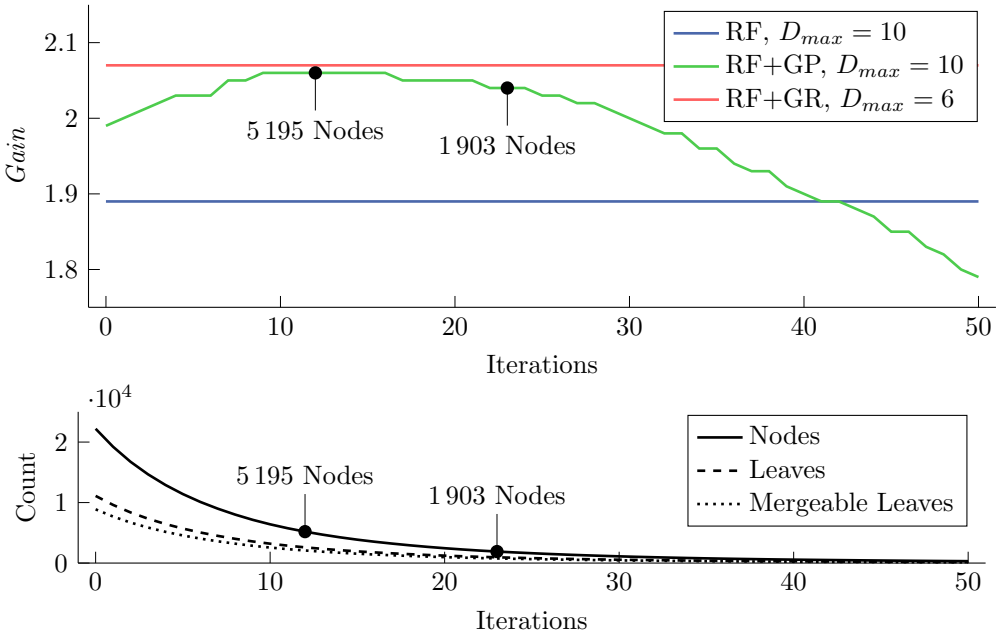


Figure 4.23: Progression of *GP* over multiple iterations for *SR*.

RF+GP achieves the best performance after 12 iterations, at this point the model consists of 5 195 nodes. *GP* simplifies the model in each iteration. In the beginning iterative *GP* reduces overfitting, but at some point the model becomes too simple and we observe underfitting for *RF+GP*.

RF+GR outperforms *RF+GP* in this experiment. Interestingly, *RF+GR* achieves better performance with less nodes compared to *RF+GP*. *RF+GR* shows maximum performance at a node count of 1 905. In contrast, *RF+GP* shows maximum performance at a node count of 5 195. If we prune *RF+GP* to a comparable node count of 1 903 (23 iterations) the performance gap increases.

RF+GP outperforms *RF* until iteration 40. At this point the model consists of 565 nodes. This is a significant reduction in model size compared to *RF* with $D_{max} = 10$ (22 201 nodes).

For *SR*, we observe that pre-pruned shallow *RF+GR* outperforms post-pruned deep *RF+GP*. Moreover, the training of pre-pruned shallow *RF+GR* is significantly faster, because we train *DTs* with lower D_{max} and perform a single refinement pass only.

CHAPTER 5

Conclusion

In this work we present three methods for constructing Random Forests (RFs) with reduced model size under a global training objective: Global Refinement of Alternating Decision and Regression Forests (ADRFs+GR), Additive Global Refinement (AGR) and Intermediate Refined Random Forests (IRRFs). Our methods combine the benefits of Alternating Decision and Regression Forests (ADRFs) [89, 90] and Global Refinement (GR) of *RFs* [77]. In most experiments our methods show significantly reduced model size while achieving competitive performance compared to state of the art *RF* approaches. For standard machine learning tasks, we observe a compression of factor 70. For single image Super-Resolution (SR), we observe a compression of factor 22.

ADRFs+GR perform *GR* on *RFs* constructed by the alternating training procedure of *ADRFs*. In most experiments *GR* benefits from the more decorrelated Decision Trees (DTs) of *ADRFs*. We observe improved performance compared to *GR* of standard *RFs*.

AGR improves the robustness of refinement approaches to overfitting. In contrast to *GR*, *AGR* improves the existing prediction models of *RFs* instead of relearning them from scratch. In this way, the contribution of the refinement to the final prediction is reduced. As a result, refined *RFs* are less prone to overfitting.

IRRFs interweave the construction and refinement of *RFs*. For models with low complexity, *IRRFs* show improved performance compared to standard *RFs*.

Additionally, we present a global refinement algorithm for linear leaf prediction models. For both the refinement of constant and linear prediction models, we observe significantly improved performance for *RFs* with low complexity.

We show that it is possible to replace Global Pruning (GP) [77] by directly training *RFs* with low complexity followed by a single global leaf prediction model optimization, without compromising on accuracy or model size. While *GP* achieves a similar level of

performance compared to our training approach in terms of accuracy and model size, the training is slow, because the algorithm iteratively simplifies and refines pre-trained *RFs* with high complexity. Overall, our approach is significantly faster regarding both construction and refinement of *RFs*, since we train more shallow *DTs* and only apply a single global leaf prediction model optimization.

For single refinement approaches, the selection of the optimal model complexity is critical, since refinement techniques show overfitting for complex models. The selection of the optimal model complexity is less critical if *AGR* is employed. In contrast to *GP*, *AGR* significantly reduces overfitting, but cannot completely avoid overfitting. However, the training of single refinement approaches for different model complexities is computationally less expensive than *GP*.

In contrast to non-refinement approaches, refinement approaches are more prone to overfitting as the complexity of the *DTs* increases. For *RFs* with high complexity, non-refinement approaches are more competitive to refinement approaches. For *SR*, deep Alternating Regression Forests (ARFs) slightly outperform refinement approaches in terms of accuracy. However, the modest performance improvement achieved by deep *ARFs* is disproportionate to the significantly increased model size.

Our experiments show that refinement techniques are computationally expensive for large problems. A large number of training samples, a high number of *DTs* or multiple output dimensions significantly increase the training time. To overcome this limitation, we use parallel refinement which significantly accelerates the training and achieves almost linear speedup. Still, *GP* is prohibitively slow for *SR*. In contrast to *GP*, our training approach is two orders of magnitude faster.

Finally, we conclude that refinement techniques are an effective method for boosting the performance of *RFs* with low complexity. In contrast to previous works, we present a *SR* approach which is accurate, fast and memory efficient. Additionally, our training approach is significantly faster than other approaches, which reduce the model size of *RFs* without compromising on accuracy.

Bibliography

[1] Yali Amit and Donald Geman. Shape Quantization and Recognition with Randomized Trees. *Neural Computation*, 9(7):1545–1588, 1997. (page 10)

[2] Rodrigo Barros, André de Carvalho, and Alex Alves Freitas. *Automatic Design of Decision-Tree Induction Algorithms*. Springer, 2015. (page 9)

[3] Simon Bernard, Sébastien Adam, and Laurent Heutte. Dynamic Random Forests. *Pattern Recognition Letters*, 33(12):1580–1586, 2012. (page 2, 17)

[4] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie-Line Morel. Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding. In *British Machine Vision Conference*, 2012. (page 1, 58, 59, 60, 61)

[5] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. (page 1, 5, 21)

[6] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. (page 15, 20, 49)

[7] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996. (page 10, 39)

[8] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001. (page 2, 10, 11, 39, 44, 52)

[9] Leo Breiman, Jerome Friedman, Charles Stone, and Richard Olshen. *Classification and Regression Trees*. CRC Press, 1984. (page 6, 8, 9, 10)

[10] Rich Caruana and Alexandru Niculescu-Mizil. Data Mining in Metric Space: An Empirical Analysis of Supervised Learning Performance Criteria. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004. (page 25, 27, 29)

[11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2(4):1–27, 2011. (page 21, 28)

[12] Hong Chang, Dit-Yan Yeung, and Yimin Xiong. Super-Resolution Through Neighbor Embedding. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2004. (page 1)

[13] Thomas Cover and Joy Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012. (page 9)

[14] Koby Crammer and Yoram Singer. On the Algorithmic Implementation of Multiclass Kernel-Based Vector Machines. *The Journal of Machine Learning Research*, 2(1):265–292, 2002. (page 27)

[15] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2–3):81–227, 2012. (page 8)

[16] Dengxin Dai, Radu Timofte, and Luc Van Gool. Jointly Optimized Regressors for Image Super-Resolution. In *Computer Graphics Forum*, 2015. (page 1, 58, 60)

[17] Thomas Dietterich. An Experimental Comparison of three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, 2000. (page 8)

[18] Thomas Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*, pages 1–15. Springer, 2000. (page 10)

[19] Piotr Dollár. Piotr's Computer Vision MATLAB® Toolbox. Website, 2014. University of California, San Diego. `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html` last visited 12th March 2016. (page 26)

[20] Piotr Dollár and C. Lawrence Zitnick. Structured Forests for Fast Edge Detection. In *IEEE International Conference on Computer Vision*, 2013. (page 7)

[21] Pedro Domingos. A Unified Bias-Variance Decomposition. In *International Conference on Machine Learning*, 2000. (page 10)

[22] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning A Deep Convolutional Network for Image Super-Resolution. In *European Conference on Computer Vision*, 2014. (page 1, 58, 61, 62)

[23] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image Super-Resolution Using Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016. (page 1, 58, 61, 62)

[24] Claude Duchon. Lanczos Filtering in One and Two Dimensions. *Journal of Applied Meteorology*, 18(8):1016–1022, 1979. (page 1)

[25] Richard Duda, Peter Hart, and David Stoark. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973. (page 39)

[26] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9(1):1871–1874, 2008. (page 26, 46, 56, 66)

[27] Sean Fanello, Cem Keskin, Pushmeet Kohli, Shahram Izadi, Jamie Shotton, Antonio Criminisi, Ugo Pattacini, and Tim Paek. Filter Forests for Learning Data-Dependent Convolutional Kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. (page 8, 9, 21)

[28] Christoph Feichtenhofer, Hannes Fassold, and Peter Schallauer. A Perceptual Image Sharpness Metric based on Local Edge Gradient Analysis. *IEEE Signal Processing Letters*, 20(4):379–382, 2013. (page 57)

[29] Eibe Frank. *Pruning Decision Trees and Lists*. PhD thesis, University of Waikato, 2000. (page 2, 9)

[30] William Freeman, Egon Pasztor, and Owen Carmichael. Learning Low-Level Vision. *International Journal of Computer Vision*, 40(1):25–47, 2000. (page 1, 57)

[31] Yoav Freund and Robert Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. (page 11, 12)

[32] Jerome Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5):1189–1232, 2001. (page 12)

[33] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*. Springer, 2009. (page 5, 10, 44)

[34] Jerome Friedman and Jacqueline Meulman. Multiple Additive Regression Trees with Application in Epidemiology. *Statistics in Medicine*, 22(9):1365–1381, 2003. (page 5)

[35] Johannes Fürnkranz, Dragan Gamberger, and Nada Lavrač. *Foundations of Rule Learning*. Springer Science & Business Media, 2012. (page 2, 9, 16)

[36] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely Randomized Trees. *Machine Learning*, 63(1):3–42, 2006. (page 44)

[37] Daniel Glasner, Shai Bagon, and Michal Irani. Super-Resolution from a Single Image. In *IEEE International Conference on Computer Vision*, 2009. (page 1, 57)

[38] Gene Golub and Christian Reinsch. Singular Value Decomposition and Least Squares Solutions. *Numerische Mathematik*, 14(5):403–420, 1970. (page 58)

[39] Gaël Guennebaud and Benoît Jacob. Eigen v3. Website, 2010. University of California, San Diego. `http://eigen.tuxfamily.org` last visited 18th March 2016. (page 26)

[40] Eric Hamilton. JPEG File Interchange Format. *C-Cube Microsystems*, 1(1):1–22, 1992. (page 57)

[41] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Elsevier, 2011. (page 30)

[42] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Unsupervised Learning*. Springer, 2009. (page 12)

[43] Tin Kam Ho. Random Decision Forests. In *IEEE International Conference on Document Analysis and Recognition*, 1995. (page 10)

[44] Tin Kam Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998. (page 8, 10)

[45] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single Image Super-Resolution from Transformed Self-Exemplars. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015. (page 1)

[46] Kwok-Wai Hung and Wan-Chi Siu. Real-Time Interpolation using Bilateral Filter for Image Zoom or Video Up-Scaling/Transcoding. In *International Conference on Consumer Electronics*, 2012. (page 1)

[47] Quan Huynh-Thu and Mohammed Ghanbari. Scope of Validity of PSNR in Image/Video Quality Assessment. *Electronics Letters*, 44(13):800–801, 2008. (page 58)

[48] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013. (page 44)

[49] Jeremy Jancsary, Sebastian Nowozin, and Carsten Rother. Loss-Specific Training of Non-Parametric Image Restoration Models: A New State of the Art. In *European Conference on Computer Vision*, 2012. (page 2, 18)

[50] Jeremy Jancsary, Sebastian Nowozin, and Carsten Rother. Regression Tree Fields - An Efficient, Non-Parametric Approach to Image Labeling Problems. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. (page 18)

[51] Liu Jing, Gan Zongliang, and Zhu Xiuchang. Directional Bicubic Interpolation - A New Method of Image Super-Resolution. In *International Congress on Image and Signal Processing*, 2013. (page 1)

[52] Ian Jolliffe. *Principal Component Analysis*. Wiley & Sons, 2002. (page 58)

[53] Chandrika Kamath and Erick Cantu-Paz. Creating ensembles of decision trees through sampling. In *Symposium on the Interface of Computing Science and Statistics*, 2001. (page 10)

[54] Robert Keys. Cubic Convolution Interpolation for Digital Image Processing. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29(6):1153–1160, 1981. (page 1, 57)

[55] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate Image Super-Resolution Using Very Deep Convolutional Networks. *The Computing Research Repository*, arXiv:1511(04587):1–8, 2015. (page 1, 58, 61, 62)

[56] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-Recursive Convolutional Network for Image Super-Resolution. *The Computing Research Repository*, arXiv:1511(04491):1–8, 2015. (page 1, 58, 61, 62)

[57] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep Neural Decision Forests. In *IEEE International Conference on Computer Vision*, 2015. (page 2, 18)

[58] Peter Kontschieder, Samuel Rota Bulò, Horst Bischof, and Marcello Pelillo. Structured Class-Labels in Random Forests for Semantic Image Labelling. In *IEEE International Conference on Computer Vision*, 2011. (page 7)

[59] Mu-Chu Lee, Wei-Lin Chiang, and Chih-Jen Lin. Fast Matrix-Vector Multiplications for Large-Scale Logistic Regression on Shared-Memory Systems. In *Industrial Conference on Data Mining*, 2015. (page 46, 67)

[60] Vincent Lepetit and Pascal Fua. Keypoint Recognition Using Random Forests and Random Ferns. In *Decision Forests for Computer Vision and Medical Image Analysis*, pages 111–124. Springer, 2013. (page 17)

[61] Andy Liaw and Matthew Wiener. Classification and Regression by Random Forest. *R News*, 2(3):18–22, 2002. (page 5)

[62] Moshe Lichman. UCI Machine Learning Repository. Website, 2013. University of California, Irvine, School of Information and Computer Sciences. `http://archive.ics.uci.edu/ml` last visited 11th March 2016. (page 25)

[63] J. Kent Martin. An Exact Probability Metric for Decision Tree Splitting and Stopping. *Machine Learning*, 28(2):257–291, 1997. (page 9)

[64] Erik Meijering and Michael Unser. A Note On Cubic Convolution Interpolation. *IEEE Transactions on Image Processing*, 12(8):477–479, 2003. (page 62)

[65] Ryszard Michalski, Jaime Carbonell, and Tom Mitchell. *Machine Learning: An Artificial Intelligence Approach*. Springer Science & Business Media, 2013. (page 5)

[66] Thomas Mitchell. *Machine Learning*. McGraw-Hill, Inc., 1997. (page 5)

[67] Douglas Montgomery, Elizabeth Peck, and Geoffrey Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, 2015. (page 29)

[68] Sebastian Nowozin, Carsten Rother, Shai Bagon, Toby Sharp, Bangpeng Yao, and Pushmeet Kohli. Decision Tree Fields. In *IEEE International Conference on Computer Vision*, 2011. (page 18)

[69] David Opitz and Richard Maclin. Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, 11(1):169–198, 1999. (page 10)

[70] Mustafa Ozuysal, Pascal Fua, and Vincent Lepetit. Fast Keypoint Recognition in ten Lines of Code. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007. (page 2, 17)

[71] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang. Super-Resolution Image Reconstruction: A Technical Overview. *IEEE Signal Processing Magazine*, 20(3):21–36, 2003. (page 1, 57)

[72] Anthony Parker, Robert Kenyon, and Donald Troxel. Comparison of Interpolating Methods for Image Resampling. *IEEE Transactions on Medical Imaging*, 2(1):31–39, 1983. (page 1)

[73] John Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in Large Margin Classifiers*, 10(3):61–74, 1999. (page 21)

[74] John Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986. (page 2, 5)

[75] John Ross Quinlan. Simplifying Decision Trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987. (page 2, 16)

[76] Russell Reed. Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993. (page 2, 16)

[77] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Global Refinement of Random Forest. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015. (page 2, 3, 15, 16, 19, 21, 25, 26, 27, 29, 38, 39, 46, 49, 55, 69)

[78] Alvin Rencher. *Methods of Multivariate Analysis*. John Wiley & Sons, 2003. (page 65)

[79] Alberto Suá Rez and James Lutsko. Globally Optimal Fuzzy Decision Trees for Classification and Regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999. (page 2, 18)

[80] Iain Richardson. *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*. John Wiley & Sons, 2004. (page 57)

[81] David Richmond, Dagmar Kainmueller, Michael Ying Yang, Eugene Myers, and Carsten Rother. Relating Cascaded Random Forests to Deep Convolutional Neural Networks for Semantic Segmentation. *The Computing Research Repository*, arXiv:1507(07583):1297–1311, 2015. (page 2, 18)

[82] Lior Rokach and Oded Maimon. Top-Down Induction of Decision Trees Classifiers - A Survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 35(4):476–487, 2005. (page 9)

[83] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific, 2014. (page 9, 10)

[84] Samuel Rota Bulò and Peter Kontschieder. Neural Decision Forests for Semantic Image Labelling. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. (page 9)

[85] Warren Sarle. Neural Network FAQ, part 2 of 7: Learning. Website, 1997. SAS - Statistical Analysis System, North Carolina State University. `ftp://ftp.sas.com/pub/neural/FAQ2.html` last visited 15th March 2016. (page 30)

[86] Robert Schapire. The Strength of Weak Learnability. *Machine Learning*, 5(2):197–227, 1990. (page 11, 17)

[87] Samuel Schulter. *Loss Minimization for Random Forests in Computer Vision*. PhD thesis, Graz University of Technology, 2015. (page 14)

[88] Samuel Schulter, Christian Leistner, and Horst Bischof. Fast and Accurate Image Upscaling with Super-Resolution Forests. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015. (page 2, 26, 57, 58, 60, 61, 63)

[89] Samuel Schulter, Christian Leistner, Paul Wohlhart, Peter Roth, and Horst Bischof. Alternating Regression Forests for Object Detection and Pose Estimation. In *IEEE International Conference on Computer Vision*, 2013. (page 2, 3, 11, 14, 25, 29, 31, 38, 39, 69)

[90] Samuel Schulter, Paul Wohlhart, Christian Leistner, Amir Saffari, Peter Roth, and Horst Bischof. Alternating Decision Forests. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013. (page 2, 3, 11, 12, 21, 25, 27, 69)

[91] Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision Jungles: Compact and Rich Models for Classification. In *Advances in Neural Information Processing Systems*, 2013. (page 2, 17)

[92] Andrey Tikhonov and Vasilii Arsenin. *Solutions of Ill-Posed Problems*. VH Winston, 1977. (page 15, 29, 45, 49, 56, 60)

[93] Radu Timofte, Vincent De, and Luc Van Gool. Anchored Neighborhood Regression for Fast Example-Based Super-Resolution. In *IEEE International Conference on Computer Vision*, 2013. (page 1, 57, 58, 60, 63)

[94] Radu Timofte, Vincent De Smet, and Luc Van Gool. A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. (page 1, 58, 60, 61, 62, 63)

[95] Radu Timofte, Rasmus Rothe, and Luc Van Gool. Seven Ways to Improve Example-Based Single Image Super Resolution. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. (page 63)

[96] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Incremental and Decremental Training for Linear Classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014. (page 46)

[97] Chih-Yuan Yang and Ming-Hsuan Yang. Fast Direct Super-Resolution by Simple Functions. In *IEEE International Conference on Computer Vision*, 2013. (page 1, 60)

[98] Jianchao Yang, John Wright, Thomas Huang, and Yi Ma. Image Super-Resolution as Sparse Representation of Raw Image Patches. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008. (page 1, 58)

[99] Jianchao Yang, John Wright, Thomas Huang, and Yi Ma. Image Super-Resolution via Sparse Representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, 2010. (page 1)

[100] Roman Zeyde, Michael Elad, and Matan Protter. On Single Image Scale-Up using Sparse-Representations. In *Curves and Surfaces*, pages 711–730. Springer, 2012. (page 1, 58, 59, 60, 61)