Alternating Regression Forests for Object Detection and Pose Estimation

Samuel Schulter[†] Christian Leistner[‡] Paul Wohlhart[†] Peter M. Roth[†] Horst Bischof[†]

[†]Graz University of Technology
Institute for Computer Graphics and Vision
{schulter,wohlhart,pmroth,bischof}@icg.tugraz.at

[‡]Microsoft Photogrammetry
Austria
christian.leistner@microsoft.com

Abstract

We present Alternating Regression Forests (ARFs)*, a novel regression algorithm that learns a Random Forest by optimizing a global loss function over all trees. This interrelates the information of single trees during the training phase and results in more accurate predictions. ARFs can minimize any differentiable regression loss without sacrificing the appealing properties of Random Forests, like low computational complexity during both, training and testing. Inspired by recent developments for classification [19], we derive a new algorithm capable of dealing with different* regression *loss functions, discuss its properties and investigate the relations to other methods like Boosted Trees.*

We evaluate ARFs on standard machine learning benchmarks, where we observe better generalization power compared to both standard Random Forests and Boosted Trees. Moreover, we apply the proposed regressor to two computer vision applications: object detection and head pose estimation from depth images. ARFs outperform the Random Forest baselines in both tasks, illustrating the importance of optimizing a common loss function for all trees.

1. Introduction

Random Forests (RFs), introduced in [1, 3], are ensembles of randomized decision trees. They are relatively easy to implement and have several appealing characteristics. For instance, RFs are fast to train and to evaluate, can handle high-dimensional input and output spaces, are robust to noise and flexible while being competitive with other learning algorithms. These benefits make them especially interesting for computer vision applications, where they have been applied to various tasks, *e.g.*, object detection [9] and tracking [11], semantic segmentation [20] or object categorization [16], to name but a few.

During training, each tree of the forest is grown independently by recursively splitting the labeled training data until some stopping criteria are fulfilled, in order to disambiguate the uncertainty of the predictions. The combination of the trees results in a strong and highly non-linear predictor, applicable to many different tasks like classification, regression or density estimation [5].

However, RFs minimize the uncertainty of the predictions only locally, *i.e.*, on the node level of independent trees. While this allows for easy parallelization, the global structure of the forest is not taken into account, which might be a potential drawback and is also in contrast to other well established machine learning algorithms like Boosting or Support Vector Machines. To overcome this issue, [19] presents Alternating Decision Forests (ADFs), a RF formulation for the *classification task*, which can incorporate different losses over all trees and reports improved performance in machine learning and computer vision experiments.

Recently, RFs have also been extensively used in computer vision to solve different *regression* tasks. In this case, class labels are real valued vectors and the target domain is disambiguated by growing the randomized trees. This mode of operation has led to state-of-the-art in many computer vision applications, such as, facial fiducial detection [4, 6] or pose estimation of heads [7, 8] and humans [10].

In this paper, we propose a novel Random Forest training procedure named *Alternating Regression Forests (ARFs)*, which, inspired by [19], globally optimizes differentiable regression loss functions. In particular, we formulate the training of Random Regression Forests as a general risk minimization problem and model the whole forest as a stage-wise classifier, akin to Gradient Boosting [12] and ADFs [19]. The iterative training procedure *alternates* between growing the forest by one level and evaluating the global loss for all training samples. In contrast to ADFs, where importance weights are assigned in a classification task, we calculate loss-specific "pseudo targets" with the residuals of the current state of the forest. These new target variables are optimized in the next iteration of the forest, which can be done in parallel for each tree, thus maintaining the low computational costs of RFs. These iterations continue until the same stopping criteria as in standard RFs are met.

After having reviewed the basics of Random Regression Forests in Sec. 3.1, we describe the ARF training procedure in detail in Sec. 3.3. Additionally, we discuss the properties of ARFs, the influence of different choices of the regression loss function and give some insights about the relations to other approaches like Boosted Trees or standard Random Regression Forests.

In Sec. 4, we perform three different experiments to assess the performance of ARFs. First, we show regression performance on several standard machine learning benchmarks and analyze the relevant parameters. Second, we extend Hough Forests [9] for object detection by applying the ARF principle to the regression split nodes and show improved results on three test sets. Finally, we integrate ARFs into the human head pose estimation system of Fanelli *et al*. [8] and show that our proposed regressor gives more accurate predictions.

2. Related Work

Recently, Random Forests (RFs) have been increasingly employed for difficult regression tasks in several computer vision applications, enabled by the non-parametric and highly non-linear structure of RFs. This allows for learning complex mappings from input to output spaces.

One recent successfull application is facial fiducial estimation from 2D images [6]. In this work, a Random Regression Forest is used to learn a mapping from small image patches to several facial feature points, like the mouth or eye corners. The prediction of the RF is conditioned on an estimated head pose and gives state-of-the-art results on standard benchmarks. Another task is human head pose estimation [7] from single depth images, where a Random Regression Forest is trained to estimate the position of the head (*e.g.*, the nose tip) in 3D and also the pose in Euler angles. Similar approaches have also been used for human pose estimation [10], where several body joint locations are regressed from single depth images.

RFs have also been employed in joint classification and regression tasks, where the objective function is formulated jointly for both tasks. Hough Forests (HFs) for object detection [9] or head detection and simultaneous pose estimation [8] are just two examples. In our experiments (see Sec. 4), we show that integrating our proposed regression algorithm into two of these applications gives better performance.

Random Regression Forests were also used in combination with Random Field models in [14]. The proposed Regression Tree Field builds on a Gaussian Random Field and its parameters are trained from image data with regression trees. In [13], this approach is extended and applied for image restoration tasks and yields state-of-the-art results. The paper also shows how the regression trees can be optimally trained for minimizing the Random Field energy.

In contrast to single tree optimization, we present a regression algorithm based on an ensemble of trees and show how this ensemble can be optimized with a global loss function. Our approach is inspired by Gradient Boosting [12] and Alternating Decision Forests (ADFs) [19], which presents a similar approach for classification tasks.

Boosted Trees for regression [12] are also related to our proposed regressor. Thus, we point out the similarities and differences to Alternating Regression Forests (ARFs) in a more detailed discussion (see Sec. 3.4). Moreover, in the experiments (see Sec. 4) we compare ARFs with RFs and also Boosted Trees.

3. Alternating Regression Forests

To derive *Alternating Regression Forests (ARFs)*, we first briefly review standard Random Regression Forests and show how a global loss function can be integrated into Random Forests (RFs) for classification [19]. Inspired by this concept, we then present our novel regressor in Sec. 3.3. We show how a Random Regression Forest can be trained in a risk minimization framework, akin to Gradient Boosting [12] and Alternating Decision Forests (ADFs) [19], however, without sacrificing the low computational costs of standard RFs. Finally, we discuss the properties of the new algorithm, the employed loss functions and point out the relations between ARFs, RFs and Gradient Boosting [12].

3.1. Random Forests for Regression

In general, for regression we are given labeled training samples $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathcal{X} = \mathbb{R}^M$ and $\mathbf{y}_i \in \mathcal{Y} = \mathbb{R}^K$, sampled from a joint probability distribution $q(\mathbf{x}, \mathbf{y})$. The dimensions of the input and output variables are given as M and K, respectively, and N is the total number of training examples.

Random Regression Forests typically describe a non-linear mapping $\mathcal{M} : \mathbb{R}^M \to \mathbb{R}^K$, where an example $\mathbf{x}$ is mapped to a target prediction $\mathbf{y}$. This mapping is learned by an ensemble of binary decision trees $\{\mathcal{T}_t\}_{t=1}^T$ (T being the number of trees in the ensemble), each trained on a subset of the training data (*c.f.* bagging [3]). A single decision tree $\mathcal{T}_t$ recursively splits the given training data into two partitions, such that the uncertainty of the target variables in the resulting subsets is minimized.

In particular, each node in a tree randomly samples a set of splitting functions $\phi(\mathbf{x})$, each separating the data into two disjoint subsets, L and R, respectively. All splitting functions are then evaluated by measuring the information gain

$$I = H(L \cup R) - \frac{|L|}{|L| + |R|} H(L) - \frac{|R|}{|L| + |R|} H(R) \ , \quad (1)$$

where $H(\cdot)$ is the entropy over the target labels and $|\cdot|$ denotes the size of a set. For classification, $H(\cdot)$ is the discrete entropy. For regression tasks, the differential entropy

$$h(q) = \int_{\mathcal{Y}} q(\mathbf{y}|\mathbf{x}) \log(q(\mathbf{y}|\mathbf{x}))\, \mathrm{d}\mathbf{y} \quad (2)$$

over continuous outputs can be employed, where $q(\mathbf{y}|\mathbf{x})$ denotes the conditional probability of a target variable given the input sample. Assuming $q(\cdot,\cdot)$ to be a Gaussian distribution and having only a finite set S of samples, the differential entropy can be written in closed form as

$$h_{\text{Gauss}}(S) = \frac{K}{2}(1 - \log(2\pi)) + \frac{1}{2}\log(\det(\Sigma_S)) \ , \quad (3)$$

where $\det(\Sigma_S)$ is the determinant of the estimated covariance matrix of the target variables in S.

The splitting function $\phi^*(\mathbf{x})$ giving the highest information gain is fixed for this node and the data is separated accordingly into the subsets L and R. This procedure of splitting the data continues until some stopping criteria are reached, *e.g.*, a maximum tree depth or a minimum number of samples left in the splitting node. If one of these criteria is fulfilled, a leaf node is created by estimating a density model $p(\mathbf{y})$ from all samples falling in this leaf, in order to predict the target value.

The most simple way to estimate the probability distribution $p(\mathbf{y})$ is by calculating the sample mean over the target variables reaching a leaf node. However, there are also more elaborate variants like fitting a Gaussian (including a covariance estimation), kernel density estimation or non-parametric densities [18].

3.2. Optimizing a Global Loss

As described in the previous section, RFs only decide locally on the node level how the data is further split, without considering the state of the whole classifier. While this allows for easy parallelization and thus leads to low computational costs, the learning procedure is not globally controlled by an appropriate loss function.

The recently proposed ADFs [19] tackle this issue by reformulating the training phase of RFs as a stage-wise classifier. The trees are trained breadth-first up to depth D_{max} and each level of depth d corresponds to a single stage. After training stage $d-1$, the current prediction of each training sample $\mathbf{x}_i$ can be calculated by evaluating the current state of the RF, *i.e.*, the forest trained up to stage $d-1$. Then, similar to Gradient Boosting [12], the gradient of the loss for each training sample can be calculated and exploited to optimize a global loss function over the whole Random Forest in the next stage d.

For classification tasks, this can be achieved by assigning each training sample $\mathbf{x}_i$ a weight w_i^d for training stage d, *i.e.*, depth d of the forest. The weights of the training sample are derived from the corresponding gradient of the loss function and have a straight-forward interpretation: Training samples that are hard to classify get assigned a high weight and "easy" samples a low weight, which forces the classifier of the current stage d to concentrate on "hard" samples. Thus, ADFs optimize a global loss over all trees by keeping a weight distribution over the training samples, which gets updated in each stage d according to the given loss function and the current state of the classifier. The classifier *alternates* between weight updates and training of a single stage. Contrary to [17, 15], where only nodes within a single tree are entangled, ADFs entangle all trees in the forest, as each local node split depends on the output of all other trees.

In the following, we exploit these ideas to develop Alternating Regression Forests, which can optimize any differentiable, global *regression* loss.

3.3. Training Alternating Regression Forest

We now formulate *Alternating Regression Forests* as a stage-wise risk minimization problem for regression tasks. The general learning objective can be written as loss minimization problem:

$$\arg\min_{\Theta} \sum_{\{\mathbf{x}_i,\mathbf{y}_i\}} l(\mathbf{y}_i; F_{D_{\max}}(\mathbf{x}_i;\Theta)) \ , \quad (4)$$

where $l(\cdot)$ is a differentiable loss function and $F_{D_{\max}} = \sum_{d=0}^{D_{\max}} f_d(\mathbf{x},\Theta_d)$ denotes the random forest with a tree depth of $D_{\max}$ as an additive classifier, akin to Boosting [12] and ADFs [19]. Please note that the summation is defined over the stages, *i.e.*, depths, not over the trees. The parameters Θ_d correspond to the parameters of all splitting functions $\phi(\mathbf{x})$ in depth d of the whole forest. To keep the notation uncluttered, Θ denotes all splitting functions of the corresponding forest $F(\cdot)$. The minimization problem (4) can be rewritten as a greedy stage-wise optimization

$$\arg\min_{\Theta_d} \sum_{\{\mathbf{x}_i,\mathbf{y}_i\}} l(\mathbf{y}_i; F_{d-1}(\mathbf{x}_i;\Theta) + f_d(\mathbf{x}_i;\Theta_d)) \ , \quad (5)$$

where $F_{d-1}(\mathbf{x};\Theta)$ is the classifier trained up to stage $d-1$ (*i.e.*, the Random Forest of depth $d-1$) and Θ_d are the parameters to be optimized in the current stage.

We start with an initial classifer $F_0 = f_0(\mathbf{x};\Theta_0)$, which corresponds to the T root nodes, and each iteration d adds a new level of depth to the forest. A classifier $F_{d-1}(\mathbf{x}_i;\Theta)$ trained up to iteration $d-1$ gives predictions for all training samples $\mathbf{x}_i$, according to the accumulated probability distributions $p(\mathbf{y})$ stored in the corresponding nodes. These predictions yield a corresponding loss, which should be minimized (*c.f.* Eq. (5)). In contrast to ADFs [19], where importance weights for each sample are calculated to approximate the optimization of a global loss, we employ a more elaborate way for training the next stage $f_d(\mathbf{x})$. Similar to

Gradient Boosting [12], we calculate so-called "pseudo targets"

$$-\mathbf{g}_i^d = -\left[\frac{\partial l(\mathbf{y}_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(x)=F_{d-1}(\mathbf{x})} \tag{6}$$

for each sample i, which correlate with the negative gradient of the loss function w.r.t. the output of the current classifier. These pseudo targets form an intermediate training set $\{\mathbf{x}_i, -\mathbf{g}_i^d\}$ for the current iteration. This data is then used to train $f_d(\mathbf{x})$, *i.e.*, the depth d of the forest, according to the information gain criterion, Eq. (1) (see Fig. 1).

Please note that training depth d of the forest corresponds to transforming the leaf nodes in $F_{d-1}(\mathbf{x})$ into splitting nodes and creating new leaf nodes in depth d, which make predictions about the "pseudo targets" $-\mathbf{g}_i^d$. Unlike in Boosting (with trees or decision stumps as weak learners), where the path of any sample $\mathbf{x}$ is unknown beforehand, in ARFs, this path is always fixed as we have a hierarchical classifier structure. Thus, we can immediately add the target distribution $p_{\text{pa}}(\mathbf{y})$ of the parent node, *i.e.*, the new split node in depth $d-1$, to the current target distributions $p_{\text{ch}}(\mathbf{y})$ of the new child nodes in depth d. After having updated $p_{\text{ch}}(\mathbf{y})$ for iteration d, we can set the prediction of p_{pa} to 0 and continue the training of ARFs with iteration $d+1$. The intermediate classifier $F_d(\mathbf{x})$ can thus be used like a standard RF for making predictions. In Algorithm 1, we summarize the complete training procedure of ARFs.

During inference, RFs, Boosting with decision stumps/trees as weak learner, and ARFs are exactly the same, except for the aggregation of the predictions. A given test sample $\mathbf{x}$ is routed through all trees $\mathcal{T}_t$ and thus ends up in T leaf nodes, each storing its estimated target variable distribution $p(\mathbf{y})$. These distributions are either averaged (RFs and ARFs) or accumulated (Boosting) over the ensemble to yield the final prediction. Nevertheless, the computational complexity of all methods during testing is the same.

Algorithm 1 Training of *Alternating Regression Forests*

```
Require: Labeled training set $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N \in \mathcal{X} \times \mathcal{Y}$
Require: Number of trees $T$, maximum tree depth $D_{\max}$
1: Init $F_0 = f_0(\mathbf{x})$ as $T$ root nodes with $p(\mathbf{y})$
2: for $d$ from 1 to $D_{\max}$ do
3:     Check stopping criteria for all nodes in depth $d$
4:     Calculate "pseudo targets" $-g_i^d$ (Eq. (6))
5:     Find $\Theta_d$, i.e., $\phi^*(\mathbf{x})$ for "pseudo targets" (Eq. (1))
6:     Calculate $p_{\text{child}}(\mathbf{y})$ in (intermediate) leaf nodes
7:     Add $p_{\text{parent}}(\mathbf{y})$ to corresponding $p_{\text{child}}(\mathbf{y})$
8:     $F_d(\mathbf{x}; \Theta) = F_{d-1}(\mathbf{x}; \Theta) + f_d(\mathbf{x}; \Theta_d)$
9: end for
```

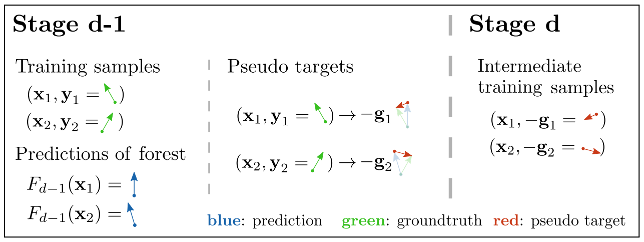

Stage d-1
Stage d
Training samples
$(\mathbf{x}_1, \mathbf{y}_1 =)$
$(\mathbf{x}_2, \mathbf{y}_2 =)$
Predictions of forest
$F_{d-1}(\mathbf{x}_1) =$
$F_{d-1}(\mathbf{x}_2) =$
Pseudo targets
$(\mathbf{x}_1, \mathbf{y}_1 =) \rightarrow -\mathbf{g}_1$
$(\mathbf{x}_2, \mathbf{y}_2 =) \rightarrow -\mathbf{g}_2$
Intermediate training samples
$(\mathbf{x}_1, -\mathbf{g}_1 =)$
$(\mathbf{x}_2, -\mathbf{g}_2 =)$
blue: prediction
green: groundtruth
red: pseudo target

Figure 1: Illustration of the "pseudo targets" in ARFs used to train the intermediate stages.

3.4. Discussion

In the following, we specify the loss functions used in our implementation and briefly discuss the properties of the algorithm compared to related approaches.

In general, any differentiable loss function can be integrated into ARFs, however, we use three of the most common regression losses, the Squared, the Absolute, and the Huber loss, well known from robust statistics [12]. The Squared loss is the most restrictive one, giving relatively high penalty to samples that are incorrectly regressed, compared to, *e.g.*, the Absolute loss. The Huber loss [12] can be adapted with the parameter δ, resulting in different shapes of the loss. It behaves like a Squared loss for residuals below δ and like an Absolute loss for residuals larger than δ.

We further want to discuss the relations between ARFs and Boosting (with decision stumps/trees). Although ARFs can be equivalently formulated as Gradient Boosting, there are two main differences. First, while Boosting pools from the same space of weak learners in all iterations, *i.e.*, the number of splitting functions is the same, for ARFs, this space increases over time as the number of split functions to be optimized increases in each iteration. This property allows ARFs to be easily trained in parallel. Second, Boosting trains a weak learner from scratch in each iteration, *e.g.*, a new regression tree. Contrary, in ARFs, a weak learner corresponds to the splitting functions in a single depth d. This implies that for $d > 0$ the training samples are conditioned on the structure of the trees up to depth $d-1$, which eases the task for the weak learner in the current iteration.

Finally, we also discuss the computational costs of ARFs, Boosting, and RFs. In Boosting, the overall classifier corresponds to a flat additive combination of the weak learners, *e.g.*, decision or regression stumps/trees. Thus, each weak learner has to be trained consecutively, which may take a long time to get a desired model complexity for the task at hand. In contrast, ARFs regard each depth of the forest as a single weak leaner, *i.e.*, several different splitting functions $\phi(\mathbf{x})$, thus being able to parallelize them. Standard RFs can also be trained easily in parallel, which also makes them fast during both training and testing, however, without optimizing a global loss.

4. Experiments

To assess the performance of the proposed *Alternating Regression Forests (ARFs)*, we performed several experiments. We evaluate ARFs on 19 machine learning benchmarks and give a comparison to Boosted Trees (BTs) and standard Random Forests (RFs). Moreover, we present two different computer vision applications, namely object detection and human head pose estimation, where we integrate the ARF principle.

4.1. Results on Machine Learning Data

We conduct a standard regression experiment on a set of 19 machine learning benchmarks from different sources (UCI, StatLib, Delve)[1] to evaluate the difference between standard Random Regression Forests (RFs), Boosted Trees (BTs), and *Alternating Regression Forests (ARFs)*. For BTs and ARFs, we evaluate the loss functions presented above (Squared, Absolute, and Huber).

As most benchmarks do not provide specific train-test splits, we split the given data into 60% training and 40% testing data. To provide statistically fair results, we average them by repeating the following procedure 5 times. We first build a random train-test split (unless an explicit split is given) with the above defined ratio and then, for each split, we train and test all methods 4 times in order to further decrease statistical uncertainties due to the random tree growing schemes. This procedure results in a total of 20 averaging runs per method and data set. We measure the performance as the Root Mean Squared Error (RMSE).

The parameters of all trees for the different methods (RFs, BTs, and ARFs) are set equally: we used 50 trees with a maximum depth of 15, however, tree growing also stops if the sample size in a node is below 10. We use $\sqrt{D}$ random tests [3] (D being the input feature dimensionality) and 20 randomly chosen thresholds. For BTs, the number of trees always corresponds to the number of iterations/weak learners, thus having the same model complexity as the other methods.

In Tab. 1, we present our results as mean and standard deviation of RMSE values. Bold faced values indicate the best performing method for a single data set. We also highlight statistically significantly better results compared to RFs according to a Student t-test. As can be seen from the table, ARF-based methods win in all but 2 data sets, in most cases significantly. The performance difference between the loss functions for both ARFs and BTs is rather minor. Throughout all experiments we fix the parameter δ for the Huber loss to 0.3.

While the goal in this first experiment was to evaluate all methods with the same model complexity (*i.e.*, same number of trees and maximum depth), we further compare the performance for different complexities in a second experiment. We choose the *autompg* data set, fixed the loss for BTs and ARFs to be the Squared loss, and varied the number of trees, *i.e.*, weak learners, between 10 and 500 for two fixed maximum depth values, 5 and 15. Again, we averaged the results for each parameter combination over several train-test splits. The results are illustrated in Fig. 2. We make three observations: First, a larger amount of weak learners is important for BTs (both plots), which, however, also implicates a longer training time compared to RFs and ARFs, as no parallelization is possible. Second, BTs can handle shallow trees as weak learners much better than RFs or ARFs (see Fig. 2a). Finally, the performance of both BTs and ARFs is similar with the appropriate parameter settings (RMSE values of 2.85 and 2.82, respectively), while RFs, which do not optimize a global loss function, lag behind (RMSE $= 2.98$).

Finally, we also evaluate the overall training and testing time. Averaged over all data sets, ARFs and RFs have more or less the same computational costs, while BTs take approximately 5 times longer. Please note that these values correspond to 50 weak learners and would even be more distinctive if the number of weak learners is increased.

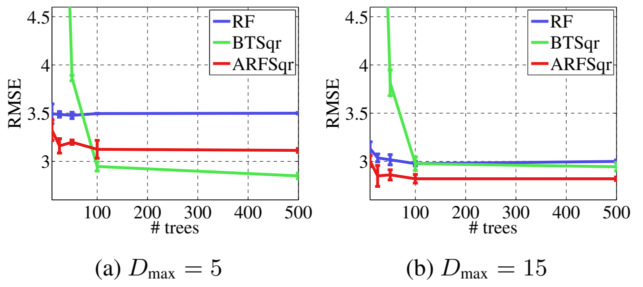


Figure 2: Parameter evaluation of RFs, BTs and ARFs for the experiment conducted on *autompg*. We vary the number of trees for two choices of the maximum tree depth $D_{\max}$.

4.2. Application I: Object Detection

In this first vision experiment, we integrate the ARF principle in the Hough Forests (HFs) framework [9] for object detection. We first give a brief review of HFs and describe our modifications to the regression nodes, before we present our results on three popular object detection benchmarks.

Hough Forests [9] are a detection model based on RFs, which uses small appearance patches $\mathcal{P}_i$ to describe the desired object. Foreground patches store an offset vector $\mathbf{d}_i$ to the object's center, while background patches do not. The appearance of each patch is encoded in several feature channels that include the LAB color channels or gradient information, among others. During training, a Random Forest is built to disambiguate both the class uncertainty of all patches and the regression variance of foreground patches.

[1]A collection of the data sets can be found at `http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html`

	Scale	*RF*	*BTAbs*	*BTSqr*	*BTHub*	*ARFAbs*	*ARFSqr*	*ARFHub*
abalone		2.44 ± 0.02	2.87 ± 0.18	2.95 ± 0.15	2.84 ± 0.14	2.45 ± 0.02	$\mathbf{2.44 \pm 0.03}$	2.45 ± 0.02
ailerons	10^{-4}	2.15 ± 0.02	1.97 ± 0.03	1.97 ± 0.02	1.97 ± 0.02	$\mathbf{1.78 \pm 0.01}$	1.78 ± 0.01	1.78 ± 0.01
autompg		3.03 ± 0.04	3.44 ± 0.21	3.40 ± 0.22	3.47 ± 0.17	$\mathbf{2.89 \pm 0.06}$	2.89 ± 0.05	2.90 ± 0.04
breastcancer		$\mathbf{35.3 \pm 0.1}$	38.9 ± 1.0	38.9 ± 1.6	38.9 ± 1.5	35.3 ± 0.1	35.3 ± 0.1	35.3 ± 0.2
cartdelve		1.12 ± 0.01	1.29 ± 0.01	1.29 ± 0.01	1.29 ± 0.01	1.02 ± 0.00	$\mathbf{1.02 \pm 0.00}$	1.02 ± 0.00
cpuact		2.89 ± 0.01	3.34 ± 0.43	3.20 ± 0.20	3.11 ± 0.15	2.63 ± 0.04	2.63 ± 0.03	$\mathbf{2.62 \pm 0.03}$
cpusmall		3.20 ± 0.01	3.42 ± 0.16	3.42 ± 0.11	3.41 ± 0.15	2.94 ± 0.03	$\mathbf{2.94 \pm 0.03}$	2.94 ± 0.02
deltaailerons	10^{-4}	1.71 ± 0.00	1.91 ± 0.02	1.89 ± 0.04	1.90 ± 0.02	1.68 ± 0.00	$\mathbf{1.68 \pm 0.00}$	1.68 ± 0.00
deltaelevators	10^{-3}	1.48 ± 0.00	1.57 ± 0.01	1.57 ± 0.01	1.57 ± 0.02	$\mathbf{1.46 \pm 0.00}$	1.46 ± 0.00	1.46 ± 0.00
diabetes		$.725 \pm .018$	$.790 \pm .065$	$.797 \pm .057$	$.777 \pm .083$	$.710 \pm .011$	$.711 \pm .010$	$\mathbf{.709 \pm .008}$
elevators	10^{-3}	3.64 ± 0.03	3.41 ± 0.09	3.37 ± 0.06	3.39 ± 0.07	$\mathbf{2.98 \pm 0.04}$	2.98 ± 0.02	2.98 ± 0.04
friedmandelve		1.66 ± 0.01	1.65 ± 0.04	1.64 ± 0.03	1.65 ± 0.04	1.10 ± 0.00	$\mathbf{1.10 \pm 0.01}$	1.11 ± 0.00
housing		3.46 ± 0.09	3.94 ± 0.28	3.94 ± 0.26	4.03 ± 0.29	$\mathbf{3.19 \pm 0.07}$	3.21 ± 0.08	3.22 ± 0.06
kinematics		$.166 \pm .001$	$.174 \pm .003$	$.173 \pm .003$	$.172 \pm .004$	$\mathbf{.125 \pm .001}$	$.126 \pm .002$	$.125 \pm .001$
pol		17.7 ± 1.1	10.7 ± 0.6	10.6 ± 0.6	10.7 ± 0.5	$\mathbf{9.72 \pm 0.24}$	9.77 ± 0.22	9.86 ± 0.18
pyrimidines		$.086 \pm .003$	$.094 \pm .009$	$.098 \pm .016$	$.097 \pm .011$	$\mathbf{.074 \pm .001}$	$.075 \pm .003$	$.075 \pm .003$
servo		$.698 \pm .025$	$.669 \pm .098$	$.672 \pm .078$	$\mathbf{.657 \pm .084}$	$.667 \pm .010$	$.659 \pm .026$	$.670 \pm .021$
stockairplane		1.03 ± 0.02	1.11 ± 0.12	1.15 ± 0.09	1.09 ± 0.07	1.01 ± 0.03	1.01 ± 0.03	$\mathbf{.996 \pm .029}$
triazines		$.133 \pm .002$	$.134 \pm .014$	$.134 \pm .008$	$.135 \pm .009$	$\mathbf{.129 \pm .002}$	$.129 \pm .001$	$.129 \pm .003$
Wins		1	0	0	1	9	5	3

Table 1: Machine learning results for the compared methods (RFs, BTs, ARFs) with different loss functions (Absolute, Squared, Huber) for BTs and ARFs on standard regression benchmarks. The results are presented as RMSE values averaged over several runs (mean and standard deviation is given). Bold values mark the best performing method for each data set and highlighted results indicate a significant improvement over RFs.

Each split node is randomly assigned to be either a classification or a regression node. Classification nodes are similarly designed as in standard RFs and extended to optimize a global loss in [19]. In the following, we further extend HFs in the regression nodes.

Regression nodes in HFs follow the simple Reduction-In-Variance approach [9], which measures the uncertainty of the node predictions as

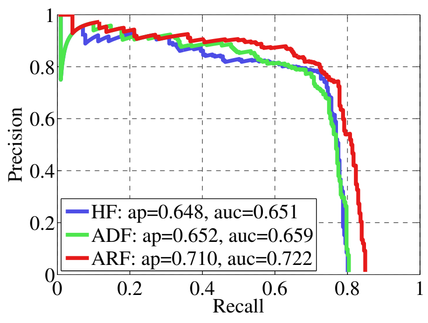
$$H(S) = \frac{1}{|S|} \sum_{i=1}^{|S|} \|\mathbf{d}_i - \bar{\mathbf{d}}\|_2^2 \, , \tag{7}$$

where S defines the set of positive patches with offset vectors $\mathbf{d}_i$, and $\bar{\mathbf{d}}$ is the mean offset vector of S. The ARF principle is integrated in the Hough Forest framework as described in Sec. 3.3 for regression split nodes. The trees are grown depth by depth and after each iteration the "pseudo targets" are calculated via the given loss and the current prediction of the forest.

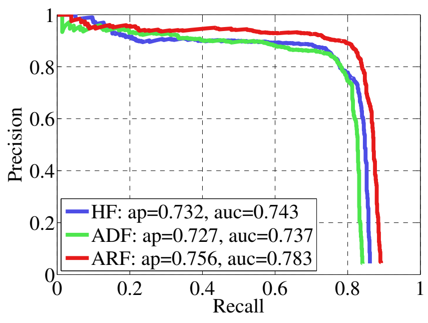
HFs stop growing trees if either a maximum depth is reached or less than 20 samples are available in a node. Then, leaf nodes are created that store a class probability, *i.e.*, a fore- and background probability, and votes for object centers. While standard HFs store all offset vectors $\mathbf{d}_i$ reaching a leaf node, we follow a different approach [10], where only modes of the distribution of offset vectors are stored. Storing only modes instead of all offset vectors gives similar voting accuracy, but reduces the computation time during inference vastly. As we already calculate the mean of the offset vectors for the residuals during tree growing, we simply stick with these estimates for the final offset vectors in the leaf nodes. We increase the maximum tree depth to 30, in order to handle occurring multi modal distributions.

During inference, each patch in a test image is scanned and routed to the leaf nodes, which vote for tentative object centers in a Hough space. All votes are accumulated and the resulting maxima then indicate object detections.

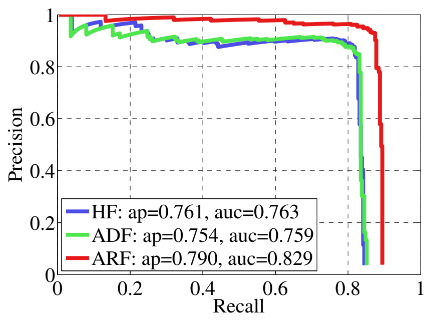
Experimental Setup: We use three object detection data sets commonly employed for evaluating HFs, namely the *TUD-pedestrian*, *TUD-crossing* and *TUD-campus* benchmarks [2]. Only the *TUD-pedestrian* data set provides training data in the form of 400 bounding-box annotated images. The test data consists of 250, 201, and 72 images for the three data sets, respectively. In this experiment, we directly evaluate the influence of the global loss in the regression nodes of HFs. We thus compare standard Hough Forests [9] (HF) with our proposed modifications as described above (ARFs). Further, we also include the influence of a global loss for the classification nodes [19] (ADF). For a fair comparison, we endow all methods with 10 trees, each having a maximum depth of 30, and 20000 random tests per node. We use a training data set consisting of 16000 foreground (randomly extracted within the bounding boxes) and 16000 background patches. To get statistically fair results (patch extraction and tree growing is random), we repeat the train-test procedure 10 times. During each round, we extract the same set of patches for all methods. For ARFs, we use the Squared loss throughout all experiments, as this loss consistently gave the best results.



(a)



(b)



(c)

Figure 3: Precision-Recall curves of HFs [9], ADFs [19] and our proposed ARF on (a) the *TUD-pedestrian*, (b) the *TUD-crossing* and (c) the *TUD-campus* data sets.

Results: We present our results as precision recall curves in Fig. 3. As can be seen in the plots, for this setup, Alternating Decision Forests (ADFs) only improve on one data set over HFs. However, optimizing a global regression loss during the training procedure, *i.e.*, ARFs, boosts the performance on all three data sets. We get a gain of 7.2%, 5.2% and 6.4% in terms of the area-under-curve (auc) score over HFs (we also provide average precision (ap) scores). We also note that the combination of ADFs and ARFs did not further improve the results.

4.3. Application II: Head Pose Estimation

Our second computer vision application for ARFs is human head pose estimation from depth images, where we follow the experimental setup of Fanelli *et al*. [8]. The goal is to train a joint classification and regression forest on patches from labeled depth images, in order to detect the head and to estimate the pose from unseen depth images. All training patches store a class label (*i.e.*, being a face or non-face patch) and only positive patches store target regression values (*i.e.*, the head center position relative to the patch in 3D coordinates and the pose in Euler angles).

Thus, the Random Forest is very similar to standard HFs [9] with a few exceptions, like a different form of the splitting functions $\phi(\mathbf{x})$ (Haar-like features), a larger patch size of 100px, or a slightly different entropy measure $H(\cdot)$. Due to lack of space, we refer the reader to [8] for more details. Each leaf node stores a foreground probability p_{fg}, the target prediction $p(\mathbf{y})$ (mean of all training sample targets), and the variance σ^2 of those targets. We can extend the training procedure of [8] with our proposed regressor in the same way as for HFs in Sec. 4.2, as the differences between [8] and [9] do not directly affect our modified training procedure.

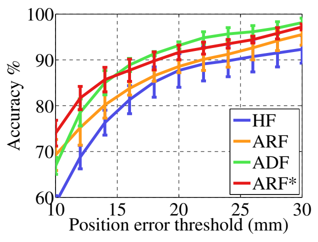
During testing, patches from the depth image are extracted on a regular grid and routed to the corresponding leaf nodes in all trees. Each leaf node having a foreground probability $p_{\text{fg}} = 1$ and a variance $\sigma^2 < \sigma^2_{\text{max}}$ are selected to vote for a head center and pose [8]. Using a meanshift variant, a single mode is found from all votes for a single depth image.

Experimental Setup: For all experiments, we use the data set from [8], which is publicly available and contains more than 15K frames of depth images capturing human faces. It is split into several sequences, each showing one person. As in [8], we split the data into 18 training sequences and 2 testing sequences. Groundtruth is given as the head center position in 3D and the pose in Euler angles for each frame. Like in the previous experiment, we compare HFs, modified for this regression task [8], ADFs, and ARFs. In order to directly compare the different training procedures, we endow all RFs with the same parameters from [8], *i.e.*, we use 7 trees, a maximum depth of 15 and 20000 random tests per node.

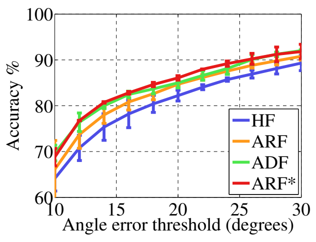
Results: Following [8], we also present our results as the percentage of correctly predicted frames over different success thresholds for both, position and pose regression of the head, see Fig. 4. For a fair comparison, we use our own implementation of all competing methods on the same code basis. However, as some details of the experimental setup in [8] (train-test split, *etc*.) are not fully specified, we get slightly different results compared to those reported in [8]. Nevertheless, we can see from the plots that both approaches optimizing a global loss (ADFs and ARFs) consistently improve over the HFs baseline. Although this is a regression task, we can observe that ADFs (classification nodes) significantly improve over HFs. A reason for this might be that ADFs produce cleaner leaf nodes (*i.e.*, leafs having $p_{\text{fg}} = 1$), which leads to more effective voting nodes that improve the overall result. Furthermore, in this experiment we also evaluate the performance of the combination of ADFs and ARFs, denoted ARF*. We can observe that ARF* gives the best results for the more important tighter success thresholds (*i.e.*, 10mm) in the head position regression (see Fig. 4a), and also gives good results for the pose estimation (see Fig. 4b).

Method	X	Y	Z	**Position**	Yaw	Pitch	Roll	**Angle**
HF [8]	6.03 ± 7.49	8.63 ± 13.45	4.39 ± 6.86	12.99 ± 15.80	3.79 ± 3.74	9.27 ± 13.22	6.62 ± 9.07	13.48 ± 15.37
ARF	5.84 ± 6.87	4.93 ± 5.34	4.28 ± 4.25	9.84 ± 8.72	3.67 ± 3.39	9.17 ± 11.95	4.83 ± 4.94	12.14 ± 12.39
ADF [19]	4.77 ± 5.50	5.88 ± 3.85	3.53 ± 3.69	9.50 ± 6.18	3.54 ± 3.33	7.87 ± 11.55	5.39 ± 4.79	11.48 ± 11.80
*ARF**	4.64 ± 5.29	4.91 ± 5.88	4.00 ± 4.18	8.68 ± 8.21	3.52 ± 3.25	8.18 ± 11.01	4.77 ± 4.52	11.17 ± 11.38

Table 2: Raw regression errors (mean and standard deviation) of HF [8], ARF, ADF [19] and ARF* in mm for X, Y, Z, and Position and in degree for Yaw, Pitch, Roll and Angle.



(a)



(b)

Figure 4: Frame accuracy of the competing methods (HF, ADF, ARF and ARF*) for different success thresholds of (a) the head position in mm and (b) the head pose in degree.

To get a better insight in the accuracies of all methods, we also give the raw errors for all 6 variables and the aggregated head position and head pose (angle) errors in Tab. 2. Again, we observe that all methods optimizing a global loss consistently outperform the baseline [8] in this task.

5. Conclusion

We presented *Alternating Regression Forests*, a novel Random Forest training procedure for regression tasks, which, in contrast to standard Random Regression Forests, optimizes any differentiable global loss function without sacrificing the computational benefits of Random Forests. ARFs are easy to implement and can be exchanged with standard Random Regression Forests without great efforts. This novel regressor gives better performance on machine learning benchmarks compared to Random Forests and Boosted Trees. Furthermore, we also integrated our ideas into two computer vision applications (object detection with Hough Forests and pose estimation from depth images). In both cases, ARFs could beat the baselines, illustrating the benefits of optimizing a global loss during training.

Acknowledgements: This work was supported by the Austrian Research Promotion Agency (FFG) projects FACTS (832045) and DIANGO (840824).

References

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

[2] M. Andriluka, S. Roth, and B. Schiele. People-Tracking-by-Detection and People-Detection-by-Tracking. In *CVPR*, 2008.

[3] L. Breiman. Random Forests. *ML*, 45(1):5–32, 2001.

[4] T. F. Cootes, M. Ionita, C. Lindner, and P. Sauer. Robust and Accurate Shape Model Fitting using Random Forest Regression Voting. In *ECCV*, 2012.

[5] A. Criminsi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013.

[6] M. Dantone, J. Gall, G. Fanelli, and L. v. Gool. Real-time Facial Feature Detection using Conditional Regression Forests. In *CVPR*, 2012.

[7] G. Fanelli, J. Gall, and L. v. Gool. Real Time Head Pose Estimation with Random Regression Forests. In *CVPR*, 2011.

[8] G. Fanelli, T. Weise, J. Gall, and L. v. Gool. Real Time Head Pose Estimation from Consumer Depth Cameras. In *DAGM*, 2011.

[9] J. Gall and V. Lempitsky. Class-Specific Hough Forests for Object Detection. In *CVPR*, 2009.

[10] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon. Efficient Regression of General-Activity Human Poses from Depth Images. In *ICCV*, 2011.

[11] M. Godec, P. M. Roth, and H. Bischof. Hough-based Tracking of Non-rigid Objects. In *ICCV*, 2011.

[12] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[13] J. Jancsary, S. Nowozin, and C. Rother. Loss-Specific Training of Non-Parametric Image Restoration Models: A New State of the Art. In *ECCV*, 2012.

[14] J. Jancsary, S. Nowozin, T. Sharp, and C. Rother. Regression Tree Fields. In *CVPR*, 2012.

[15] P. Kontschieder, S. Rota Bulò, A. Criminisi, P. Kohli, M. Pelillo, and H. Bischof. Context-Sensitive Decision Forests for Object Detection. In *NIPS*, 2012.

[16] C. Leistner, A. Saffari, and H. Bischof. Semi-Supervised Random Forests. In *ICCV*, 2009.

[17] A. Montillo, J. Shotton, J. Winn, J. E. Iglesias, D. Metaxas, and A. Criminisi. Entangled Decision Forests and their Application for Semantic Segmentation of CT Images. In *IPMI*, 2011.

[18] S. Nowozin. Improved Information Gain Estimates for Decision Tree Induction. In *ICML*, 2012.

[19] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. Alternating Decision Forests. In *CVPR*, 2013.

[20] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, 2008.