FASTER TREES: STRATEGIES FOR ACCELERATED TRAINING AND PREDICTION OF RANDOM FORESTS FOR CLASSIFICATION OF POLSAR IMAGES

R. Hänsch*, O. Hellwich

Computer Vision & Remote Sensing, Technische Universität Berlin, Germany - (r.haensch, olaf.hellwich)@tu-berlin.de

Commission III, WG 1

KEY WORDS: Random Forest, PolSAR, Information Extraction, Classification, Computation Time

ABSTRACT:

Random Forests have continuously proven to be one of the most accurate, robust, as well as efficient methods for the supervised classification of images in general and polarimetric synthetic aperture radar data in particular. While the majority of previous work focus on improving classification accuracy, we aim for accelerating the training of the classifier as well as its usage during prediction while maintaining its accuracy. Unlike other approaches we mainly consider algorithmic changes to stay as much as possible independent of platform and programming language. The final model achieves an approximately 60 times faster training and a 500 times faster prediction, while the accuracy is only marginally decreased by roughly 1%.

1. INTRODUCTION

Polarimetric Synthetic Aperture Radar (PolSAR) measures amplitude and phase of the echo of a microwave pulse backscattered at the ground by using different polarisations during transmission and/or reception. In contrast to optical and hyperspectral sensors, it operates independently from daylight and the electromagnetic properties of the used microwave allow to penetrate clouds, dust, and to some degree even vegetation. These properties have lead to the launch of many modern sensors providing PolSAR data, i.e. images containing complex-valued vectors in each pixel, in ever increasing spatial, spectral, and temporal resolution.

As the measured echoes depend on several surface characteristics such as moisture, roughness, and object geometry, PolSAR images are well suited for the (automatic) generation of semantic maps of land use/cover by pixelwise classification. This typically involves the extraction of hand-crafted image features and feeding them as input to a supervised machine learning method that fits a generic model to the training data. The work in (Hänsch and Hellwich, 2017) proposed to apply patch-based Random Forests (RFs) directly to PolSAR images without any computation of predefined features. Instead, the tests of the internal nodes of the decision trees are directly defined over image patches containing Hermitian matrices. The resulting RF equals in accuracy to a RF based on numerous polarimetric features extracted from a PolSAR image and shows stable performance on multiple data sets. However, neither computation time nor memory have to be spent on the explicit computation of image features.

Remotely sensed images, in particular modern PolSAR data, have often an image size in the megapixel range and thus require not only accurate but also efficient algorithms for their analysis. RFs are per se rather efficient classifiers as the tree structure of the individual baselearners divides the data in each level. In this way only a few processing steps (usually $10 - 100$) are necessary to estimate the target variable. Nevertheless, training a RF and using it for prediction can take several minutes to hours depending on the degree of optimization and the size of the data set.

*Corresponding author

As RFs are a widely used machine learning framework, there exist several open-source as well as commercial implementations which at least partially strive for computationally efficiency, e.g. *randomForest* (implemented in R and not optimized for high dimensional data (Liaw and Wiener, 2002)), *party* (R package of general recursive partitioning (Hothorn et al., 2006)), *Willows* (can deal with many samples but not many features (Zhang et al., 2009)), *Random Jungle* (available only as C++ executable (Schwarz et al., 2010)), *scikit-learn*(Pedregosa et al., 2011), *RandomForests* (commercial software (Systems, 2013)), *bigrf* (optimized for classification of very large datasets (Lim et al., 2014)), *randomForestSRC* (includes classification and regression trees (Ishwaran and Kogalur, 2015)), the *Rborist* package (another R implementation (Seligman, 2015)), and *ranger* (platform independent, designed for the analysis of high dimensional data (Wright and Ziegler, 2017)). Other work have proposed to use different hardware for acceleration, such as multi-core CPUs (Boström, 2011), distributed CPUs (Shotton et al., 2011)), FPGAs (Cheng and Bouganis, 2013), or GPUs (e.g. (Sharp, 2008, Schulz et al., 2016)). While these work make significant contributions, they mainly focus on efficient implementations or specific hardware architectures and only seldom consider algorithmic adaptations. Furthermore, they provide implementations of the standard, multi-purpose version of RFs and cannot easily be adapted to images in general and PolSAR images in particular.

This work investigates multiple strategies to decrease computation time during both, training and prediction, while maintaining accuracy. A RFs computation time is mainly determined by two factors: First, the time cost to create an internal node during training and to apply its test function during prediction. Second, the expected value of the length of a path a sample has to take through the trees, i.e. the number of nodes - and therefore tests - it has to pass. While previous optimization approaches select node tests solely to achieve an optimal (but greedy) performance, we take computation time explicitly into consideration. While Section 2 briefly revisits training and prediction procedures of RFs with a focus on the specific parts tailored towards the use case of classifying PolSAR images under time constraints, Section 4 discusses the acceleration strategies and their experimental results.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-3-105-2018 | © Authors 2018. CC BY 4.0 License.

2. RANDOM FORESTS

A Random Forest (RF, (Ho, 1998, Breiman, 2001)) is a set of multiple decision trees that act as baselearners in an ensemble. By fusing the decision of the individual trees, the advantages of single decision trees (e.g. being applicable to different kinds of data, interpretability, simplicity) are kept, but their limitations (e.g. high variance, prone to overfitting) are avoided. One of the main factors for the success of this strategy is the diversity of the ensemble, i.e. creating accurate but still different baselearners. In the case of RFs, this is usually achieved by using a certain degree of randomness during tree creation.

While a detailed discussion of Decision Trees and Random Forests is beyond the scope of this paper it can be found e.g. in (Criminisi and Shotton, 2013). The following subsections repeat the most essential parts of tree creation, training, and application of the RF introduced in (Hänsch and Hellwich, 2017) that play a role in the optimization of the computation time as discussed in Section 4.

2.1 Tree creation and training

Each tree in a RF consists of several internal nodes including a single root node and multiple terminal nodes. If used as a supervised learning framework, tree creation and training rely on a training set $D = \{(\mathbf{x}, \mathbf{y})_i\}_{i=1,\dots,N}$ of N samples $\mathbf{x}$. The supervised signal consists of the known value of the corresponding target variable $\mathbf{y}$, i.e. a semantic label in the case of classification. Starting at the root node, each non-terminal node applies a binary test to all data points that reach that node and propagates them to the left or right child node depending on the test outcome. The recursive application of the creation of internal nodes stops if the maximum tree height is reached, the current node contains too few samples, or samples of only one class (among others).

2.2 Node Tests

The node tests of the decision trees give RFs their generality as they can be redefined for different data without the need of changing the overall framework. The work of (Lepetit and Fua, 2006, Fröhlich et al., 2012) introduce specific node tests for images that implicitly analyse the local spatial structure by performing comparisons between random pixel pairs within a patch and are extended in (Hänsch and Hellwich, 2017) to the characteristics of PolSAR images. A node test of the RF in this work takes the form

$$\psi(\mathbf{x}) < \theta? \tag{1}$$

where $\mathbf{x}$ is a $w \times w$ patch of local sample covariance matrices of a k-channel PolSAR image (i.e. $\mathbf{x} \in \mathbb{C}^{w \times w \times k \times k}$), the projection function $\psi : \mathbb{C}^{w \times w \times k \times k} \to \mathbb{R}$ is defined below, and $\theta \in \mathbb{R}$ a threshold (see Section 2.3).

The projection function ψ selects one to four matrices $\mathbf{C}_i \in \mathbf{x}$ and computes their spectral distances by Eq. 2-4 where $\tilde{\mathbf{C}}$ is a covariance matrix randomly sampled from the image.

$$\psi_{1p} = d(\mathbf{C}_1, \tilde{\mathbf{C}}) \tag{2}$$
$$\psi_{2p} = d(\mathbf{C}_1, \mathbf{C}_2) \tag{3}$$
$$\psi_{4p} = d(\mathbf{C}_1, \mathbf{C}_2) - d(\mathbf{C}_3, \mathbf{C}_4) \tag{4}$$

These projections enable a spectral and textural analysis of the local neighborhood and are based on a distance measure defined over the space of Hermitian matrices as for example the Euclidean distance of the real-valued elements of the main diagonal (Eq. 5), the distance induced by the Frobenius norm (Eq. 6), the Wishart (Eq. 8) and revised Wishart (Eq. 11) distance as well as their symmetric counterparts (Eq. 9 and 12), the Bartlet distance (Eq. 10), as well as geodesic distances (Eq. 13 and 14). While (Hänsch and Hellwich, 2017) provides a discussion of their basic principles and properties, it should be noted that at least some of those distances are partially based on computationally expensive operations such as matrix functions (e.g. the matrix logarithm).

$$d_E(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{k} (a_{ii} - b_{ii})^2} \tag{5}$$

(6)

$$d_F(\mathbf{A}, \mathbf{B}) = ||\mathbf{A} - \mathbf{B}||_F = \sqrt{\sum_{i=1}^{k} \sum_{j=1}^{k} |a_{ij} - b_{ij}|^2} \tag{7}$$

$$d_W(\mathbf{A}, \mathbf{B}) = \ln(|\mathbf{B}|) + \mathrm{Tr}(\mathbf{B}^{-1}\mathbf{A}) \tag{8}$$

$$d_{WS}(\mathbf{A}, \mathbf{B}) = \frac{\ln(|\mathbf{AB}|) + \mathrm{Tr}(\mathbf{AB}^{-1} + \mathbf{BA}^{-1})}{2} \tag{9}$$

$$d_{Ba}(\mathbf{A}, \mathbf{B}) = \ln \frac{|\mathbf{A} + \mathbf{B}|^2}{|\mathbf{A}||\mathbf{B}|} \tag{10}$$

$$d_{RW}(\mathbf{A}, \mathbf{B}) = \ln \left(\frac{|\mathbf{B}|}{|\mathbf{A}|} \right) + \mathrm{Tr}(\mathbf{B}^{-1}\mathbf{A}) \tag{11}$$

$$d_{RWS}(\mathbf{A}, \mathbf{B}) = \frac{\mathrm{Tr}\left(\mathbf{AB}^{-1} + \mathbf{BA}^{-1}\right)}{2} \tag{12}$$

$$d_G(\mathbf{A}, \mathbf{B}) = ||\ln(\mathbf{A}^{-\frac{1}{2}}\mathbf{B}\mathbf{A}^{-\frac{1}{2}})||_F \tag{13}$$

$$d_{LE}(\mathbf{A}, \mathbf{B}) = ||\ln(\mathbf{A}) - \ln(\mathbf{B})||_F \tag{14}$$

2.3 Split Point Definition

While there are several ways to define the split point θ in Eq. 1 (see e.g. (Hänsch and Hellwich, 2015)), this paper investigates four variants that span a range from simple random sampling to a fully-optimized selection. Given samples $\mathbf{x} \in D_n \subset D$ at a node n, the split point θ of this node can be determined by one of the following four methods:

- Uniform sampling:

$$\theta \sim U\left(\min_{\mathbf{x} \in D_n} (\psi(\mathbf{x})), \max_{\mathbf{x} \in D_n} (\psi(\mathbf{x})) \right) \tag{15}$$

  where $U(\cdot, \cdot)$ is the uniform distribution.

- Median based:

$$\theta = \mathrm{median}\,(D_n) \tag{16}$$

- Inter class:

$$\theta = \frac{\mathbf{x}_{c_1} + \mathbf{x}_{c_2}}{2} \tag{17}$$

  where $\mathbf{x}_{c_1}, \mathbf{x}_{c_2}$ are two random samples of two random, but different classes $c_1 \neq c_2$.

- Grid search:

$$\theta = \arg\max_{\theta^*} \Delta I(D_n, \theta^*) \tag{18}$$

  where $\Delta I(D_n, \theta^*)$ is the drop of impurity defined below in Eq. 19 achieved by split point θ^* on the local data set D_n.

2.4 Optimized Test Selection

A common optimization strategy that aims for stronger baselearners and thus increased performance of the whole ensemble is to create multiple test candidates and select the best test based on the drop of impurity ΔI (Equation 19):

$$\Delta I = I(P(y|D_n)) - \sum_{i \in \{L,R\}} \frac{|D_{n_i}|}{|D_n|} I(P(y|D_{n_i})) \quad (19)$$

$$I(P(y)) = 1 - \sum_{i=1}^{C} P(y_i)^2 \quad (20)$$

where n_L, n_R are the left and right child nodes of node n, respectively, with the respective data subsets D_{n_L}, D_{n_R} (with $D_{n_L} \cup D_{n_R} = D_n$ and $D_{n_L} \cap D_{n_R} = \emptyset$). The node impurity $I(P(y))$ is measured by the Gini impurity (Equation 20) of the local class posteriors $P(y|D_n)$.

2.5 Prediction

For prediction of a query sample by a trained RF, it is propagated through all trees. Starting at the root node, it will reach exactly one leaf node $n_t(\mathbf{x})$ in each tree t. The class posterior $P(y|n_t(\mathbf{x}))$ assigned to these leafs during training, is averaged to obtain the final class posterior $P(y|\mathbf{x})$:

$$P(y|\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} P(y|n_t(\mathbf{x})) \quad (21)$$

3. DATA

(a) Image data (b) Reference data

Figure 1. Image and reference data for the Oberpfaffenhofen data set (PolSAR image acquired by E-SAR, DLR).

For the experiments of the next section, a fully polarimetric image acquired by the E-SAR sensor (DLR) in L-band over Oberpfaffenhofen, Germany, is used. Figure 1(a) shows a false-color composite. This dataset contains man made as well as natural structures and is manually labelled with five different classes as depicted in Figure 1(b): City (red), Road (blue), Forest (dark green), Shrubland (light green), and Field (yellow).

The total image size is $1390 \times 6640px$ and contains more than 6M labelled samples. The image data is divided into five different folds, training data are randomly sampled from four stripes while the (complete) fifth stripe is used for testing resulting in roughly 1.8M query pixels.

The balanced accuracy is measured as the average detection rate per class for each fold, while the final accuracy estimate is the average over all folds.

4. ACCELERATION

A RF with the parameters stated in Table 1 is used as the initial model. It is trained on 4K samples per class randomly drawn from the respective training regions (i.e. 20K samples in total) and achieves a balanced accuracy of 87%.

Number of trees:	30
Maximal tree height:	50
Minimal samples to continue splitting:	10
Split point definition:	Grid search
Number of test candidates:	100
Number of threads:	1

Table 1. Parameter settings

The influence of tree height and number of trees on computation time and accuracy is well known (see e.g. (Hänsch, 2014)) and thus excluded from the discussion in the following subsections.

4.1 Multithreading

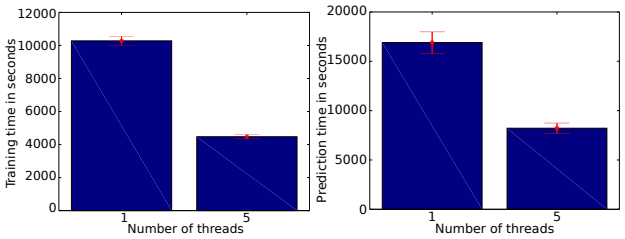


Figure 2. Change in training and prediction time by using different number of threads

In contrast to other approaches such as Boosting, the individual trees within a (standard) RF are trained completely independent of each other. Also during applying a RF, the predictions of a single tree are independent of the other trees. Only after the tree predictions are made, they need to be fused (usually averaged) which might introduce dependencies among the trees. If the trees have their own part of the memory to store their predictions, these dependencies are strongly minimized and only occur after the individual tree predictions are finished.

Consequently, Random Forests are very well suited for multithreading. Theoretically, each tree can be trained and used for prediction in its own thread which runs completely independent of the threads of the other trees.

However, RF training requires intense memory access which makes thread scheduling computationally demanding, in particular if CPU cache cannot be exploited to the full extend. Nevertheless, using a certain number of threads does lead to a decrease of computation time, but how many threads should be used and how big the gain really is depends a lot on the used computer architecture and the specific implementation.

Figure 2 shows the training and prediction time of a RF using a single thread and using five parallel threads. Both times have been decreased by more than 50% (from 10276sec to 4472sec for training and 16897sec to 8217sec for prediction) which does state a considerable gain, but not as much as one might have expected. Using more than five threads did not lead to further time improvements for the used implementation as the number of cache misses and internal mutex locks due to memory allocations increases. Multithreading has no influence on the achieved accuracy.

4.2 Feature Pre-computation

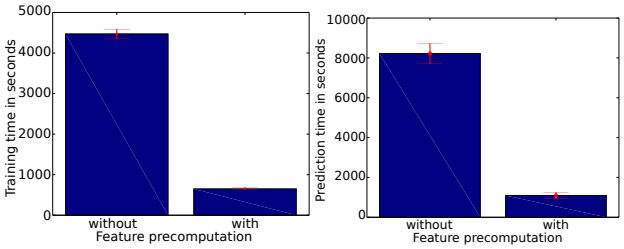


Figure 3. Change in training and prediction time by using precomputed features.

Many of the node tests in Section 2.2 are based on matrix properties such as determinant, trace, and inverse, or apply matrix functions such as the matrix logarithm. While the PolSAR image patches are processed by the trees, these operations are applied thousand to millions of times and at least partially to the same matrices, i.e. the covariance matrices contained in the individual pixels. These operations, however, are computationally very demanding, as for example the computation of the matrix logarithm involves an eigenvalue decomposition of the corresponding matrix.

One possibility to speed up processing considerably is to precompute at least some of those entities. In the current implementation we decided to represent the data in its eigenvalue decomposition, which only has to be computed once while the data is loaded from the hard disc. This causes an increased memory footprint of the data: The covariance matrices are Hermitian and consist of only six unique elements of which three (the main diagonal) are real valued. Thus, nine floats are sufficient to represent a given covariance matrix. The eigenvalue decomposition, however, requires 21 floats: Three for the real-valued eigenvalues and six for the three real and imaginary parts for each of the three eigenvectors.

However, the gained decrease in computation time during training and prediction is significant as Figure 3 shows. Both times are reduced to roughly 14% of the original time (training from 4472sec to 650sec and prediction from 8217sec to 1096sec). The different data representation has no influence on the achieved accuracy.

4.3 Node Optimization

The methods discussed in Section 4.1 and 4.2 have only an influence on the computation time during training and prediction, but neither alter the topology of the trees nor the quality of the leaf predictions and thus the classification accuracy stays unaffected.

The approach evaluated in this section, however, does have an influence on the accuracy of the resulting RF, stating to some extend a tradeoff between an increase of computational efficiency and a decrease in performance.

There are different ways to define the split point within the node tests of a decision tree as briefly explained in Section 2.3. This section discusses four different variants, namely uniform sampling between the minimal and maximal value of the projected data, the median of the projected data, a grid-search of the optimal split point (according to the drop of impurity measured by Gini), and an inter-class split point definition, which randomly

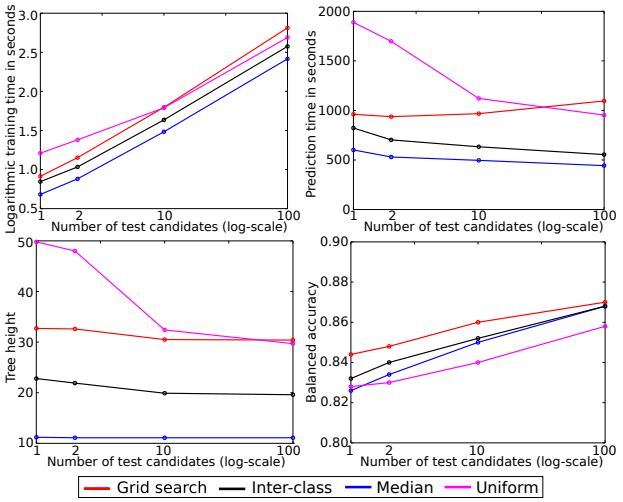


Figure 4. Influence on computation time, tree topology, and accuracy by optimized test selection.

draws one sample of two different classes and defines the split point as their average value.

It should be noted that these split point definitions state different degrees of optimization: While uniform sampling does not perform any kind of optimization, the splits defined by the median of the projected values are at least data-optimal in the sense that they are (if possible) of equal size and thus lead to balanced trees. Balanced trees are advantageous as they perform the maximal amount of tests to all data points while keeping the minimum tree height. However, no class information is taken into account and thus the defined split points are independent from the given classification task. Since inter-class splits randomly draw a sample from the given class distribution, there is a high chance that the corresponding data point is from a feature area with the majority probability mass of the corresponding class likelihood. By defining the split point to be in between two samples of different classes, there is a high potential to separate those two classes very well. Performing a grid search of the optimal split point is the strongest form of optimization.

Those four different kinds of node tests come with different time complexities as well. While uniform sampling requires only the computation of the minimal and maximal value within the given data set at a specific node, the median based split point definition requires a sorting operation (only a partial sorting, though, as only the center element of the completely sorted array has to be determined). The inter-class splits only draw one sample of two classes and compute their average. It is thus very efficient, while grid search is computationally very demanding as it requires the computation and evaluation of multiple splits.

Training and prediction time of a decision tree are, however, not only determined by the computational cost to create a node test, but also on how many of those node tests have to be applied to the data. This effect can be seen in Figure 4. Both, median and uniform split point selections have on average a linear time complexity, inter-class is very efficient, while grid search is computationally demanding. Nevertheless, median based split points lead by far to the fasted training only followed by the much more efficient inter-class sampling. The reason is that median based splits result in very balanced trees and thus to the shortest average path length (roughly 11 in this example). While the creation of indi-

vidual nodes might take longer time, fewer of them have to be applied to the data. The same reason causes uniform sampling to have the largest training time: The defined splits are very unbalanced which causes the creation of higher trees (most of the trees reach the maximal tree height of 50 in this example) and thus an increased average path length. For all methods, the training time increases almost linearly with the number of test candidates per node. Interestingly, if many test candidates are used, trees created by uniform sampling become faster during training than trees based on grid search. In this case, the optimization helps to avoid unbalanced splits which equalizes the overall path length (decreased for uniform sampling from 50 for 1 test candidate to 30 for 100 test candidates) and leads to a larger influence of the individual split point selection costs. Grid-search, inter-class, and median split points already lead to well balanced splits so that multiple candidates do not influence the path length much.

The prediction time is not directly influenced by the type of split point definition or the number of test candidates per node as those optimizations are only carried out during training. However, the creation of better and in particular more balanced trees does reduce the prediction time as less tests have to be applied to the data. This effect is strongest for the uniform split point selection, but is also evident for inter-class and median based splits.

The accuracy of all methods increases considerably with an increased number of test candidates. If only one test candidate is used, the strong optimization of grid search pays off, leading to the highest accuracy, followed by inter-class split point definition which takes the class label into account as well. There is no significant difference between uniform sampled and median-based split point selection if node tests are not optimized. However, if multiple split candidates are used, the difference between grid search, inter class, and median based splits shrinks significantly until there is barely a difference anymore for 100 tests. Thus, while median based split point selection needs roughly only 40% of the training and prediction time of grid-search, it does achieve the nearly same accuracy.

4.4 Constrained Node Optimization

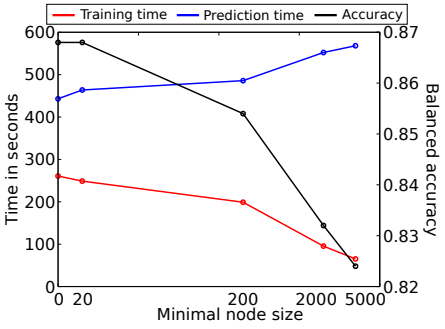


Figure 5. Influence on computation time and accuracy by constrained node optimization.

While optimized test selection as evaluated in Section 4.3 does increase the performance significantly, it is questionable whether the same amount of optimization has or can be carried out everywhere in the tree. Nodes at higher tree levels, close to the leafs, contain only a small fraction of the original samples. Most of the “easy” decisions are already made at that point and very often further splits are dominated by noise either in the data or in the labels. Finding an optimal split based on samples, which are very similar to each other (otherwise they are unlikely to have been propagated to the same node so high in the tree), is a very hard task. The corresponding decisions will show a high degree of variance due to the stronger influence of noise and the small amount of samples. That is why it might be reasonable to restrict optimized test selection to nodes that have a certain amount of samples and to skip it for nodes with too few data.

Figure 5 shows the influence of this optimization strategy for different minimal amounts of samples a node have to obtain in order to perform optimized test selection. As expected, the higher the threshold on the number of samples is, the faster is the training. However, the prediction time increases, which already indicates that inferior tests are selected, i.e. tests that lead to less balanced trees and thus to an increased average path length. This consequently leads to a decreased accuracy of the tree. The effect is relatively small for a minimal number of 20 samples, where accuracy stays constant. Training time decreases to 95%. However, prediction time is increased by 5% as well.

Thus, at least for median based split-point definition, constraining the optimized test selection does not state a meaningful strategy to improve time performance while maintaining accuracy.

4.5 Time-sensitive Node Optimization

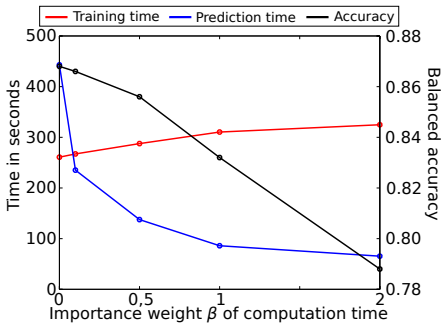


Figure 6. Influence on computation time and accuracy by time sensitive node optimization.

So far, optimized test selection only took the obtained drop of impurity into account and selected the test that leads to the purest child nodes given the training data of the current node. However, different node tests, in particular the different distance measures discussed in Section 2.2, have different time complexities. The computation time can be included as another objective function during test selection, e.g. by recomputing the test score as

$$s = \Delta I \cdot t^{-\beta} \tag{22}$$

where ΔI is the drop of impurity (Eq. 19), t is the time needed to perform this test on the training samples, and β is a weight that controls the influence of the computation time on the final score. If, for example, $\beta = 1$, then a test that needs twice the computation time of another test, has to have twice the drop of impurity in order to be still competitive.

Figure 6 reports the changes in computation times and accuracy for different weights, ranging from $\beta = 0$ (i.e. computation time is not considered at all) to $\beta = 2$ (i.e. computation time dominates the final score). As expected, the larger β (i.e. the more importance is given to the computation time), the more does the prediction time decrease. However, it appears that at least for the used test functions, faster tests are also weaker in performance.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-3-105-2018 | © Authors 2018. CC BY 4.0 License.

Thus, if they are selected with a higher probability, the accuracy of the forest decreases as well. The selection of inferior tests also leads to a slight increase of the training time.

While the increase of training time and decrease of accuracy are only marginal for $\beta = 0.1$ (i.e. less than 3% increase of training time and accuracy changes from 86.8% to 86.6%), prediction time increased considerably from 443sec to 235sec, i.e. roughly 50%. Thus, as long as not too much importance is given to computation time and accuracy remains the dominating factor, computational efficiency can successfully be included into the optimization function.

4.6 Subsampling

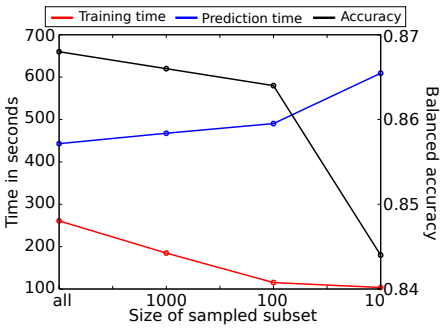


Figure 7. Influence on computation time and accuracy by subsampling.

RFs, at least in their standard formulation, already perform a tree-wise subsampling due to bagging (Breiman, 1996): In order to increase the diversity of the ensemble, N samples are drawn with replacement from the original training set of size N. This does not lead to a decreased training set size, but to the repetition of around 33% of samples which can lead to faster processing (depending on the implementation). Some RF implementations (including the one of this paper) actually perform subbagging, i.e. real subsampling by drawing only around $2/3$ of the original training samples without replacement.

This section investigates subsampling that is additionally performed in each node, i.e. each node n performs split point computation and optimized test selection only on a random subset of size $N_s \leq N_n$ of the original N_n samples of that node. The finally selected test, however, is applied to all N_n samples as the whole data set has to be propagated to the child nodes which will perform the further splitting based on a different random subset.

Figure 7 illustrates the influence of differently sized subsets on computation time and accuracy. Subsampling has an adverse effect on accuracy which is only marginal for larger random sets (i.e. for $N_s = 1000$ accuracy changes from 86.8% to 86.6% while training time decreases to 70%), but quite severe for small sets (i.e for $N_s = 10$ accuracy changes from 86.8% to 84.4% while training time decreases to 40%).

4.7 Structured Prediction

RFs are able to perform structured prediction, i.e. instead of estimating a single posterior distribution (usually for the center pixel of an image patch), they can provide an estimation of the spatial posterior distribution around a query sample. Those structured predictions are usually used to increase the number of estimates

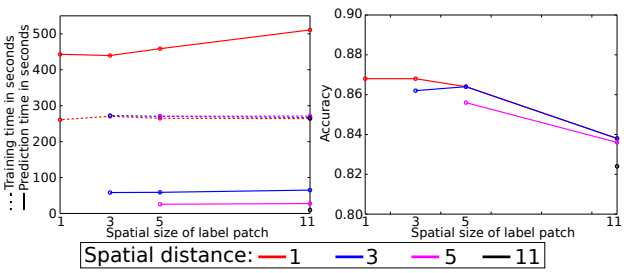


Figure 8. Influence on computation time and accuracy by structured prediction.

per pixel for the final averaging in order to provide a better per-pixel estimate. However, they can be used to speed-up the prediction time as well: Since a structured label already provides an estimation of the semantic information of the spatial neighborhood of a sample, not all pixels of the query image have to be propagated through the forest. It is possible to skip a certain amount of pixels in between two estimates and use their structured prediction to fill the gaps.

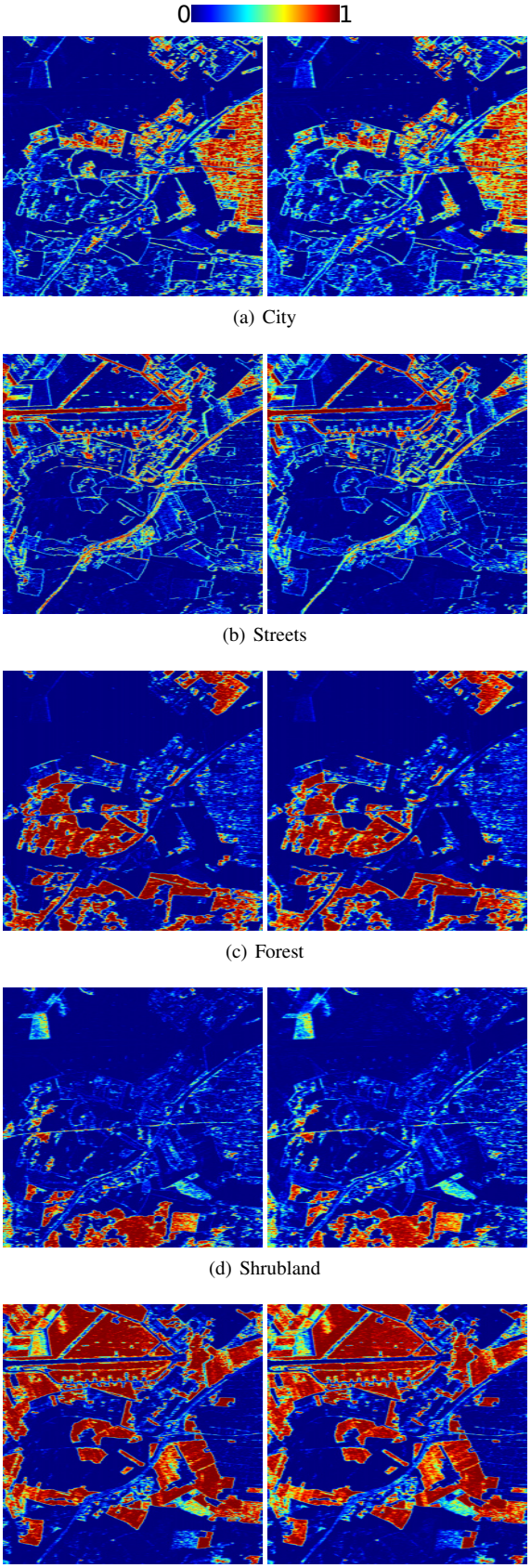
Figure 8 shows the influence of different sizes of the estimated posterior patches and the distance between two estimates. Naturally, an increased distance between query samples leads to a tremendous decrease of prediction time since even for a gap of one pixel one 25% of the samples have to be used for prediction. The training time increases only marginal as most of the computational effort is spent on optimized test selection and propagating the samples through the trees, while the actual estimation of the leaf information is a simple histogram computation. If only every third pixel is used during the estimation (i.e. a gap of two pixels between two queries), the prediction time decreased by roughly 75% (averaged over all patch sizes). However, structured prediction in this setup mostly has a adverse effect on accuracy. If every pixel is queried and the patch size is small (i.e. 3×3 instead of 1×1) accuracy stays unchanged, but decreases for larger patch sizes. If only every third pixel is queried with a patch size of 3×3, accuracy decreases slightly from 86.8% to 86.2%, but can partially recovered by using overlapping patches of size 5×5. Larger patch sizes or distances between query samples decrease accuracy significantly.

Thus, if moderate distances between query samples are used as well as small patch sizes, a considerable speed-up regarding prediction time can be achieved, while training time and accuracy stay almost unaffected.

4.8 Final Model

Based on the previous subsections, the initial model stated in Table 1 is changed in the following way:

- Multithreading (Section 4.1) as well as feature precomputation (Section 4.2) are used as they are only beneficial to both training and prediction times while having no effect on accuracy.
- Node optimization (Section 4.3) is changed from grid search with 100 test candidates to median based split point definition with 100 test candidates, using a more efficient split point definition and more importantly leading to very balanced trees and thus the shortest expected path length. Experiments show a considerable decrease in computation time during training and prediction, while accuracy is maintained.



(a) City

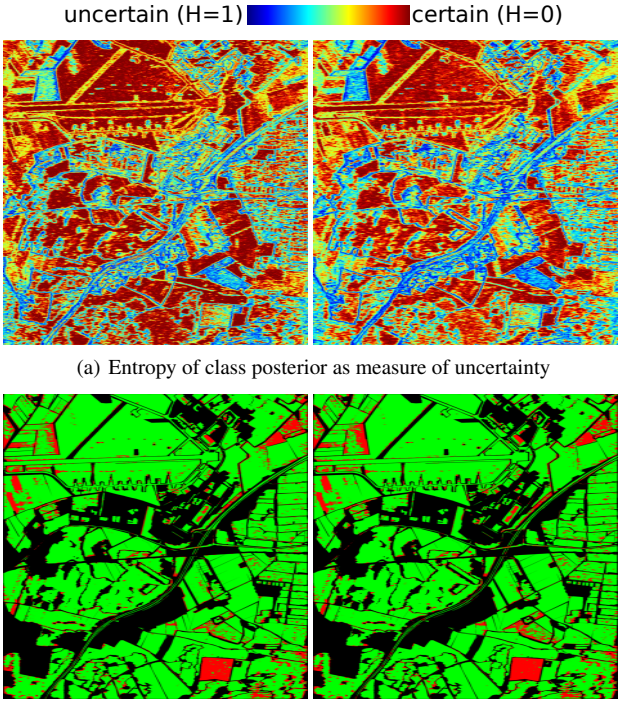
(b) Streets

(c) Forest

(d) Shrubland

(e) Field

Figure 9. Class posterior of the test samples (acquired over all folds) of the initial (left) and final (right) model.



(a) Entropy of class posterior as measure of uncertainty

(b) Correct (green) and wrong (red) classification according to reference data (pixels with no available reference label are denoted in black, see Figure 1(b).)

Figure 10. Correctness of the classification result for the initial (left) and final (right) model.

- Limiting the test optimization to nodes with a specific sample size (Section 4.4) is rejected as only marginal gains in computation time could be achieved during training, while prediction time and accuracy are adversely affected.
- Computation time is included into the objective function of the node optimization (Section 4.5), but only with a relatively small importance factor of $\beta = 0.1$.
- Moderate subsampling (Section 4.6) is performed in each node with a subset of size $N_s = 1000$ as it leads to a considerable speed-up during training with marginal (negative) effects on prediction time and accuracy.
- Structured prediction (Section 4.7) is performed with a spatial distance of two pixels (i.e. every third pixel is used) and a patch size of 5×5 pixels.

Compared to the initial model, training time is decreased from 10276sec to 178sec (i.e. to 1.7% or equivalently by a factor of 58), prediction time is decreased from 16897sec to 34sec (i.e. to 0.2% or equivalently by a factor 500), while accuracy is only marginally decreased from 87% to 85.8%.

Figure 9 shows the class posterior (of the test data) of the initial model on the left and of the final model on the right side (obtained from the different folds). The difference of the balanced accuracy values already indicated that there are barely any changes in the final (discrete) classification decisions. The visual representation of the class posterior qualitatively confirms that not only the final classification stays consistent but also the intermediate estimations of the posterior distribution. This is further confirmed by Figure 10(a) which shows the entropy of the class posterior for

both models, which is one for a uniform distribution (maximal uncertainty) and zero for a posterior where only a single class obtains 100% of the probability mass (minimal uncertainty). Figure 10(b) illustrates that both models make highly accurate decisions while the remaining errors are consistently distributed.

5. CONCLUSION

This paper investigates and discusses several strategies to accelerate training and prediction of RFs which are specifically tailored towards the semantic pixel-wise labeling of PolSAR images. The experiments show that in particular an efficient representation of the PolSAR data as precomputed Eigenvalue decomposition enables a tremendous decrease of computation time during both, training and prediction, as most of the applied polarimetric distance measures in the node tests can be computed more efficiently and repeated calculations of the eigenvalues are avoided. A careful choice of the split point definition leads to a further decrease of the computation times, in particular if median based split points are used. As it is not only possible to compute these split points efficiently but they also result in maximally balanced trees and thus the shortest expected path length. Subsampling or restricting the test optimization to nodes of a certain size has to be used with care - if at all - as it bears the risk to decrease classification performance without much gain in efficiency. Computation time can be explicitly included into the optimization framework and leads to faster but equally accurate trees if the class impurity remains the dominant factor. Finally, structured prediction can be used to speed-up the prediction process by an order of magnitude at the cost of a moderate decrease in accuracy.

The final model that applies multithreading, feature precomputation, median based split point definition, node-wise data subsampling, time-sensitive test selection, and structured prediction is 58 times faster during training and 500 times faster during prediction while maintaining 98.6% of accuracy.

Future work will focus on a stronger exploitation of parallelization of the individual trees by preallocating the necessary memory and reducing the risk of cache misses as far as possible.

REFERENCES

Boström, H., 2011. Concurrent learning of large-scale random forests. In: A. Kofod-Petersen, F. Heintz and H. Langseth (eds), *SCAI*, Frontiers in Artificial Intelligence and Applications, Vol. 227, IOS Press, pp. 20–29.

Breiman, L., 1996. Bagging predictors. *Machine Learning* 24(2), pp. 123–140.

Breiman, L., 2001. Random Forests. *Machine Learning* 45(1), pp. 5–32.

Cheng, C. and Bouganis, C. S., 2013. Accelerating random forest training process using fpga. In: *2013 23rd International Conference on Field programmable Logic and Applications*, pp. 1–7.

Criminisi, A. and Shotton, J., 2013. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer Publishing Company, Incorporated.

Fröhlich, B., Rodner, E. and Denzler, J., 2012. Semantic segmentation with millions of features: Integrating multiple cues in a combined Random Forest approach. In: *11th Asian Conference on Computer Vision*, Daejeon, Korea, pp. 218–231.

Hänsch, R., 2014. Generic object categorization in PolSAR images - and beyond. PhD thesis, TU Berlin, Germany.

Hänsch, R. and Hellwich, O., 2015. Evaluation of tree creation methods within Random Forests for classification of PolSAR images. In: *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Milan, Italy, pp. 361–364.

Hänsch, R. and Hellwich, O., 2017. Skipping the real world: Classification of polsar images without explicit feature extraction. *ISPRS Journal of Photogrammetry and Remote Sensing*.

Ho, T. K., 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(8), pp. 832–844.

Hothorn, T., Hornik, K. and Zeileis, A., 2006. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics* 15(3), pp. 651–674.

Ishwaran, H. and Kogalur, U., 2015. randomforestsrc: Random forests for survival, regression and classification, r package version 1.6.1. `http://CRAN.R-project.org/package=randomForestSRC`.

Lepetit, V. and Fua, P., 2006. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28(9), pp. 1465–1479.

Liaw, A. and Wiener, M., 2002. Classification and regression by randomforest. *R News* 2(3), pp. 18–22.

Lim, A., Breiman, L. and Cutler, A., 2014. bigrf: Big random forests: Classification and regression forests for large data sets. r package version 0.1-11. `http://CRAN.R-project.org/package=bigrf`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.* 12, pp. 2825–2830.

Schulz, H., Waldvogel, B., Sheikh, R. and Behnke, S., 2016. *CURFIL: A GPU Library for Image Labeling with Random Forests*. Springer International Publishing, Cham, pp. 416–432.

Schwarz, D. F., Knig, I. R. and Ziegler, A., 2010. On safari to random jungle: a fast implementation of random forests for high-dimensional data. *Bioinformatics* 26(14), pp. 1752–1758.

Seligman, M., 2015. Rborist: Extensible, parallelizable implementation of the random forest algorithm, r package version 0.1-0. `http://CRAN.R-project.org/package=Rborist`.

Sharp, T., 2008. *Implementing Decision Trees and Forests on a GPU*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 595–608.

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A. and Blake, A., 2011. Real-time human pose recognition in parts from single depth images. In: *CVPR 2011*, pp. 1297–1304.

Systems, S., 2013. Salford predictive modeler users guide. version 7.0. `http://www.salford-systems.com`.

Wright, M. N. and Ziegler, A., 2017. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software* 77(1), pp. 1–17.

Zhang, H., Wang, M. and Chen, X., 2009. Willows: a memory efficient tree and forest construction package. *BMC Bioinformatics* 10(1), pp. 130.