Self-Optimizing Random Forests

Anonymous[1]

[1]Anonymous Institution

Abstract Random Forests (RFs) have become a standard pick of data scientists for classification and regression tasks over the last two decades. Several modifications of RFs have been proposed to eliminate the limitation of standard RFs to axis-aligned splits through the notion of Oblique Random Forests (ORFs), which allow that splits are defined over linear combinations of attributes rather than a single attribute. However, like there is no single best learner for all supervised learning problems, there is also not a single best RF type. In this paper, we present self-optimizing random forests (SORFs). SORFs incrementally create a forest by growing different tree types and identifying the best tree type based on extrapolations of the forest performance curves obtained from out-of-bag performance estimate. Our exhaustive empirical evaluation shows that SORFs consistently achieve the maximum performance across three considered types of RFs while requiring only half the time for training all these forests on average. At the same time, SORFs outperform standard RFs statistically significantly and by at least 0.01 in accuracy on 25% of the considered datasets and even substantially beyond 0.35 in two cases.

1 Introduction

Random Forests (RFs) (Breiman, 2001) have become a standard pick of data scientists for classification and regression tasks over the last two decades. RFs are Bagging ensembles (Breiman, 1996) of decision trees (Quinlan, 1997) in which the decision trees only have access to a random subset of features in each inner node (Ho, 1998). The main strength of RFs is that they exhibit oftentimes strong performance without the need of hyperparameter tuning.

Several modifications of RFs have been proposed to eliminate the limitation of standard RFs to axis-aligned splits through the notion of Oblique Random Forests (ORFs). ORFs are RFs in which the split is not defined over a single original feature but a linear combination thereof, which amounts to an arbitrarily oriented hyperplane in the input space. ORFs were already discussed in Breiman's original work (Breiman, 2001), but optimizing the decision direction was, to our knowledge, first considered by Lemmond et al. who used a Linear Discriminant Analysis (LDA) to identify the split direction (Lemmond et al., 2008). Follow-up approaches considered a solution by regression (Menze et al., 2011), the application of a (potentially kernelized) Principal Component Analysis (PCA), (Zhang and Suganthan, 2014; Zhang et al., 2014; Wang et al., 2020) or even a combination of PCA and LDA (Zhang and Suganthan, 2014). ORFs have a close connection to Rotation Forests (Rodríguez et al., 2006), which compute the projection not in the inner leafs but previously once for the whole tree. Thereby, Rotation Forests can be seen as one type of ORFs as well.

As often observed in algorithm selection or configuration, there is not a single best rotation strategy. In many situations, it is best to not use any rotation at all while sometimes and LDA-based rotation or PCA-based rotation is best. Based on this observation, Zhang et al. proposed ensemble forests (Zhang and Suganthan, 2014), which optimize over different possible rotations in each inner node (including no rotation). However and perhaps surprisingly, we could verify that this type of forest does not consistently achieve the best performance obtained by any of the three forest types while exhibiting a substantially increased training time.

This paper makes two contributions. The minor contribution is to bring Oblique Random Forests to the attention of the AutoML community, which has not taken notice of this interesting extension

of RFs so far. The main contribution is then the introduction of Self-Optimizing Random Forests (SORFs). The idea is to incrementally create a forest by growing different tree *types*. Extrapolating the performance curves obtained from the out-of-bag error of sub-forests of the different tree types, SORFs early detect the type of tree that is most suitable for the dataset on which they are being trained. As soon as the performance curve models have stabilized, the best tree type is picked, and only trees from this type are grown. Here, we focus on standard random trees, LDA-rotated trees, and PCA-rotated trees. Besides, SORFs use this performance curve to automatically decide whether to add additional trees to the forest. So SORFs do *not* require the number of trees as a hyperparameter, which makes them even more straight forward to use than standard RFs.

Our experimental evaluation reveals that SORFs consistently achieve close-optimal performance compared to the best of the three RF types while requiring substantially less time to be trained. On 96 classification datasets, SORFs present an average regret of only 0.001 in accuracy compared to the single best forest type. On 25% of the datasets, significant improvements of at least 0.01 in accuracy can be achieved over standard RFs, in two cases even above 0.3. At the same time, SORFs exhibit substantially lower runtime compared to evaluating all of the three forest types separately, which would be the time required by an AutoML tool that treat them as a black box.

Therefore, we consider that a broader impact of SORFs to AutoML or even data science in general could be that SORFs replace standard RFs and even porfolios containing Oblique RFs. Thanks to the obsolete forest size hyperparameter this will even *reduce* the search space size of an AutoML tool that replaces standard RFs by SORFs.

2 Background on Oblique Random Forests

Random Forests (RFs) are ensembles of decision trees with two special properties (Breiman, 2001). First, RFs are *bagging ensembles* (Breiman, 1996), which means that each decision tree is being trained not on the original data but on a bootstrap sample thereof. Second, in each inner node of the tree, the training mechanism considers a random subspace (Ho, 1998), which means that it has access only to a random subset of the features. Both mechanisms diversify the ensemble.

Oblique RFs (ORFs) are a generalization of RFs that consist of decision trees in which the split points are not necessarily axis-aligned. Formally, a split point is then not formulated as $x_j \leq c$, where x_j is the value of the j-th feature of some instance x, but it is formulated over a *projection* of the considered attributes to some line β, i.e., $x^T\beta \leq c$. ORFs have been introduced by Breiman himself (Breiman, 2001) with random projections but received attention rather sporadically over time (Lemmond et al., 2008; Menze et al., 2011; Zhang and Suganthan, 2014; Zhang et al., 2014; Wang et al., 2020). A special type of ORF is a *rotation forest* (Rodríguez et al., 2006), in which the projections are defined *per tree* and cannot vary among inner nodes.

The general learning algorithm of an oblique random forest consisting of L trees is as follows:

1. For each $i = 1, .., L$
2. Create a dataset D_i by sampling from the original dataset D with replacement (bootsrapping).
3. At each node with purity $< \pi$ and with more than η instances associated:
   (a) randomly select r out of the original d features.
   (b) transform the data available in the node from the r selected features according to some projection to the new directions $\beta_1, .., \beta_{r'}$
   (c) calculate the best split in the transformed space, and create a left and right child with instances partitioned according to the split point.
   (d) repeat this procedure with the left and right child respectively.

The only difference to the standard RF is step 3b, which is only used by ORFs. The transformation used in that part must be stored in the node because it needs to be applied to new instances arriving at prediction time. A new instance x is classified like in a standard RF except that the (original) instance is projected in each inner node according to the projection stored in it. The ORF approaches mainly differ in how the projections in step 3b are chosen. We briefly recap four of them.

The simplest approach is to create random projections. Such an approach was mentioned by Breiman himself (Breiman, 2001). In this case, the coefficients of β are uniformly drawn from $[-1, 1]$. Notably, random projections do not even need to be rotations of the original data.

Another technique to create the split is to map the class labels to numbers and to address the direction question as a regression problem. Menze et al. (Menze et al., 2011) introduce an approach in which the coordinate of a point on the line onto which it is projected is directly interpreted as a class prediction. Treating classes as numbers, the distance between the true label and this prediction is computed in order to identify a smooth loss. The best projection line is then found with a ridge regression approach.

Another broadly applied technique is to transform the data via a Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). The PCA approach with a full rotation without projection is for example taken in (Zhang and Suganthan, 2014). If desired, only a subset of the first k principal components can be considered for projection. An extreme case is taken in (Wang et al., 2020), which only consider the case of $k = 1$ and hence project the data onto a line. An approach based on LDA projection is taken for example in (Lemmond et al., 2008; Zhang and Suganthan, 2014). In this paper, we refer to RFs of these types, i.e., RFs that project the data in each inner node, as PCA RFs and LDA RFs respectively.

To make projection-based RFs more stable and consistently superior to standard RFs, Zhang et al. (Zhang and Suganthan, 2014) have introduced projection ensemble RFs (PERFs). This has motivated the application of PERFs, which consider possibly *several* projections *in addition* to the standard random space used by the standard RF. The term "ensemble" here stems from the fact that an ensemble of projections (including no projection) is considered in each inner node. In (Zhang and Suganthan, 2014), the authors consider the identity (no) projection, a PCA, and an LDA.

However, PERFs are not a good final answer to the question of rotations. While experimental evaluations have shown that such projections, specifically LDA, oftentimes improve over standard RFs, there are also some counter-examples in which results deteriorate. Besides, the training time of PERFs is enormous corresponding to the one of training all forest types.

3 Self-Optimizing Random Forests

The core idea of Self-Optimizing Random Forests (SORFs) is to maintain different tree *types* and to detect during training which of the tree types works best. At a high level, SORFs grow different sub-forests, one for each tree type, and observe which of these forests evolves best. Projecting the performance curves of each of these sub-forest, the best tree type is chosen as soon as the performance projection is stable for each of the types. The chosen forest type is then grown until its performance curve plateaus or a predefined maximum number of trees has been reached.

The final forest will only contain the trees of the chosen type. That is, the trees grown for the tree types not finally chosen are simply discarded. While it would be possible in principle to create a mixed forest with different tree types, we abstain from this option in this paper to create a simple baseline for then perhaps more sophisticated follow-up works.

In this paper, we consider only three tree types based on the used rotation (no rotation, LDA, or PCA) throughout all nodes in the tree. Other tree types like Kernel PCA trees or common hyperparameters of decision trees and hence RFs such as purity and minimum number of instances in a leaf could be compiled into different tree types as well, which is however beyond our scope.

3.1 Creating Performance Curves Using the Out-of-Bag Error

If the trees of an RF are trained (semi-)sequentially, one can create a performance curve during training. This performance curve is similar to what Cortes et al. (Cortes et al., 1994) refer to as a *capacity* curve since it shows the performance of the ensemble, i.e., the accuracy or the cross-entropy-loss, as a function of the ensemble *size* and hence the capacity of the ensemble. We deliberately avoid the term of a learning curve here, because this is typically considered as a function of the number of training samples or time or iterations spent by an optimizer.

There are mainly two possibilities to create this type of performance curve in RFs. The first one is to separate a fraction of data points for this purpose. This is the same strategy most AutoML tools would apply when considering each forest type as one algorithm and optimizing over these (without of course building an explicit performance curve). The second approach is to exploit the fact that random forests have the capacity to estimate the generalization performance through the notion of the out-of-bag (OOB) estimate. The OOB estimate is computed simply by predicting the labels of all the data points in the training data but, for every instance x_i in the dataset only using those trees of the ensemble for predictions that did not have x_i in their bootstrap sample.

None of the two strategies is obviously superior over the other. Based on the claims that the OOB error is an unbiased estimate for the test error (Breiman, 2001; Zhang et al., 2010; Goldstein et al., 2011) and quickly becomes stable, one might consider the OOB error the better choice since it does not require to sacrifice data points. However, it has been known for a time that the OOB estimate can (even for forests with hundreds of trees and with thousands of instances) substantially deviate from the test performance (Bylander, 2002; Matthew et al., 2011; Janitza and Hornung, 2018). While literature mostly reports that OOB is too optimistic (Janitza and Hornung, 2018), we also could observe many cases in which the test performance was significantly *better* than the OOB estimate. So on large datasets it could be preferable to use a validation fold instead of an OOB estimate to create the performance curve.

In this paper, we stick to the OOB estimate despite its possible bias. Separating validation data can be unacceptable on small datasets, and providing a principled decision criterion for one of the two techniques based on the dataset at hand is not straight forward. Our experiments show that using the OOB estimate yields convincing results, but still this can be interesting future work.

3.2 Performance Curve Extrapolation

In this paper, we follow an extrapolation technique based on a maximum likelihood estimate for the four-parametric Morgan-Mercer-Flodin (MMF) model class, which has been used to model performance curves before in terms of sample size (Gu et al., 2001) or iterations of a neural network (Domhan et al., 2015). This class models a performance curve by

$$f(x) = \frac{ab + cx^d}{b + x^d} \tag{1}$$

where x in our case is the number of trees in the forest. Given a set of observations, the parameters $a, b, c, d \geq 0$ can be easily estimated with standard interpolation techniques such as the Levenberg-Marquardt method (Bard, 1974). Preliminary experiments showed that this class approximates the performance curve of RFs much more accurate than other common models like the inverse power law. Once the parameters are estimated, one can predict the performance for any specific forest size. If no limit is given, the best possible performance of the forest is estimated by $\lim_{x\to\infty} f(x) = c$.

3.3 The Training Algorithm

The training algorithm is simple and sketched in Alg. 1. It consists of two stages: In the first stage, a forest is trained for each tree type in sequence until a stopping criterion is met. The stopping criteria are (i) a stagnation of the performance curve, (ii) stability in the forecast created by the

Algorithm 1: SelfOptimizingRandomForest

```
p ← [];
M ← [];
foreach tree type t do
    m ← new RandomForest(t);
    while m.size()< L ∧ ¬m.is_stale() ∧¬m.stable_forecast() do
        m.train_and_add_tree()
    M.append(m);
    p.append(oob(F_t));
m ← M[arg max p];
while m.size()< L ∧ ¬m.is_stale() do
    m.train_and_add_tree()
return m
```

performance curve extrapolation, or (iii) that a possibly predefined maximum number of trees has been trained. Once the first stage is completed, the performance forecast is used to pick the tree type that is expected to perform best. The algorithm then adds trees to the forest of this tree type and updates the forecast until stagnation of the performance curve is detected or the upper limit of the number of trees is reached. In the following, we describe the stopping criteria more formally.

The stagnation stopping criterion $m.is\_stale()$ is based on two parameters. The parameter ε_{stag} controls by how much the accuracy of the forest must improve at least in order to keep training. However, the performance curves obtained from OOB estimates can exhibit a significant zig-zag behavior, because, in contrast to cross-validation, there is only one observation at each forest size. To reduce this effect, the check is not applied to the curve itself but a *smoothened* curve in which the performance at "size" k averages the values of the forest sizes $[k - w_{stag}, k]$. Observe that, since the increments in size are always 1, ε_{stag} is a threshold for the minimum *slope* of the (smoothened) performance curve required to continue learning.

Stability in the forecast of performance curves is considered to occur in a similar way based on windowing. In each iteration, the algorithm memorizes the expected performance of the considered tree type as a result of the performance curve extrapolation. These forecasts then generate a forecast history over time. $m.stable\_forecast()$ returns true iff the *variance* in the $w_{forecast}$ most recent forecasts is less than $\varepsilon_{forecast}$. In other words, if the prediction about the best possible performance of a forest type does not change substantially anymore, we consider the prediction model to be sufficiently accurate and rely on its prediction. Unless being configured with a very tight (small) $\varepsilon_{forecast}$, it will usually be the case that this criterion will apply prior to stagnation.

4 Experimental Evaluation

Our evaluation seeks to answer the following research questions:

1. How often are standard RFs outperformed by LDA RFs or PCA RFs?
2. How consistent are SORFs in their ability to identify the best forest type?
3. What is the regret of (limited) SORFs compared to the optimal RF type?
4. How much faster are (limited) SORFs compared to growing three forests?

Since this is an empirical evaluation, all insights and conclusions are limited to the datasets considered here.

4.1 Experiment Setup

To answer the above questions, we computed the voting scheme of RFs of the three tree types for forest sizes between 1 and 100 each on 96 datasets, all of which are available at openml.org (Vanschoren et al., 2013) for reproducibility. The 96 datasets are a mixture of datasets from the AutoML benchmark (Gijsbers et al., 2019) and previous works on Oblique RFs (Zhang and Suganthan, 2014). For technical reasons, we excluded datasets of the AutoML benchmark with a sparse data representation. The exact collection of datasets is described in the supplementary material, which also contains the algorithms' results on each of them individually.

For every dataset, 20 random train/test splits of 80%/20% were created. From these voting schemes, which are shared with the community, it is possible to recover the behavior of the three simple RFs (standard, LDA, PCA) as well as the behavior of SORFs. This procedure not only saved computational resources but also allows other researchers to reproduce the results without explicitly training the forests again. All reported performances are summarized test-fold performances.

The RFs were configured as follows. To enable better comparison, the maximum number of trees was set to 100 even though this hyperparameter is even optional for SORFs. A comparison of the trees under this condition is however beyond our scope. The purity parameter π was set to 0.9, and the minimum number of instances in a leaf η was set to 5. PCA RFs were configured to consider all of the dimensions (PCA simply used for rotation but not projection). The SORF hyperparameters were set to $w_{stag} = 10$, $\varepsilon_{stag} = 10^{-4}$, $w_{forecast} = 10$, and $\varepsilon_{forecast} = 10^{-3}$, which we would arguably consider reasonable default values.

The computations for this experimentation required 58.5 CPU days in a compute center with Linux machines, each of them equipped with 2.6Ghz Intel Xeon E5-2670 processors and 16GB memory. No timeouts were imposed when training the forests and computing the voting schemes.

4.2 Results

4.2.1 How often are standard RFs outperformed by LDA RFs or PCA RFs?. We are not the first to provide an answer to this question. Zhang et al. (Zhang and Suganthan, 2014) have reported similar results. However, since we could not reproduce their results on ensemble forests, we aimed at verifying that ORFs indeed can substantially improve the performance over standard RFs.

An answer to this question can be found in Fig. 1. The left figure compares the performances of a standard RF with the LDA and PCA RF respectively. Every point represents one dataset: blue points compare the standard RF against an LDA RF, and orange points compare it against an PCA RF trained on the same dataset. The scores are test set accuracies averaged over the 20 random train/test splits of size 80%/20%.

The plot suggests that none of the tree types is dominant. On many datasets, the standard RF without any rotation outperforms both the LDA RF and the PCA RF (points below the line). So none of the two alternatives dominates the standard RF. On the other side, for both RF types LDA RF and PCA RF there are datasets on which they perform substantially better than a standard RF. The most extreme example is the HILL-VALLEY dataset (1566) on which the two rotation based forest types achieve virtually a 100% test accuracy while the standard RF only achieves a 60% accuracy.

While the presentation of results in the scatter plot suggests that LDA RF and PCA RF only occasionally are the best choice, the other two plots in Fig. 1 show that indeed the standard RF is the best choice in only approximately a little less than 50% of the cases. In fact, LDA RF is the best choice in 29% of the cases an PCA RF in 23% of the cases. Here "cases" refers to splits of datasets, because on some splits of a dataset one algorithm might work better than on a different split. With these results, there is a clear motivation to identify the best forest type for a given dataset.

4.2.2 How consistent are SORFs in their ability to identify the best forest type?. This question is answered by the the middle and right plot of Fig. 1, now also considering the actual decisions of the SORF (y-axes). Summing up the probabilities on the diagonal of the confusion matrix, we observe

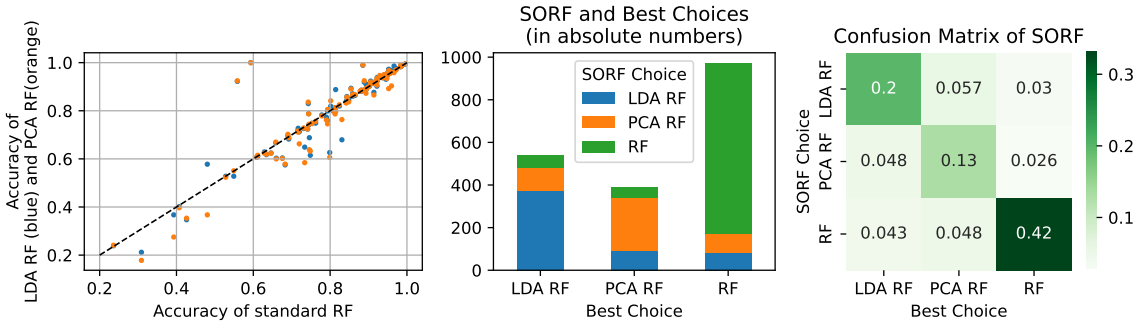

Figure 1: Left: Comparison of the accuracies of standard RFs vs LDA RFs (blue) and PCA RFs (orange). Middle and Right: Best possible RF type choice and the type choices made by a SORF.

that SORFs identify the correct tree type in only 75% of the cases. The middle plot reflects this by rather impure bars in which the bar for every best choice contains the correct choice in the majority but certainly not all of the cases.

The explanation for this supposedly disappointing result is that, on many datasets, there is only very little difference between the algorithm performances and the best one is only very marginally better than the other or at least the second best one. In such cases, it is possible that the forecast of the performance curve is not precise enough to identify the best algorithm.

Interestingly, the more intuitive explanation for these results, namely the mediocre quality of the out-of-bag (OOB) estimate, is not a relevant factor here. Clearly, since the OOB does not accurately resemble the test error on many datasets, one would intuitively expect that this misguidance leads to wrong decisions. However and perhaps surprisingly, considering the same confusion plots not with the best choice on the test data but the best choice according to the OOB estimate, the results look worse: the SORF then identifies the best tree type (according to OOB) in only 64% of the cases.

Based on these observations, the next research question about the regret becomes even more urgent. Apparently SORFs are not able to consistently identify the best tree type. So the question is how much they are outperformed by the indeed best single RF type in the situation in which they don't pick the best type.

4.2.3 What is the regret of (limited) SORFs compared to the optimal RF type?.

To analyze this question, Fig. 2 shows the performance difference between SORFs and each of the three RF types. The left plot compares the accuracy of SORFs (x-axis) against the other three tree type RFs. Blue points show the comparison to LDA RFs, orange points to PCA RFs, and green points to standard RFs. Every point is the average accuracy of the two compared algorithms on a single dataset across the 20 train/test splits. Bullets show statistically significant differences according to a Wilcoxon signed rank test (p-value 0.05), and circles show datasets on which the difference is not significant in this sense. To give a different perspective on the same data, the right plot summarizes the *improvements* in accuracy the SORF achieves over each of the three individual forest types. For each of them, there is one boxplot summarzing the improvements over *all* the datasets, and one only about those on which the mean value could be identified to be statistically significantly different (same test as for left plot). Besides, there is one boxplot showing the improvement over the "oracle", which is effectively the regret of the SORF since the oracle picks the best RF type and hence cannot be improved upon. Three vertical dotted lines serve as visual aids for a deterioration of 0.01 (red line), and improvements of 0.01 and 0.03 (black lines). The vertical dashed line is simply the 0 improvement (no improvement but also no regret).

These results now show SORFs in a much more positive light than the plots in Fig. 1. There are almost *no* datasets in which SORFs significantly underperform the best RF type for that dataset

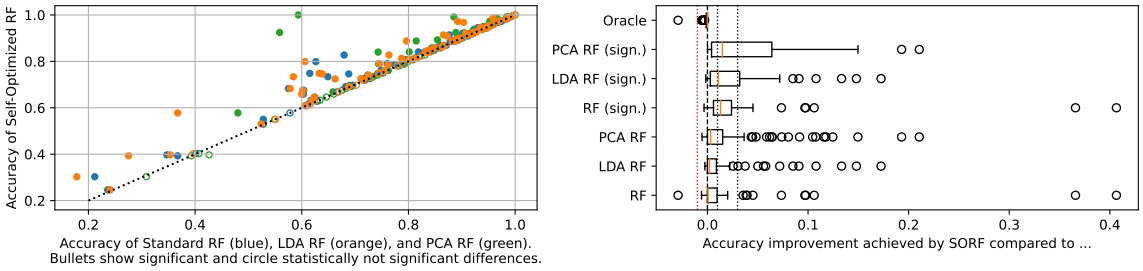

Figure 2: Left: Comparison of accuracies of a SORF compared to standard RFs (green), LDA RFs (blue) and PCA RFs (orange). Right: Summary of improvements of the SORF over the RF types on the different datasets. The boxplots labeled with "sign." only summarize the datasets on which the difference of the mean performance is statistically significant. The oracle entry refers to the comparison to the best RF type on a dataset (hence always negative).

(points below the line in the left plot). In fact, on only one dataset a deterioration of more than 0.01 in accuracy can be observed, and this deterioration is not statistically significant. Among the statistically significantly deviating results, the regret is always under 0.007, and in over 70% of the cases there is no regret at all since SORF obtains the optimal performance among all tree types.

From a different viewpoint, the regret of the standard RF is above 0.01 in 25% of the cases. This can be seen in the bottom boxplot of the right plot, which shows that the 75%-quantile is exactly at 0.01. In other words, we should not expect SORFs to significantly outperform standard RFs all too often, but it *can* happen, and in such situations the improvement is often not only statistically significant but also quite substantial in terms of absolute improvement.

All this being said, we conclude that SORFs have negligible regret when being compared to an oracle that picks the best forest type. The average regret is 0.001, and the maximum statistically significant regret is below 0.01. In other words and putting this into the light of the discussion in Sec. 4.2.2, SORFs not always identify the best forest type, but in the case where they don't, the chosen alternative is almost equally good. The regret here is the price one has to pay getting the runtime advantages of SORFs in contrast to training all forest types and choosing the best one. We next assess precisely this runtime advantage.

4.2.4 How much faster are (limited) SORFs compared to growing three forests?.

To analyze the training time behavior, Fig. 3 relates the numbers of trained trees and the actual training times of SORFs and the individual RF types in several ways. The panel with the two left plots shows the relative training time compared to the full forest of the respective types (left plot) and the number of trees grown for each of the types on average based on the chosen forest type (middle left plot). While the left panel only shows mean values averaged over the datasets, the panel with the two boxplots on the right summarizes the relative reduction in training time (middle right plot) or numbers of trained tree (right plot) per tree type without taking the chosen tree type into account. Again, it is important to note that the SORFs were limited to a forest size of 100 for the finally chosen tree type. Without this restriction, some forests would have built more trees requiring more training time while possibly leading to better results. This analysis is beyond our scope.

Looking first at the left plots, we can observe a substantial runtime reduction on average. The fourth column of the first plot shows that SORFs exhibit a training time that is roughly only 40% of the time required to fully train all three forest types with 100 trees and then pick the best one. It also shows that the eventual choice of the tree types barely affects this number even though PCA RFs have a significantly higher runtime than the other two forest types on average (not shown in the figure). Consistently, the middle left plot shows that the number of trained trees is only 40% of the trees that would be trained otherwise, i.e. approximately 120 instead of 300. What is

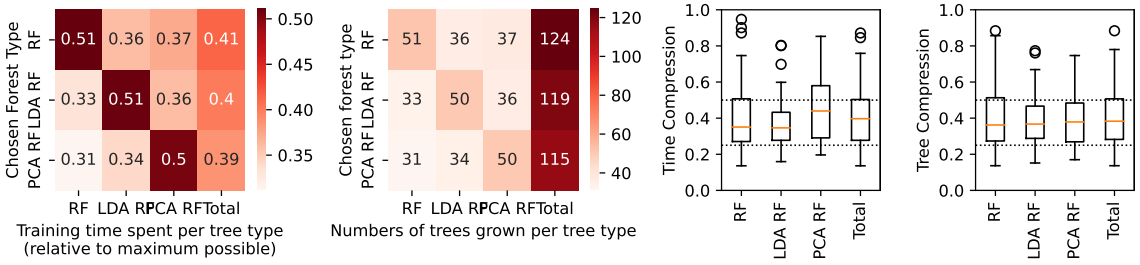

Figure 3: Left: Mean training time compression in SORFs for different tree types depending on the finally chosen tree type. Middle left: Mean number of trees trained per tree depending on the finally chosen tree type. Middle and outer right: Mean reduction per dataset in training time/trained trees per tree type and in total summarized over all datasets.

particularly interesting is that, on average, even for the chosen tree type only roughly 50 instead of 100 trees are trained. This suggests the usefulness of the automated stopping criterion applied by SORFs, which could of course also be used independently by the RF training algorithms. Note that the actual numbers of trained trees of the selected type ranges between 10 and 100, and the fact that it is on average close to 50 is only a coincidence.

The right panel offers a more detailed view on the range of the concrete reductions. They summarize across the datasets, for each tree type but also in total, how much the training time or number of trees of that tree type is reduced on average on the respective dataset. For both boxplots it shows two visual aids at 50% and 25% through the dotted lines. Remarkably, we can see that the compression of both time and trained trees is better (below) than 50% in more than 75% of the cases. In roughly 25% of the analyzed datasets, these quantities are even reduced by 75% or more.

Putting everything together, we can conclude, with the usual limitation of a limited number of datasets used for analysis, that SORFs achieve the same performance as a portfolio consisting of three RF types while requiring substantially less time for training. Thanks to the projection of the performance curves and early stopping based on the detection of stale training progress, the training time can be significantly reduced while sacrificing only a marginal portion of accuracy. The trivial approach of considering full RFs for all of the types seems only justified if the user is concerned about accuracy differences on an order that is below 0.01 in accuracy.

5 Conclusion

In this paper, we have proposed Self-Optimizing Random Forests (SORFs). SORFs build sub-forests with different tree types up to a size in which the final performance of each type can be safely predicted based on a performance curve model. The experimental evaluation over 96 datasets shows that SORFs with three tree types (standard, with LDA rotations, and with PCA rotations) exhibit optimal performance compared to a portfolio of RFs of these tree types while requiring only 40% of the training time on average. While SORFs can be seen as a RF-centered AutoML approach itself, the results suggest to replace standard RFs in AutoML tools by SORFs since they cover a broader class of learners than standard RFs with potentially significant performance improvements while exhibiting a much lower training time compared to adding each of these RF types separately to the portfolio of the AutoML tool.

There are plenty of options for immediate future work. One straight forward question is on the potential improvements of SORFs if no tree limit is imposed. A second question is on the evaluation mechanism and whether the usage of a validation set instead of using the OOB estimate can improve the performance or runtime of the SORF. Finally, analyzing the prediction performance of *mixed* forests that consist of trees of *different* types would be interesting, because this could reduce the training time even more while adding more diversity to the ensemble.

References

Bard, Y. (1974). *Nonlinear parameter estimation.* Academic Press.

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

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Bylander, T. (2002). Estimating generalization error on two-class datasets using out-of-bag estimates. *Mach. Learn.*, 48(1-3):287–297.

Cortes, C., Jackel, L. D., and Chiang, W. (1994). Limits in learning machine accuracy imposed by data quality. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*, pages 239–246. MIT Press.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3460–3468. AAAI Press.

Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., and Vanschoren, J. (2019). An open source automl benchmark. *CoRR*, abs/1907.00909.

Goldstein, B. A., Polley, E. C., and Briggs, F. B. (2011). Random forests for genetic association studies. *Statistical applications in genetics and molecular biology*, 10(1).

Gu, B., Hu, F., and Liu, H. (2001). Modelling classification performance for large data sets. In Wang, X. S., Yu, G., and Lu, H., editors, *Advances in Web-Age Information Management, Second International Conference, WAIM 2001, Xi'an, China, July 9-11, 2001, Proceedings*, volume 2118 of *Lecture Notes in Computer Science*, pages 317–328. Springer.

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

Janitza, S. and Hornung, R. (2018). On the overestimation of random forest's out-of-bag error. *PloS one*, 13(8):e0201904.

Lemmond, T. D., Hatch, A. O., Chen, B. Y., Knapp, D., Hiller, L., Mugge, M., and Hanley, W. G. (2008). Discriminant random forests. In Stahlbock, R., Crone, S. F., and Lessmann, S., editors, *Proceedings of The 2008 International Conference on Data Mining, DMIN 2008, July 14-17, 2008, Las Vegas, USA, 2 Volumes*, pages 55–61. CSREA Press.

Matthew, W. et al. (2011). Bias of the random forest out-of-bag (oob) error for certain input parameters. *Open Journal of Statistics*, 2011.

Menze, B. H., Kelm, B. M., Splitthoff, D. N., Köthe, U., and Hamprecht, F. A. (2011). On oblique random forests. In Gunopulos, D., Hofmann, T., Malerba, D., and Vazirgiannis, M., editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II*, volume 6912 of *Lecture Notes in Computer Science*, pages 453–469. Springer.

Quinlan, J. R. (1997). Decision trees and instance-based classifiers. In Tucker, A. B., editor, *The Computer Science and Engineering Handbook*, pages 521–535. CRC Press.

Rodríguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1619–1630.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2013). Openml: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60.

Wang, F., Wang, Q., Nie, F., Yu, W., Wang, R., and Li, Z. (2020). A forest of trees with principal direction specified oblique split on random subspace. *Neurocomputing*, 379:413–425.

Zhang, G., Zhang, C., and Zhang, J. (2010). Out-of-bag estimation of the optimal hyperparameter in subbag ensemble method. *Commun. Stat. Simul. Comput.*, 39(10):1877–1892.

Zhang, L., Ren, Y., and Suganthan, P. N. (2014). Towards generating random forests via extremely randomized trees. In *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, pages 2645–2652. IEEE.

Zhang, L. and Suganthan, P. N. (2014). Random forests with ensemble of feature spaces. *Pattern Recognit.*, 47(10):3429–3437.

Reproducibility Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   Yes

   (b) Did you describe the limitations of your work?
   Yes, we explicitly state that the conclusions are limited to the considered datasets.

   (c) Did you discuss any potential negative societal impacts of your work?
   No, because we think that this does not apply here. None of the points in the ethic guidelines applies to our paper.

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them?
   Yes.

   (e) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)?
   Yes, included in the supplementary material.

   (f) Did you include the raw results of running the given instructions on the given code and data?
   Yes, the raw results are part of the repository coming along with this paper.

   (g) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given?
   Yes, the repository contains the python notebook that was used to generate all the figures and result tables.

   (h) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented?
   We hope so.

(i) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)?
Yes, in parts this is described in the paper. The exact technical details (how the train/test splits were created) at the implementation level are part of the code repository.

(j) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code?
Yes

(k) Did you run ablation studies to assess the impact of different components of your approach?
We believe that this does not apply to our paper since SORFs do not have several components. One could, maybe, evaluate SORFs for different configurations, but this was beyond our scope and probably not yield relevant insights.

(l) Did you use the same evaluation protocol for the methods being compared?
Of course.

(m) Did you compare performance over time?
Does not apply. But yes we also considered the aspect of time.

(n) Did you perform multiple runs of your experiments and report random seeds?
Yes. The random seeds are documented in the code base and not releveant in the paper itself.

(o) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? No, but we applied a significance test.

(p) Did you use tabular or surrogate benchmarks for in-depth evaluations?
Effectively, we created our own benchmark for this problem, which can be re-used by other researchers.

(q) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? Yes.

(r) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? Does not apply since we did not tune parameters.

2. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators?
Yes, in the supplement.

(b) Did you mention the license of the assets?
Yes, in the supplement.

(c) Did you include any new assets either in the supplemental material or as a URL? Yes. The implementation of SORFs.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating?
We only use data from openml.org where consent is part of the upload policy.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content?
Does not apply.