Randomer Forests

Tyler M. Tomita, Mauro Maggioni, Joshua T. Vogelstein

Abstract

Random forest (RF) has been shown to outperform many other classifiers on a variety of datasets, yet its restriction of recursive partitions of the feature space to be axis-aligned is suboptimal in many cases. Several studies have proposed "oblique" decision forest methods to address this limitation. However, these methods either aren't well-adapted to problems in which the number of irrelevant features are overwhelming, have a time and space complexity significantly greater than RF, or require additional hyperparameters to be tuned, rendering training of the classifier more difficult. In this work, we establish a generalized forest building scheme, random projection forests. Random forests and many other currently existing decision forest algorithms can be viewed as special cases of this scheme. With this scheme in mind, we propose a special case which we call randomer forests (RerFs). We demonstrate that RerF emprically outperforms RF and another recently proposed oblique forest method in simulations and on benchmark data, and also show that it scales comparably to RF in terms of time and space complexity. Lastly, we propose two augmentations to RerF with the goal of making it more robust to affine transformations and outliers. We conclude that RerF is a competitive alternative to RF and other oblique forests. Open source code will be available.

I Introduction

Data science is becoming increasingly important as our ability to collect and process data continues to increase. Supervised learning—the act of using predictors to make a prediction about some target data—is of special interest to many applications, ranging from science to government to industry. Classification, a special case of supervised learning, in which the target data is categorical, is one of the most fundamental learning problems, and has been the subject of much study. A simple pubmed search for the term "classifier" reveals nearly 10,000 publications, and a similar arXiv search reports approximately 1000 hits. One of the most popular and best performing classifiers is random forests (RFs) (3). Two recent benchmark papers assess the performance of many different classification algorithms on many different datasets (4, 6), and both concluded the same thing: RFs are the best classifier.

RF typically refers to Breiman's Forest-IC, which uses axis-parallel (7), or orthogonal trees (10). That is, the feature space is recursively split along directions parallel to the axes of the feature space. Thus, in cases in which the classes seem inseparable along any single dimension, RF may be suboptimal. To address this, Breiman also proposed and characterized Forest-RC, which used linear combinations of coordinates rather than individual coordinates, to split along. Others have studied several different variants of "oblique" decision forests, including efforts to learn good projections (7, 12), using principal components analysis to find the directions of maximal variance (8, 11), or directly learning good discriminant directions (10). Another recently proposed method, called random rotation RF, uniformly randomly rotates the data for every decision tree in the ensemble prior to inducing the tree (2). While all of these recent approaches deal with rotational invariance, they fail to address two important issues. First, in real world data, the proportion of features that are irrelevant is often large, especially for high dimensional data, giving rise to optimal decision boundaries that are sparse. In such cases, we say that the signal is *compressive*. To our knowledge, all of the proposed oblique decision forests, with the exception of Forest-RC if tuned appropriately, do not impose a constraint on the sparsity of recursive partitions of the feature space. Therefore, such methods may be suboptimal in cases in which the signal is compressive. Second, as our ability to collect and process data continues to increase, we are encountering increasingly massive datasets. Therefore, time- and space-efficient methods are becoming more and more important. With the exception of Forest-RC, all oblique methods require expensive computation and/or storage. There is therefore a gap that calls for the construction of a scalable oblique decision forest classifier that can perform well in the presence of an overwhelming number of irrelevant features. It is true that Forest-RC fits this description. However, the number of features that are linearly combined to generate candidate splits is fixed across all nodes of every tree. Such a construction requires an additional hyperparameter to be tuned, and also limits the diversity of trees.

To address this, we first state a generalized forest building scheme, random projection forests, which includes all of the above algorithms as special cases. This enables us to formally state our objective, and provides a lens that enables us to propose a few novel special cases, which we refer to as "*Randomer Forests*" (or RerFs for short), for reasons that will become clear in the sequel. We both theoretically and empirically demonstrate that our methods have the same time and space complexity as random forests, and show on simulated datasets that it performs well in the face of many irrelevant features and when axis-aligned splits are suboptimal. Additionally, we demonstrate that our method empirically outperforms both RF and random rotation RF (RotRF), which is another oblique method (2), on a suite of benchmark datasets. Lastly, we propose two augmentations to RerF for making the method more robust to affine transformations and outliers. To conclude, we propose our method as a competitive alternative to RF and other oblique decision forest methods. Open source code will be made available.

II Random Projection Forests

Let $\mathcal{D}^n = \{(X_i, y_i) : i \in [n]\}$ be a given dataset, where $X_i \in \mathbb{R}^p$ and $y_i \in \mathcal{Y} = \{\widetilde{y}_1, \ldots, \widetilde{y}_C\}$. A classification forest, $\bar{g}(X; \mathcal{D}^n)$ is an ensemble of L decision trees, each tree $g^l(X; \mathcal{D}^l)$ is trained on a (sub)set of the data, $\mathcal{D}^l \subset \mathcal{D}^n$). Random Projection forests are a special case of classification forests that subsume all of the strategies mentioned above (see Pseudocode 1). The key idea of all of them is that at each node of the tree, we have a set of predictor data points, $\bar{X} = \{X_s\}_{s \in \mathcal{S}^l_{ij}} \in \mathbb{R}^{p \times S^l_{ij}}$, where $S^l_{ij} = |\mathcal{S}^l_{ij}|$ is the cardinality of the set of predictor data points at the $(ij)^{th}$ node of the l^{th} tree. We sample a matrix $A \sim f_A(\mathcal{D}^n)$, where $A \in \mathbb{R}^{p \times d}$, possibly in a data dependent fashion, which we use to project the predictor matrix $\bar{X}$ onto a lower dimensional subspace, yielding $\widetilde{X} = A^\mathsf{T} X \in \mathbb{R}^{d \times S^l_{ij}}$, where $d \leq p$ is the dimensionality of the subspace. To wit, Breiman's original Forest-IC algorithm can be characterized as a special case of random projection forests. In particular, in Forest-IC one constructs A such that for each of the d columns, we sample a coordinate (without replacement), and put a 1 in that coordinate, and zeros elsewhere. Breiman's Forest-RC constructs A by sampling a fixed number of coordinates (without replacement) for each of the d columns, and puts a value uniformly sampled from [-1,1] in each of those coordinates. Similarly, Ho's rotation forests construct A from the top d principal components of the data $\bar{X}$ at a given node. Thus, the key difference in all these approaches is the choice of f_A.

Pseudocode 1 Psuedocode for Random Projection Forests, which generalizes a wide range of previously proposed decision forests.

Input: data: $\mathcal{D}^n = (X_i, y_i) \in (\mathbb{R}^p \times \mathcal{Y})$ for $i \in [n]$, tree rules (stopping criteria, rules for sampling data points per tree, etc.), distributions on $d \times p$ matrices: $A \sim f_A(\mathcal{D}^n)$, preprocessing rules
Output: decision trees, predictions, out of bag errors, etc.
Preprocess according to rule
for each tree **do**
  Subsample data to obtain $(\bar{X}, \bar{y})$, the set of data points to be used in this tree
  for each leaf node in tree **do**
    Let $\widetilde{X} = A^\mathsf{T} \bar{X} \in \mathbb{R}^{d \times s}$, where $A \sim f_A(\mathcal{D}^n)$
    Find the "best" split coordinate k^* in $\widetilde{X}$ and "best" split value $t^*(k^*)$ for this coordinate
    Split X according to whether $X(k) > t^*(k^*)$
    Assign each child node as a leaf or terminal node according to stopping criteria
  end for
end for
Prune trees according to rule

The goal of this work is to find a random projection forest that shares RF's scalability and ability to perform well when many irrelvant predictors are present, yet is able to find decision boundaries that are less biased by geometrical constraints (i.e. not constrained to be axis-aligned), by changing the distribution f_A. The result we call randomer forests (or RerFs for short).

III Randomer Forests

Our choice in f_A is based on the following three ideas:

1. While RF empirically performs well in many settings, it is quite restrictive in that candidate splits evaluated at each node are constrained to be axis-aligned. It is sometimes the case that axis-aligned splits are suboptimal, and oblique splits may be desired.

2. It is often the case that the number of *relevant* features is substantially smaller than the number of total features. We say that the signal is *compressive* in such cases. RF often does well when the signal is compressive, which may be attributed, in part, to the extreme sparsity constraint on the columns of A. All of the oblique decision forest algorithms to date, with the exception of Forest-RC, do not impose a sparsity constraint on the candidate split directions, which may cause such algorithms to underperform when the signal is compressive.

3. Except for Breiman's Forest-RC, existing oblique decision forest algorithms involve expensive computations to identify and select splits, rendering them less space and time efficient than Forest-IC or Forest-RC. An oblique decision forest having a space and time complexity comparable to Forest-IC or Forest-RC is desirable.

To this end, we employ very **sparse random projections** (9). Rather than sampling d non-zero elements of A and enforcing that each column gets a single non-zero number (without replacement) as RF does, we relax these constraints and select d non-zero numbers from $\{-1, +1\}$ with equal probabilities and distribute uniformly at random in A.
It is apparent that RerF bares a resemblance to Forest-RC, yet there is one key difference. In Forest-RC, the number of nonzeros in each column of A is fixed to be the same across A and for every split node, and its optimal value has to be found through parameter selection. Our construction circumvents selection of this parameter by randomizing the number of nonzeros in each column of A. Furthermore, our algorithm allows A to have columns of varying sparsity, which may promote more diversity in the ensemble. To our knowledge, this property does not exist in any of the proposed random projection forest algorithms. Lastly, we note that our construction of A preserves distances between points with high probability (9).

IV Experimental Results

IV.A Simulations Involving Compressive Signals

Many classification problems arise in which the signal is compressive and the optimal split directions are not axis-aligned. We constructed two synthetic datasets with both of these properties to compare classification performance and training time of RF, RerF, and RotRF:
Sparse parity is a variation of the noisy parity problem. The noisy parity problem is a multivariate generalization of the noisy XOR problem and is one of the hardest constructed binary classification problems. In the noisy parity problem, a given sample has a mean whose elements are Bernoulli samples with probability 1/2, and then Gaussian noise is independently added to each dimension with the same variance. A sample's class label is equal to the parity of its mean. Sparse parity is an adaption of this problem in which the sample's class label is equal to the parity of only the first $p*$ elements of the mean, rendering the remaining $p - p*$ dimensions as noise.
Trunk is a well-known binary classification in which each class is distributed as a p-dimensional multivariate Gaussian with identity covariance matrices (13). The means of the two classes are $\mu_1 = (1, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{3}}, ..., \frac{1}{\sqrt{p}})$ and $\mu_2 = -\mu_1$. It follows from this that the signal-to-noise ratio of the ith dimension asymptotically decreases to zero with increasing i.
RotRF is an oblique decision forest method that has been recently proposed in (2). RotRF works exactly like RF, except that the feature space is uniformly randomly rotated prior to inducing each decision tree. This induces oblique splits in an unstructured way, which the authors argue is favorable to structured methods such as in Rodriguez's rotation forest, because such structured methods reduce the diversity of the trees. Uniformly random rotations of the feature space imply that in general, splits will not be sparse. Therefore, we conjecture that RotRF will perform increasingly poorly as the ratio of the number of irrelevant features

to the number of relevant features becomes larger, while RF and RerF will be relatively more robust to the increasing presence of irrelevant features.
Figure 1 depicts the true class posteriors in the first two dimensions of the sparse parity simulation in panel A and estimates of the posteriors for RF, RerF, and RotRF in panels B, C, and D, respectively. For this simulation, $p = 20$, $p* = 3$, and $n = 1000$, where p is the total number of dimensions, $p*$ is the number of relevant dimensions, and n is the number of sampled data points. The number of trees used for all three algorithms was 1000. Various values of d, the number of columns in the random projection matrix A, were tried and the best for each algorithm was selected. RerF gives estimates of the posteriors closest to the true posteriors.

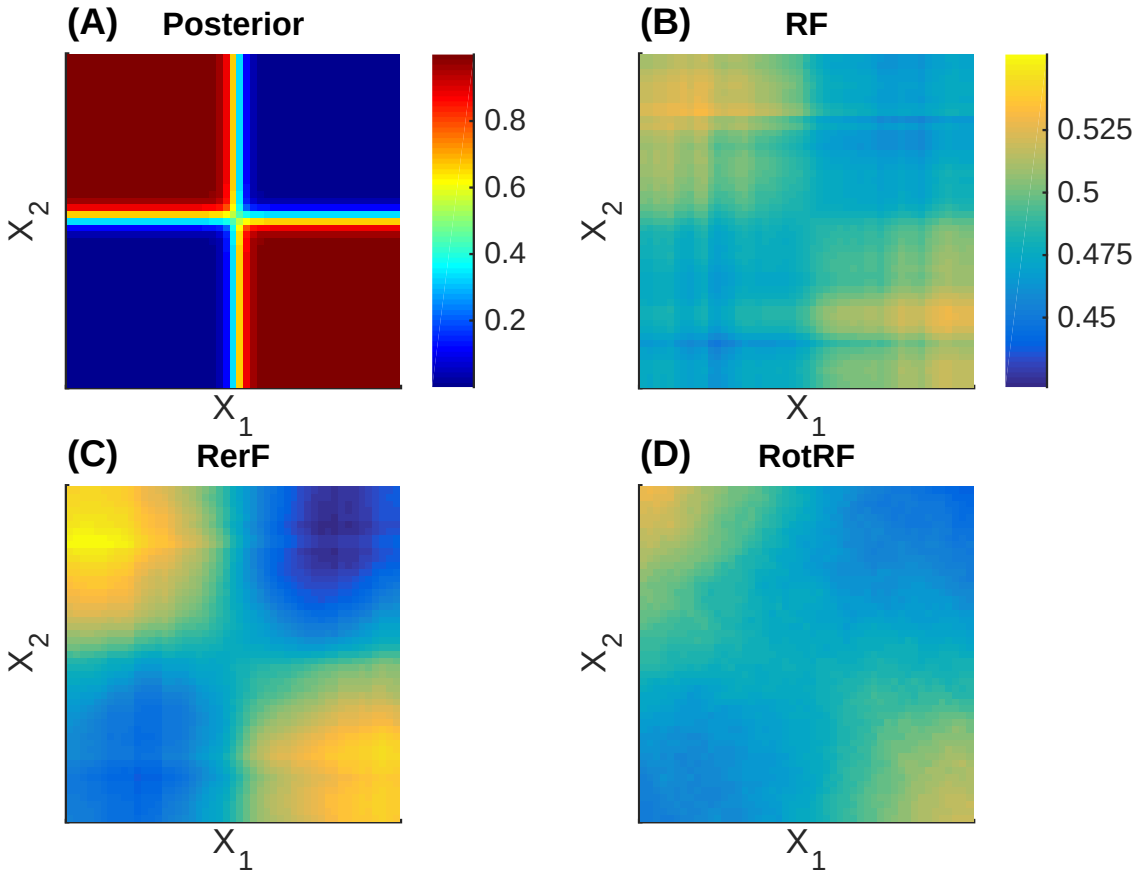

Figure 1: Class posteriors for the sparse parity problem in the first two dimensions (refer to section IV.A for details). (A): True posteriors. (B) - (D): Posterior estimates for RF, RerF, and RotRF, respectively. Comparing panels B-D with A, it is evident that RerF produces better estimates of the posteriors than does RF or RotRF

The left panels of Figure 2 show two-dimensional scatter plots from each of the two example simulations (using the first two coordinate dimensions). The middle panels show the misclassification rate relative to RF against the number of observed dimensions p, for RerF and RotRF. Relative misclassifcation rate was computed as the difference between the misclassification rate of either RerF or RotRF and that of RF. The misclassification rate of RF relative to itself is shown for reference. The right panels show training time against the number of observed dimensions p for all four classifiers. For all comparisons, we used the same number of trees for each method to enable a fair comparison. The number of trees used for each method in the sparse parity and Trunk simulations were 500 and 1000, respectively. In all methods, trees were

pruned and the minimum number of data points at a node in order to be considered for splitting was 10. Gini impurity was used as the split criteria. The only parameter tuned was d, the number of candidate split directions evaluated at each split node. When $p \leq 5$, each classifier was trained for all $d \in [p]$. When $p > 5$, each classifier was trained for all $d \in \{1, p^{1/4}, p^{1/2}, p^{3/4}, p\}$. Misclassification rates for each classifier were selected as the lowest achieved from the different values of d tried. Training times for each classifier were computed by averaging the training times given by all values of d tried. For sparse parity, n was fixed at 1000 and classifiers were evaluated for $p \in \{2, 5, 10, 25, 50, 100\}$. The relevant number of features $p*$ was fixed at a value of 3. For Trunk, n was fixed at 100 and RF and RerF were evaluated for $p \in \{2, 10, 50, 100, 500, 1000\}$. RotRF was not evaluated for $p = 1000$ due to computational burden. Relative error and training times plotted are the average of 25 repeated trials, and error bars represent the standard error of the mean.
In panel B, RerF performs as well as or better than both RF and RotRF for all values of p. RotRF performs as well as or better than RF except for when $p = 25$. As conjectured, RotRF performs better than RF when p is small because oblique splits provide an advantage over axis-aligned splits in the sparse parity problem. As p increases and the ratio $p*/p$ decreases, RerF begins to outperform RotRF. Ultimately, when this ratio is small enough, RotRF performs even worse than RF. RerF's ability to perform relatively well can be attributed to the sparsity of oblique splits. In panel E, RerF outperforms RF for all values of p. This is because linear combinations of a few features can yield a higher signal-to-noise ratio than any single feature. RotRF outperforms both RF and RerF up to $p = 100$. RotRF is able to perform better than RerF in these cases because a larger number of features can be linearly combined to yield an even higher signal-to-noise ratio. When $p = 500$, classification performance of RotRF significantly degrades and becomes worse than both RF and RerF. This can be explained by the fact that when p is large enough, RotRF often samples linear combinations of many features each having a low signal-to-noise ratio. Such projections will yield a lower signal-to-noise ratio than any single feature. In panels C and F, training times are comparable when p is small. As panel F indicates, training time of RotRF is significantly longer than those of RF and RerF when $p = 500$.

IV.B Theoretical Space and Time Complexity

For a RF, assume there are L trees. If there are n data points per tree, and each tree grows until terminal nodes have only $\mathcal{O}(1)$ data points with p coordinates in them, there are $\mathcal{O}(n)$ nodes. Then the complexity of constructing the random forest, disregarding cross-validation or other randomization techniques for preventing overfitting, is $\mathcal{O}(Ln^2 p \log n)$. In practice the trees are shallower and stop much earlier than when nodes have $\mathcal{O}(1)$ points, so "in practice" the complexity often appears to be $\mathcal{O}(Lnp \log n)$. Storing RF requires $\mathcal{O}(Ln \log n)$ because each node can be represented by the index of the nonzero coordinate. The only additional space constraint is storing which indices are positive, and which are negative, which is merely another constant.
RerF has a very similar time and space complexity, unlike many of the other oblique random forest variants. Specifically, assume that RerF also has L trees, and n data points per tree, and no pruning, so $\mathcal{O}(n)$ nodes. Let n_k be the number of data points at node k of the tree. Like RF, RerF requires $\mathcal{O}(p)$ time to sample p non-zero numbers, and $\mathcal{O}(pn_k)$ time to obtain the new matrix, $\widetilde{X}$, because it is a sparse matrix multiplication, in node k with $\mathcal{O}(n_k)$ points. RerF also takes another $\mathcal{O}(p/n \log(p/n)) = \mathcal{O}(1)$ time to find the best dimension. Thus, in total, in the case of well-balanced trees, RerF also requires only $\mathcal{O}(Lpn^2 \log n)$ time to train. To store the resulting RerF, like RF, requires $\mathcal{O}(Ln \log n)$, because each node can be represented by the indices of the coordinates that received a non-zero element, and the expected number of such indices is $\mathcal{O}(1)$. Note that these numbers are in stark contrast to other oblique methods. RotRFs, in particular, require a QR decomposition having a time complexity of $\mathcal{O}(p^3)$ in order to generate random rotation matrix for each tree. Rotating the data matrix prior to inducing each tree additionally requires $\mathcal{O}(np^2)$. Therefore, RotRF becomes very expensive to compute when p is large. This can explain the trend seen in Figure 2(F). In addition to storing all of the decision trees, RotRF has to store a rotation matrix for each tree, which requires $\mathcal{O}(p^2)$.

IV.C Effects of Transformations and Outliers

We next want to compare the robustness of RF, RerF, and RotRF to various data transformations. To do so, we consider several different modifications to the simulation settings described in the previous section:

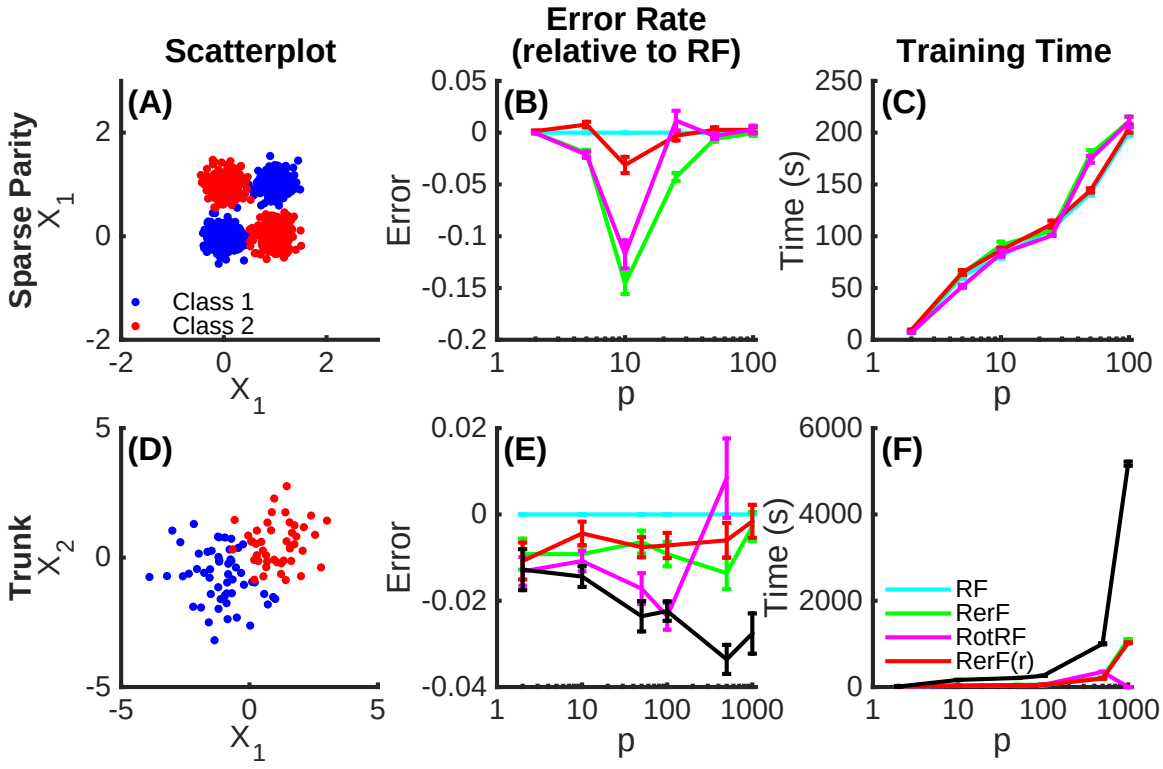

Figure 2: Sparse parity (A-C) and Trunk (D-F) simulations (see section IV.A for details). (A) and (D): Scatterplots of sampled points in the first two dimensions. (B) and (E): Error rates of RerF and RotRF relative to RF across different values of p. (C) and (F): Same as (B) and (E) except absolute training time is plotted on the y-axis instead. The cyan, green, and magenta lines correspond to RF, RerF, and RotRF, respectively. This color coding is adopted in figures 3 and 4 that follow. Both RerF and RotRF do better than RF when the number of irrelevant features is sufficiently small, due to their ability to generate oblique parititions. However, when the number of irrelevant features becomes large enough, performance of RotRF rapidly degrades. Training times show that RerF scales identically with RF, while RotRF scales poorly with large p (note that in panels E and F, RotRF is only plotted up to $p = 500$ due to computational constraints.

rotation, scale, affine, and outliers. To rotate the data, we simply generate rotation matrices uniformly and apply them to the data. To scale, we applied a scaling factor sampled from a uniform distribution on the interval [0,10] to each dimension. Affine transformations were performed by applying a combination of rotations and scalings as just described. Additionally, we examined the effects of introducing outliers. Outliers were introduced by sampling points from the distributions as previously described but instead using covariance matrices scaled up by a factor of four. Empirically, an addition of 20 points from these outlier models to the original 100 points was found to produce a noticeable but not overwhelming effect on classifier performance.

Figure 3 shows the effect of these transformations and outliers on the sparse parity (panels A-E) and Trunk (panels F-J) problems. Panels A-E show that RerF and RotRF outperform RF in both untransformed and transformed representations for Sparse Parity. This makes sense because linear combinations of variables can have some information about the class, whereas the original features are all uninformative on their own. For the trunk simulations, panel G shows that RotRF is more robust to rotations (as it should be), but is fragile in the face of affine transformations.

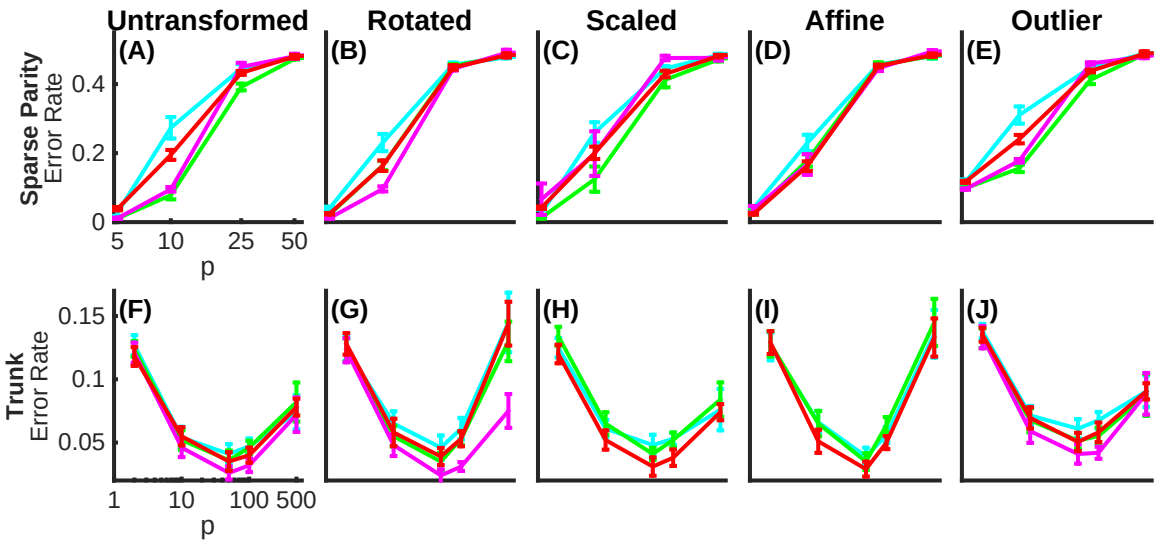

Figure 3: The effects of different transformations applied to the sparse parity (A-E) and Trunk (F-J) simulations on classification performance (see section IV.C for details). Specifically, we consider rotations, scalings, affine transformations, as well as the addition of outliers. RerF and RotRF outperform RF on sparse parity. The trunk simulations demonstrate clearly that RotRF, while invariant to rotation, is sensitive to affine transformations.

IV.D Benchmark Data

In addition to the simulations, RF, RerF, and RotRF were evaluated on 115 of the 121 datasets as described in (6). The six remaining datasets were not used because their high dimensionality and large number of data points rendered the classifiers both time and space costly, particularly for RotRF. As in the previous section, transformations, with the exception of outliers, were applied to the datasets to observe their affects on performance of the three algorithms. Classifiers were trained on the entire training sets provided. For each data set, misclassification rates were again estimated by out of bag error. The number of trees used in each algorithm was 1000 for datasets having at most 1000 data points and 500 for datasets having greater than 1000 data points. All algorithms were trained using five different values for d, and error rates for each algorithm were selected as the minimum given by the five. For each dataset, relative performance ratios for each algorithm were computed by dividing the error rate of each algorithm by the minimum error rate of the three. The empirical cumulative distribution functions of relative performance ratios for each algorithm were computed and plotted in panels A-D of Figure 4. Such plots are called performance profiles (5). Performance profiles are useful in visualizing how frequently a particular algorithm wins, and when it loses, how frequently it loses by a certain amount. Areas under the curves (AUCs) were computed for each performance profile. This metric is valuable in that it doesn't necessarily indicate how frequently a particular algorithm is the best. Rather, it is an indicator of robustness. For instance, it is possible that an algorithm that wins on the majority of datasets can have a lower AUC than the other algorithms if, when it loses, it frequently loses by a large margin. On the original benchmarks (Figure 4 panel A), RerF achieves the largest AUC, followed by RF and lastly RotRF. The same trend is seen when scaling (panel C) and affine transformations (panel D) are applied. When the datasets are rotated, RotRF achieves the largest AUC, followed by RerF, and lastly RF.

Since the simulations in section IV.A revealed that the speed of RF and RerF scaled comparably, we wanted to see if this observation held on the benchmark data. In the same fasion as the left panels of Figure 4, we plotted performance profiles for training times in the right panels. Across all transformations, we see that RF and RerF perform comparably in terms of speed. Panels A and C show that RotRF performs the worst

on the original benchmark data and the randomly scaled benchmark data. It performs well, however, on the rotated and affine transformed data. One possible explanation for this is that when the data is rotated, RF and RerF have a more difficult time finding good split directions, which could result in deeper trees (and subsequently more time spent in training).

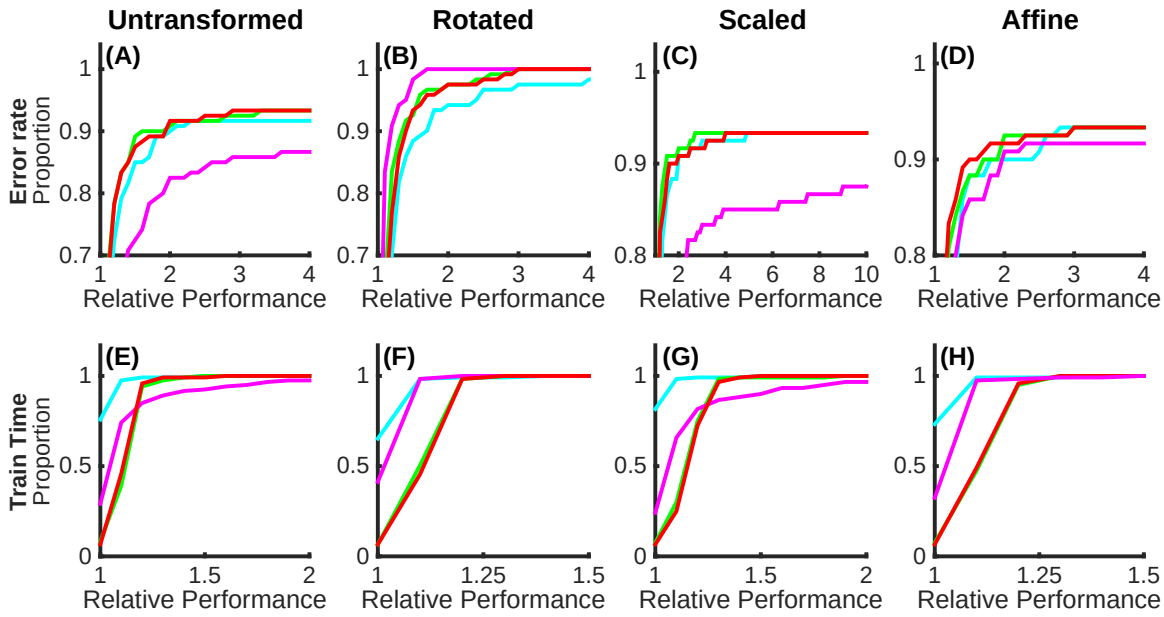

Figure 4: Classification (panels A-D) and speed (panels E-H) performance profiles on benchmarks with various transformations applied (see section IV.D for details). AUC denotes the area under the curve. In terms of classification accuracy, RerF outperforms RF in all five data settings and only loses to RotRF when the data is rotated. RF and RerF have comparable training time performance, while RotRF tends to be slower.

IV.E Rank Transforming the Data and Fast Supervised Projections

When dimensions of X are linearly combined, we lose *scale and unit invariance*. We therefore note that random forests have a special property: they are invariant to monotonic transformations of the data applied to each coordinate in the ambient (observed) space. They are effectively operating on the order statistics, rather than actual magnitudes of coordinates. In other words, if we convert for each dimension the values of the samples to their corresponding ranks, random forests yield the exact same result. Therefore, for our second trick, we adopt the same policy, and "**pass to ranks**", or rank transform, prior to doing anything else. We call RerFs that pass to ranks RerF(r).

Additionally, there is evidence that *using the data to construct A at each node* can significantly improve performance (7). However, previously proposed approaches for using the data require methods that are time- and space-intensive compared to standard random forests. We propose a simple strategy: "**compute the mean difference vectors**". In other words, given a two-class problem, let $\hat{\delta} = \hat{\mu}_0 - \hat{\mu}_1$, where $\hat{\mu}_c$ is the estimated class conditional mean. Under a spherically symmetric class conditional distribution assumption, $\hat{\delta}$ is the optimal projection vector. When there are $C > 2$ classes, the set of all pairwise distances between $\hat{\mu}_c$ and $\hat{\mu}_{c'}$ is of rank $C-1$. Because RerFs are approximately rotationally invariant, we can simply compute all the class conditional means, and subtract each from one of them (we chose the $\hat{\mu}_c$ for which n_c, the number of samples in that class, is the largest). Thus, we construct $\widetilde{X}$ by concatenating A^{T} with $\Delta = (\delta_{(2)} - \delta_{(1)}, \ldots, \delta_{(C)} - \delta_{(1)})$, where $\delta_{(j)}$ is the δ vector for the class with the j^{th} largest number of samples, to

obtain $\widetilde{A}^{\mathsf{T}}$. Then we compute $\widetilde{X} = \widetilde{A}^{\mathsf{T}}\bar{X}$. Computing this matrix is extremely fast, because it does not rely on costly singular value decompositions, matrix inversions, or convex problems. Thus, it nicely balances using the data to find good vectors, but not using much computational space or time. Furthermore, it also provides a set of dense projections to supplement the sparse projections of RerF. Denote RerFs that include this matrix RerF(d), and if they pass to ranks first, RerF(d+r).

Figure 6 demonstrates the utility of passing to ranks and using supervised projections on various transformations of the Trunk simulation. Panels A and B again show that RF and RerF are sensitive to rotation and therefore affine transformations. Panel C shows that RerF(d) achieves a lower error rate than RerF in the untransformed setting and is not sensitive to rotation. However, it is sensitive to scale and therefore affine transformations. Panel D shows that RerF(d+r) is more robust to affine transformations than the other methods, as hoped. Panel E again showsthat RotRF is sensitive to scale and subsequently affine transformations.

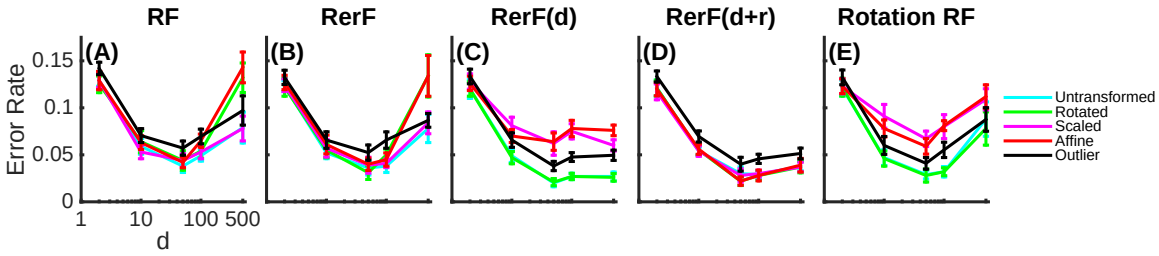

Figure 5: The utility of supplementing RerF with supervised projections and rank transforming the data prior to inducing trees (see section IV.E for details). The effect of various transformations on classification performance of RF, RerF, RerF(d), RerF(d+r), and RotRF for the Trunk simulation are shown in panels A-E, respectively. Panel D shows that RerF(d+r), which supplements RerF with supervised projections, and passes the data to ranks, is more robust to affine transformations than the other methods.

V Conclusion and Future Work

We have proposed novel methods for constructing decision forests, which we call RerFs. We view these methods as special cases of a more general random projection forest, which include Breiman's original Forest-IC and Forest-RC, as well as previously proposed oblique decision forests. We have demonstrated in simulations that RerFs are especially well-suited for classification problems in which axis-parallel splits are suboptimal, and at the same time, have a large number of irrelevant features relative to relevant ones. This could explain RerF's excellent empirical performance on a suite of over 100 benchmark datasets, as real data often has the properties just described. Moreover, RerF, unlike other oblique decision forests, preserves the time and space complexity of Forest-IC, and is only marginally sensitive to scaling. Lastly, we propose two additional modifications. One involves supplementing sparse random projections with a particular form of fast supervised projections in the construction of candidate split directions. The other involves passing the data to ranks with the goal of making the classifier more robust to outliers.

Much work is still to be done with our proposed methods. The simplicity of RerFs suggest that they will be amenable to theoretical investigation, extending the work of Biau et al. (2008) to the RerF setting (1). Moreover, we hope that theoretical investigations will yield more insight into which distributions $f_A(\mathcal{D}_n)$ will be optimal under different distributional settings, both asymptotically and under finite sample assumptions. Even under the currently proposed $f_A(\mathcal{D}_n)$, our implementation has room for improvement. Although it has the same space and time complexity as RF, we will determine explicit constants, and improve our implementation accordingly. Indeed, our current implementation is a proof-of-concept MATLAB implementation. We will utilize recent GPU and semi-external memory methods to significantly speed up RerF (14). As with all decision forests, multiclass classification is but one exploitation task they can be used for; therefore, we will also extend this work to enable regression, density estimation, clustering, and hashing. We will provide

open source code to enable more rigorous and extended analyses by the machine learning community.

Acknowledgements

This work is graciously supported by the Defense Advanced Research Projects Agency (DARPA) SIMPLEX program through SPAWAR contract N66001-15-C-4041 and DARPA GRAPHS N66001-14-1-4028.

VI Bibliography

[1] Biau, G., Devroye, L., and Lugosi, G. Consistency of random forests and other averaging classifiers. *The Journal of Machine Learning Research*, 9:2015–2033, 2008. 9

[2] Blaser, R. and Fryzlewicz, P. Random rotation ensembles. 2015. URL http://stats.lse.ac.uk/fryzlewicz/rre/rre.pdf. 1, 2, 3

[3] Breiman, L. Random forests. *Machine Learning*, 4(1):5–32, October 2001. 1

[4] Caruana, R., Karampatziakis, N., and Yessenalina, A. An empirical evaluation of supervised learning in high dimensions. *Proceedings of the 25th International Conference on Machine Learning*, 2008. 1

[5] Dolan, E. D. and Moré, J. J. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2008. 7

[6] Fernandez-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, October 2014. 1, 7

[7] Heath, D., Kasif, S., and Salzberg, S. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993. 1, 8

[8] Ho, T. K. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, Aug 1998. ISSN 0162-8828. doi: 10.1109/34.709601. 1

[9] Li, P., Hastie, T. J., and Church, K. W. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 287–296. ACM, 2006. 3

[10] Menze, B. H., Kelm, B.M, Splitthoff, D. N., Koethe, U., and Hamprecht, F. A. On oblique random forests. In Gunopulos, Dimitrios, Hofmann, Thomas, Malerba, Donato, and Vazirgiannis, Michalis (eds.), *Machine Learning and Knowledge Discovery in Databases*, volume 6912 of *Lecture Notes in Computer Science*, pp. 453–469. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23782-9. doi: 10.1007/978-3-642-23783-6_29. URL http://dx.doi.org/10.1007/978-3-642-23783-6_29. 1

[11] Rodriguez, J. J., Kuncheva, L. I., and Alonso, C. J. Rotation forest: A new classifier ensemble method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1619–1630, 2006. 1

[12] Tan, P. J. and Dowe, D. L. Mml inference of oblique decision trees. In *AI 2004: Advances in Artificial Intelligence*, pp. 1082–1088. Springer, 2005. 1

[13] Trunk, G. V. A problem of dimensionality: A simple example. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (3):306–307, 1979. 3

[14] Zheng, D., Mhembere, D., Burns, R., Vogelstein, J. T., Priebe, C. E., and Szalay, A. S. Flashgraph: Processing billion-node graphs on an array of commodity ssds. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, 2015. 9

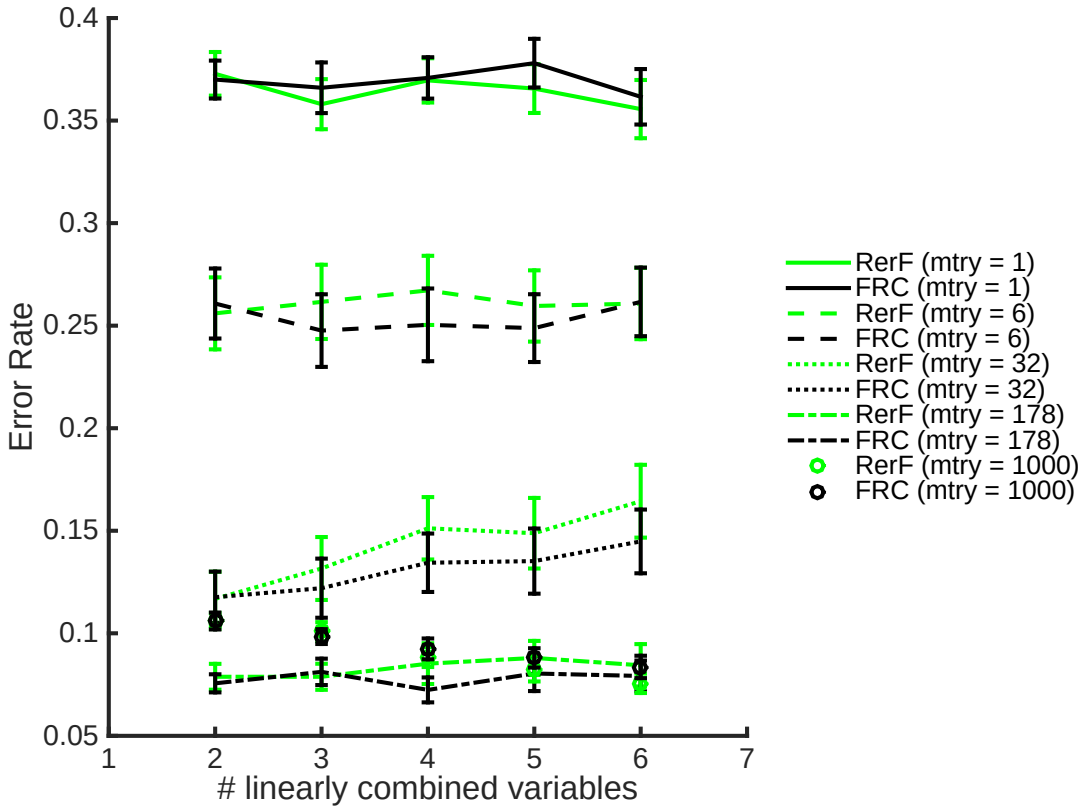

Figure 6: Error rate as a function of density of random projections and number of random projections sampled