Journal of Machine Learning Research 21 (2020) 1-39 Submitted 9/18; Revised 10/19; Published 5/20

Sparse Projection Oblique Randomer Forests

Tyler M. Tomita TTOMITA@JHU.EDU
Department of Psychological and Brain Sciences
Johns Hopkins University
Baltimore, MD 21218, USA

James Browne JAMES.BROWNE@JHU.EDU
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218, USA

Cencheng Shen SHENC@UDEL.EDU
Department of Applied Economics and Statistics
University of Delaware
Newark, DE 19716, USA

Jaewon Chung J1C@JHU.EDU
Jesse L. Patsolic JPATSOLIC@JHU.EDU
Benjamin Falk FALK.BEN@JHU.EDU
Carey E. Priebe CEP@JHU.EDU
Center for Imaging Science
Johns Hopkins University
Baltimore, MD 21218, USA

Jason Yim JASONKYUYIM@GOOGLE.COM
DeepMind
6 Pancras Square
London, UK N1C 4AG

Randal Burns RANDAL@CS.JHU.EDU
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218, USA

Mauro Maggioni MAURO@MATH.JHU.EDU
Department of Mathematics
Johns Hopkins University
Baltimore, MD 21218, USA

Joshua T. Vogelstein JOVO@JHU.EDU
Institute for Computational Medicine
Kavli Neuroscience Discovery Institute
Department of Biomedical Engineering
Johns Hopkins University
Baltimore, MD 21218, USA

Editor: Boaz Nadler

Abstract

Decision forests, including Random Forests and Gradient Boosting Trees, have recently demonstrated state-of-the-art performance in a variety of machine learning settings. Decision forests are typically ensembles of *axis-aligned* decision trees; that is, trees that split only along feature dimensions. In contrast, many recent extensions to decision forests are based on axis-oblique splits. Unfortunately, these extensions forfeit one or more of the favorable properties of decision forests based on axis-aligned splits, such as robustness to many noise dimensions, interpretability, or computational efficiency. We introduce yet another decision forest, called "Sparse Projection Oblique Randomer Forests" (`SPORF`). `SPORF` uses very sparse random projections, i.e., linear combinations of a small subset of features. `SPORF` significantly improves accuracy over existing state-of-the-art algorithms on a standard benchmark suite for classification with > 100 problems of varying dimension, sample size, and number of classes. To illustrate how `SPORF` addresses the limitations of both axis-aligned and existing oblique decision forest methods, we conduct extensive simulated experiments. `SPORF` typically yields improved performance over existing decision forests, while mitigating computational efficiency and scalability and maintaining interpretability. Very sparse random projections can be incorporated into gradient boosted trees to obtain potentially similar gains.

Keywords: Ensemble Learning, Random Forests, Decision Trees, Random Projections, Classification, Regression, Feature Extraction, Sparse Learning

1. Introduction

Over the last two decades, ensemble methods have risen to prominence as the state-of-the-art for general-purpose machine learning tasks. One of the most popular and consistently strong ensemble methods is Random Forests (`RF`), which uses decision trees as the base learners (Fernández-Delgado et al., 2014; Caruana et al., 2008; Caruana and Niculescu-Mizil, 2006). More recently, another tree ensemble method known as gradient boosted decision trees (GBTs) has seen a spike in popularity, largely due to the release of a fast and scalable cross-platform implementation, `XGBoost` (Chen and Guestrin, 2016). GBTs have been a key component of many Kaggle competition-winning solutions, and was part of the Netflix Prize winning solution (Chen and Guestrin, 2016).

`RF` and `XGBoost` are ensembles of "axis-aligned" decision trees. With such decision trees, the feature space is recursively split along directions parallel to the coordinate axes. Thus, when classes seem inseparable along any single dimension, axis-aligned splits require very deep trees with complicated step-like decision boundaries, leading to increased variance and over-fitting. To address this, Breiman also proposed and characterized Forest-RC (`F-RC`), which splits on linear combinations of coordinates rather than individual coordinates (Breiman, 2001). These so-called "oblique" ensembles include the axis-aligned ensembles as a special case, and therefore have an increased expressive capacity, conferring potentially better learning properties. Perhaps because of this appeal, numerous other oblique decision forest methods have been proposed, including the Random Rotation Random Forest (`RR-RF`) (Blaser and Fryzlewicz, 2016), and the Canonical Correlation Forest (`CCF`) (Rainforth and Wood, 2015). Unfortunately, these methods forfeit many of the desirable properties that axis-aligned trees possesses, such as computational efficiency, ease of tuning, insensitivity to a large proportion of irrelevant (noise) inputs, and interpretability. Furthermore, while these methods perform much better than axis-aligned ensembles on

some problems, they perform much worse than axis-aligned ensembles on some problems for which axis-aligned splits would in fact be highly informative. Therefore, there is a need for a method that combines the expressive capacity of oblique ensembles with the benefits of axis-aligned ensembles.

We propose Sparse Projection Oblique Randomer Forests (`SPORF`) for learning an ensemble of oblique, interpretable, and computationally efficient decision trees. At each node of each tree, `SPORF` searches for splits over a sample of very sparse random projections (Li et al., 2006), rather than axis-aligned splits. Very sparse random projections preserve many of the desirable properties of axis-aligned decision trees, while mitigating their issues.

In section 3.1, we delineate a set of desirable properties of a decision forest algorithm, and describe how current axis-aligned and oblique decision forest algorithms each fail to possess at least one of these properties. This motivates a flavor of sparse random projections for randomly sampling candidate split directions. In Section 4, we show on simulated data settings how our method possesses all of these desirable properties, while other methods do not. In Section 5 we find that, in practice, our method tends to be more accurate than RF and existing methods on many real data sets. Last, in Section 6 we demonstrate how are method is computationally expedient and scalable.

Our statistically- and computationally-efficient parallelized implementations are available from `https://neurodata.io/sporf/` in both R and Python. Our R package is available on the Comprehensive R Archive Network (CRAN) (`https://cran.r-project.org/web/packages/rerf/`), and our Python package is available from PyPi (`https://pypi.org/project/rerf/2.0.5/`), and is sklearn API compliant.

2. Background & Related Work

First we review the original Random Forest algorithm. Next we review extensions of it that have been proposed. Then we briefly review random projections, which we use in our method and which have been used in other extensions of Random Forests. Last, we review gradient boosted trees, which we empirically compare to our method.

2.1. Random Forests

The original `RF` procedure popularized by Leo Breiman is one of the most commonly employed classification learning algorithms (Breiman, 2001). We note that `RF` can be used for various other supervised and unsupervised machine learning tasks, but do not consider those tasks here. Let $X \in \mathbb{R}^p$ be a random real-valued feature vector and $Y \in \mathcal{Y} = \{c_1, ..., c_K\}$ be a random variable denoting a class label associated with X. `RF` proceeds by building T decision trees via a series of recursive binary splits of the training data. The nodes in a tree are split into two daughter nodes by maximizing some notion of information gain, which typically reflects the reduction in class impurity of the resulting daughter nodes. A common measure of information gain in decision trees is the decrease in Gini impurity, $I(S)$, for a set of observations S. The Gini impurity for classification is defined as $I(\mathcal{S}) = \sum_{k=1}^{K} f_k(1 - f_k)$, where $f_k = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \mathbb{I}[y_i = c_k]$. More concretely, let $\theta = (j, \tau)$, where j is an index selecting a dimension and τ is a splitting threshold. Furthermore, let $\mathcal{S}_\theta^L = \{i : x_i^{(j)} \leq \tau, \forall i \in \mathcal{S}\}$ and $\mathcal{S}_\theta^R = \{i : x_i^{(j)} > \tau, \forall i \in \mathcal{S}\}$ be the subsets of $\mathcal{S}$ to the left and right of the splitting thresh-

old, respectively. Here, $x_i^{(j)}$ denotes the value of the jth feature for the ith observation. Let n_S, n_L, and n_R denote the number of points in the parent, left, and right child nodes, repsectively. A split is made on a "best" $\theta^* = (j^*, \tau^*)$ via the following optimization:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, n_S I(\mathcal{S}) - n_L I(\mathcal{S}_\theta^L) - n_R I(\mathcal{S}_\theta^R).$$

This optimization is performed by exhaustive search for the best split threshold τ^* over a random subset of the features. Specifically, a random subset of the p features is sampled. For each feature in this subset, the observations are sorted from least to greatest, and the split objective function is evaluated at each midway point between adjacent pairs of observations.

Nodes are recursively split until a stopping criteria is reached. Commonly, the recursion stops when either a maximum tree depth is reached, a minimum number of observations in a node is reached, or a node is completely pure with respect to class label. The result of the tree induction algorithm is a set of split nodes and leaf nodes. The leaf nodes are disjoint hyperrectangular partitions of the feature space $\mathcal{X}$, and each one is associated with a local prediction function. Let l_m be the m^{th} leaf node of an arbitrary classification tree, and let $\mathcal{S}(l_m) = \{i : x_i \in l_m \forall i \in [n]\}$ be the subset of the training data contained in l_m. The local leaf prediction is

$$h(l_m) = \underset{c_k \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i \in \mathcal{S}(l_m)} \mathbb{I}[y_i = c_k]$$

A tree makes a prediction for a new observation x by passing the observation down the tree according to the split functions associated with each split node until a terminal leaf node is reached. Letting $m(x)$ be the index of the leaf node that x falls into, the tree prediction is $h(l_{m(x)})$. Let $\widehat{y}^{(t)}$ be the prediction made by the t^{th} tree. Then the prediction of the `RF` is the plurality vote of the predictions made by each tree:

$$\widehat{y} = \underset{c_k \in \mathcal{Y}}{\operatorname{argmax}} \sum_{t=1}^{T} \mathbb{I}[\widehat{y}^{(t)} = c_k]$$

Breiman (2001) proved that the misclassification rate of a tree ensemble is bounded above by a function inversely proportional to the strength and diversity of its trees. `RF` decorrelates (diversifies) the trees via two mechanisms: (1) constructing each tree on a random bootstrap sample of the original training data, and (2) restricting the optimization of the splitting dimension j over a random subset of the total p dimensions. The combination of these two randomizing effects typically leads to generalization performance that is much better than that of any individual tree (Breiman, 2001).

2.2. Oblique Extensions to Random Forest

Various tactics have been employed to further promote the strength and diversity of trees. One feature of `RF` that limits both strength and diversity is that splits must be along the coordinate axes of the feature space. Therefore, one main focus for improving `RF` is to somehow relax this restriction. The resulting forests are sometimes referred to as "oblique"

decision forests, since the splits can be along directions oblique to the coordinate axes. This type of tree was originally developed for computer graphics applications, and is also known as binary space partitioning (BSP) trees. Statistical consistency of BSP trees has been proven for a simplified data-agnostic BSP tree procedure (Devroye et al., 1996). Various approaches have been proposed for constructing oblique forests. Breiman (2001) proposed the Forest-RC (`F-RC`) algorithm, which constructs d univariate projections, each projection a linear combination of L randomly chosen dimensions. The weights of each projection are independently sampled uniformly over the interval $[-1, 1]$. Strangely, Breiman's `F-RC` never garnered the popularity that `RF` has acquired; both Breiman (2001) and Tomita et al. (2017) found that `F-RC` tends to empirically outperform `RF` on a wide variety of data sets. Heath et al. (1993) sample a randomly oriented hyperplane at each split node, then iteratively perturb the orientation of the hyperplane to achieve a good split. Rodriguez et al. (2006) attempted to find discriminative split directions via PCA. Menze et al. (2011) perform supervised learning of linear discriminative models at each node. Blaser and Fryzlewicz (2016) proposed the random rotation Random Forest (`RR-RF`) method, which uniformly randomly rotates the data prior to inducing each tree. Trees are then learned via the typical axis-aligned procedure on the rotated data. Rainforth and Wood (2015)'s Canonical Correlation Forests (`CCF`) employ canonical correlation analysis at each split node in order to directly compute split directions that maximally correlate with the class labels. Lee et al. (2015)'s Random Projection Forests (RPFs) generates a discriminative image filter bank for head-pose estimation at each split node and compresses the responses using random projections. The key thing to note is that all of these aforementioned oblique methods use some flavor of random projections, which we briefly introduce next.

2.3. Random Projections

Given a data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, one can construct a random projection matrix $\mathbf{A} \in \mathbb{R}^{p \times d}$ and multiply it by $\mathbf{X}$ to obtain

$$\widetilde{\mathbf{X}} = \mathbf{X}\mathbf{A} \in \mathbb{R}^{n \times d}, \quad d \ll \min(n, p).$$

If the random matrix entries a_{ij} are i.i.d. with zero mean and constant variance, then the much smaller matrix $\widetilde{\mathbf{X}}$ preserves all pairwise distances of $\mathbf{X}$ with small distortion and high probability[1].

Due to theoretical guarantees, random projections are commonly employed as a dimensionality reduction tool in machine learning applications (Bingham and Mannila, 2001; Fern and Brodley, 2003; Fradkin and Madigan, 2003; Achlioptas, 2003; Hegde et al., 2008). Different probability distributions over the entries lead to different average errors and error tail bounds. Li et al. (2006) demonstrates that **very sparse random projections**, in which a large fraction of entries in $\mathbf{A}$ are zero, can maintain high accuracy and significantly speed up the matrix multiplication by a factor of $\sqrt{p}$ or more. Specifically, a very sparse random projection matrix is constructed by sampling entries a_{ij} with the following probability distribution:

[1] In classification, preservation of pairwise interpoint distances is not necessarily important. Rather, useful projections or manifolds in classification are those that minimize within-class distances while maximizing between-class distances. We introduce the topic here because of its relevance and use in many decision tree algorithms, as is discussed in Section 3.1.

$$a_{ij} = \begin{cases} +1 & \text{with prob. } \frac{1}{2s} \\ 0 & \text{with prob. } 1 - \frac{1}{s}, \\ -1 & \text{with prob. } \frac{1}{2s} \end{cases} \qquad \text{typically } s \gg 3$$

Dasgupta and Freund (2008) proposed Random Projection Trees, in which they sampled dense random projections in an unsupervised fashion to approximate low dimensional manifolds, and later for vector quantization (Dasgupta and Freund, 2009) and nearest neighbor search (Dasgupta and Sinha, 2013). Our work is inspired by this work, but in a supervised setting.

2.4. Gradient Boosted Trees

Gradient boosted trees (GBTs) are another tree ensemble method commonly used for regression and classification tasks. Unlike `RF`, GBTs are learned in an iterative stage-wise manner by directly minimizing a cost function via gradient descent (Breiman, 1998; Friedman, 2001). Despite the obvious differences in the learning procedures between GBT and `RF`, they tend to perform comparably. A study by Wyner et al. (2017) argues that `RF` and GBT are both successful for the same reason—namely both are weighted ensembles of interpolating classifiers that learn local decision rules.

GBTs have recently seen a marked surge in popularity, and were used as components in many recent Kaggle competitions. This is in part due to their tendency to be accurate over a wide range of settings. Their popularity and success can also be attributed to the recent release of `XGBoost` (Chen and Guestrin, 2016) and `LightGBM` (Ke et al., 2017), both extremely fast and scalable open-source software implementations. Due to the success of GBTs in many data science applications, we compare the `XGBoost` implementation to our methods.

3. Methods

Here we introduce Sparse Projection Oblique Randomer Forests, and discuss how it addresses limitations of existing decision tree ensemble methods. We then describe general details regarding empirical evaluation, including the other methods we compare our method to, as well as how these methods are trained and tuned.

3.1. Sparse Projection Oblique Randomer Forests

Extensions of `RF` are often focused on changing the procedure for finding suitable splits, such as employing a supervised linear procedure or searching over a set of randomly oriented hyperplanes. Such extensions, along with `RF`, simply differ from each other by defining different random projection distributions from which candidate split directions are sampled. Thus, they are different special cases of a general random projection forest.

Specifically, let $\mathbf{X} \in \mathbb{R}^{n \times p}$ be the observed feature matrix of n samples at a split node, each p-dimensional. Randomly sample a matrix $\mathbf{A} \in \mathbb{R}^{p \times d}$ from distribution $f_{\mathbf{A}}$, possibly in a data-dependent or supervised fashion. This matrix is used to randomly project the feature matrix, yielding $\widetilde{\mathbf{X}} = \mathbf{X}\mathbf{A} \in \mathbb{R}^{n \times d}$, where d is the dimensionality of the projected

space. The search for the best split is then performed over the dimensions in the projected space. As an example, in RF, $\mathbf{A}$ is a random matrix in which each of the d columns has only one nonzero entry, and every column is required to be unique. Searching for the best split over each dimension in this projected subspace amounts to searching over a random subset of the original features.

While the best specification of a probability distribution over random projections (if one exists) is data set-dependent, it is unreasonable and/or undesirable to try more than a handful of different cases. Therefore, for general purpose classification we advocate for a default projection distribution based on the following desiderata:

- **Random Search for Splits.** The use of guided (supervised) linear search procedures for computing split directions, such as linear discriminant analysis (LDA), canonical correlation analysis (CCA), or logistic regression (LR), can result in failure to learn good split directions on certain classification problems (for example, the XOR problem). On the other hand, searching over a random set of split directions *can* identify good splits in many of such cases. Furthermore, supervised procedures run the risk of being overly greedy and reducing tree diversity, causing the model to overfit noise (we demonstrate this in high-dimensional settings in Section 5.3). This is why in RF it is typically better to evaluate a random subset of features to split on, rather than exhaustively evaluate all features. Lastly, supervised procedures become costly if performed at every split node.

- **Flexible Sparsity.** RFs search for splits over fully sparse, or axis-aligned, projections. Thus, it may perform poorly when no single feature is informative. On the other hand, methods that search for splits within a fully dense randomly projected space, such as RR-RFs, perform poorly in high-dimensional settings for which the signal is contained in a small subset of the features. This is because the large space renders the probability of sampling discriminative random projections very small (we refer the reader to pages 49-50 of Vershynin, 2019, for relevant theory of random projections). However, inducing an appropriate amount of sparsity in the random projections increases the probability of sampling discriminative projections in such cases. Tomita et al. (2017) demonstrated that F-RC, which allows control over the sparsity of random projections, empirically performed much better than both RF and RR-RF

- **Ease of Tuning.** RFs tend to work fairly well out-of-the-box, due to their relative insensitivity to hyperparameter settings (Probst et al., 2019). Unfortunately, existing oblique forests introduce additional hyperparameters to which they are sensitive to.

- **Data Insight.** Often times the goal is not simply to produce accurate predictions, but to gain insight into a process or phenomenon being studied. While RF models can have complicated decision rules, Gini importance (Breiman and Cutler, 2002) has been proposed as a computationally efficient way to assess the relative contribution (importance) of each feature to the learned model. As is explained in Section 4.5, existing oblique forests do not lend themselves well to computation of Gini importance.

- **Expediency and Scalability.** Existing oblique forest algorithms typically involve expensive computations to identify and select splits, rendering them less space and time efficient than RF, and/or lack parallelized implementations.

With these considerations in mind, we propose a new decision tree ensemble method called Sparse Projection Oblique Randomer Forests (SPORF). The name of our method stems from the fact that it searches for splits over sparse random projections (Li et al., 2006), which in some sense can be viewed as being more random than RF's feature subsampling procedure. Specifically, rather than sampling d non-zero elements of $\mathbf{A}$ and enforcing that each column gets a single non-zero number (without replacement), as RF does, we relax these constraints and sample $\lceil \lambda pd \rceil$ non-zero numbers from $\{-1, +1\}$ with equal probabilities, where $\lambda \in (0, 1]$ is the density (fraction of nonzeros) of $\mathbf{A}$ and $\lceil \cdot \rceil$ is the ceiling function rounding up to the nearest integer.[2] These nonzeros are then distributed uniformly at random in $\mathbf{A}$. See Algorithms 1 and 2 for details on how to grow a SPORF decision tree.

SPORF addresses all of the desiderata listed above. The use of sparse random projections with control over the sparsity via λ addresses the first two. Additionally, λ is the only new hyperparameter to tune relative to RF. Breiman's F-RC has an analogous hyperparameter L, which fixes the number of variables in every linear combination. However, we show later that SPORF is less sensitive to the choice in λ than F-RC is to the choice in L. By keeping the random projections sparse with only two discrete weightings of ± 1, Gini importance of projections can be computed in a straightforward fashion. Last, sparse random projections are cheap to compute, which allows us to maintain computational expediency and scalability similar to that of RF.

3.2. Training and Hyperparameter Tuning

Unless stated otherwise, model training and tuning for all algorithms except for XGBoost and CCF is performed in the following way. Each algorithm trains 500 trees, which was empirically determined to be sufficient for convergence of out-of-bag error for all methods. The split objective is to maximize the reduction in Gini impurity. In all methods, classification trees are fully grown unpruned (i.e. nodes are split until pure). While fully grown trees often cause a single tree to overfit, averaging over many uncorrelated trees tends to alleviate overfitting. A recent study suggests that RF is fairly insensitive to its hyperparameters relative to other machine learning algorithms. Furthermore, the study finds that RF is much less sensitive to tree depth than the number of candidate split directions sampled at each split node (Probst et al., 2019).

Two hyperparameters are tuned via minimization of out-of-bag error. The first hyperparameter tuned is d, the number of candidate split directions evaluated at each split node. Each algorithm is trained for $d = p^{1/4}$, $p^{1/2}$, $p^{3/4}$, and p. Additionally, SPORF and F-RC are trained for $d = p^2$. For RF, d is restricted to be no greater than p by definition. The second hyperparameter tuned is λ, the average sparsity of univariate projections sampled at each split node. The values optimized over for SPORF and F-RC are $\{1/p, \ldots, 5/p\}$. Note, for RF λ is fixed to $1/p$ by definition, since the univariate projections are constrained to be along one of the coordinate axes of the data.

For CCF, the number of trees is 500, trees are fully grown, and the split objective is to maximize the reduction in class entropy (this is the default objective found to perform best by the authors). The only hyperparameter tuned is the number of features subsampled prior to performing CCA. We optimize this hyperparameter over the set $\{p^{1/4}, p^{1/2}, p^{3/4}, p\}$.

[2] While λ can range from zero to one, we only try values from $1/p$ up to $5/p$ in our experiments.

CCF uses a different observation subsampling procedure called *projection boostrapping* instead of the standard bootstrap procedure. Briefly, in projection bootstrapping, all trees are trained on the full set of training observations. Bootstrapping is instead performed at the node level when computing the canonical correlation projections at each node. Once the projections are computed, the projection and corresponding split threshold that maximizes the reduction in Gini impurity is found using all of the node observations (not just the bootstrapped node observations). Since there are no out-of-bag samples for each tree, we base the selection of the best value on minimization of a five-fold cross-validation error rate instead.

Five hyperparameters of XGBoost are tuned via grid search using the R caret package (see Appendix B for details).

4. Simulated Data Empirical Performance

In this section we demonstrate, using synthetic classification problems, that SPORF addresses the statistical issues listed above. In a sense, SPORF bridges the gap between RF and existing oblique methods.

4.1. SPORF and Other Oblique Forests are "More Consistent" Than RF

Typically, a proposed oblique forest method is motivated through purely empirical examples. Moreover, the geometric intuition behind the proposed method is rarely clearly provided. Here we take a step towards a more theoretical perspective on the advantage of oblique splits in tree ensembles.

In classification, a learning procedure is consistent if the resultant classifier converges to the Bayes optimal classifier as the number of training samples tends to infinity. Although we do not yet have a proof of the consistency of SPORF or other oblique forests, we do propose that they are "more" consistent than Breiman's original RF. Biau et al. (2008) proposed a binary classification problem for which Breiman's RF is inconsistent. The joint distribution of (X, Y) is as follows: $X \in \mathbb{R}^2$ has a uniform distribution on $[0,1] \times [0,1] \cup [1,2] \times [1,2] \cup [2,3] \times [2,3]$. The class label Y is a deterministic function of X, that is $f(X) \in \{0,1\}$. The $[0,1] \times [0,1]$ square is divided into countably infinite vertical stripes, and $[2,3] \times [2,3]$ square is similarly divided into countably infinite horizontal stripes. In both squares, the stripes with $f(X) = 0$ and $f(X) = 1$ alternate. The $[1,2] \times [1,2]$ square is a 2×2 checker board. Figure 1A shows a schematic illustration (because we cannot show countably infinite rows or columns). On this problem, Biau et al. (2008) show that RF cannot achieve an error lower than 1/6. This is because RF will always choose to split either in the lower left square or top right square and never in the center square. On the other hand, Figure 1B shows that SPORF, RR-RF, and CCF approach perfect classification; although also greedy, they will choose with some probability oblique splits of the middle square to enable lower error. Therefore, SPORF and other oblique methods are empirically more consistent on at least some settings on which RF is neither empirically or theoretically consistent.

To our knowledge, this is the first result comparing the consistency of RF to an oblique forest method. More generally, this result suggests that relaxing the constraint of axis-alignment of splits may allow oblique forests to be consistent on a wider set of classification problems.

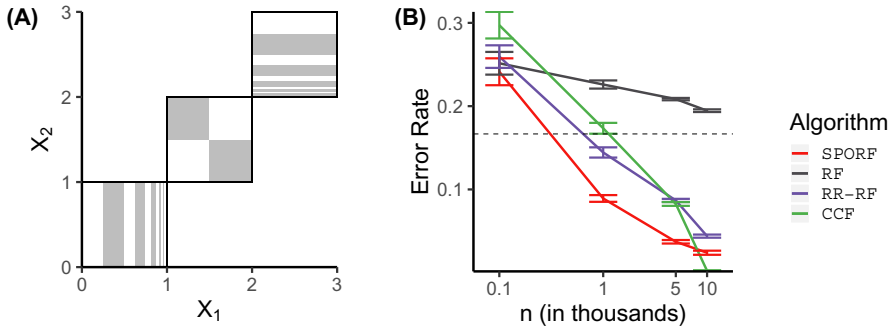

Figure 1: Classification performance on the consistency ($p = 2$) problem as a function of the number of training samples. The consistency problem is designed such that `RF` has a theoretical lower bound of error of 1/6. (A): The joint distribution of (X, Y). X is uniformly distributed in the three unit squares. The lower left and upper right squares have countably infinite stripes (a finite number of stripes are shown), and the center square is a 2×2 checkerboard. The white areas represent $f(X) = 0$ and gray areas represent $f(X) = 1$. (B): Error rate as a function of n. The dashed line represents the lower bound of error for `RF`, which is 1/6. `SPORF` and other oblique methods achieve an error rate dramatically lower than the lower bound for `RF`.

4.2. Simulated data sets

In the next few sections, we perform a variety of experiments on three carefully constructed simulated classification problems, refered to as **Sparse Parity**, **Orthant**, and **Trunk**. These constructions were chosen to highlight various properties of different algorithms and gain insight into their behavior.

Sparse Parity is a multivariate generalization of the noisy XOR problem. It is a p-dimensional two-class problem in which the class label Y is 0 if the number of dimensions having positive values amongst the first $p^* < p$ dimensions is even and $Y = 1$ otherwise. Thus, only the first p^* dimensions carry information about the class label, and no subset of dimensions contains any information. Specifically, let $X = (X_1, \ldots, X_p)$ be a p-dimensional feature vector, where each $X_1, \ldots, X_p \overset{iid}{\sim} U(-1, 1)$. Furthermore, let $Q = \sum_{j=1}^{p^*} \mathbb{I}(X_j > 0)$, where $p^* < p$ and $\mathbb{I}(\cdot)$ is the indicator function. A sample's class label Y is equal to the parity of Q. That is, $Y = \texttt{odd}(Q)$, where `odd` returns 1 if its argument is odd, and 0 otherwise. The Bayes optimal decision boundary for this problem is a union of hyperplanes aligned along the first p^* dimensions. For the experiments presented in the following sections, $p^* = 3$ and $p = 20$. Figure 2A,B show cross-sections of the first two dimensions taken at two different locations along the third dimension. This setting is designed to be relatively easy for `F-RC`, but relatively difficult for `RF`.

Orthant is a multi-class problem in which the class label is determined by the orthant in which a data point resides. An orthant in $\mathbb{R}^p$ is a generalization of a quadrant in $\mathbb{R}^2$. In other words, each orthant is a subset of $\mathbb{R}^p$ defined by constraining each of the p coordinates

to be positive or negative. For instance, in $\mathbb{R}^2$, there are four such subsets: $X = (X_1, X_2)$ can either be in 1) $\mathbb{R}^+ \times \mathbb{R}^+$, 2) $\mathbb{R}^- \times \mathbb{R}^+$, 3) $\mathbb{R}^- \times \mathbb{R}^-$, or 4) $\mathbb{R}^+ \times \mathbb{R}^-$. Note that the number of orthants in p dimensions is 2^p. A key characteristic of this problem is that the individual dimensions are strongly and equally informative. Specifically for our experiments, we sample each $X_1, \ldots, X_p \overset{iid}{\sim} U(-1, 1)$. Associate a unique integer index from 1 to 2^p with each orthant, and let $O(X)$ be the index of the orthant in which X resides. The class label is $Y = O(X)$. Thus, there are 2^p classes. The Bayes optimal decision boundary in this setting is a union of hyperplanes aligned along each of the p dimensions. We set $p = 6$ in the following experiments. Figure 2D,E show cross-sections of the first two dimensions taken at two different locations along the third dimension. This setting is designed to be relatively easy for `RF` because all optimal splits are axis-aligned.

Trunk is a balanced, two-class problem in which each class is distributed as a p-dimensional multivariate Gaussian with identity covariance matrices (Trunk, 1979). Every dimension is informative, but each subsequent dimension is less informative than the previous. The class 1 mean is $\mu_1 = (1, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{3}}, ..., \frac{1}{\sqrt{p}})$, and $\mu_0 = -\mu_1$. The Bayes optimal decision boundary is the hyperplane $(\mu_1 - \mu_0)^\mathsf{T} X = 0$. We set $p = 10$ in the following experiments.

4.3. `SPORF` Combines the Best of Existing Axis-Aligned and Axis-Oblique Methods

We compare error rates of `RF`, `SPORF`, `F-RC`, and `CCF` on the sparse parity and orthant problems. Training and tuning are performed as described in Section 3.2. Error rates are estimated by taking a random sample of size n, training the classifiers, and computing the fraction misclassified in a test set of 10,000 samples. This is repeated ten times for each value of n. The reported error rate is the mean over the ten repeated experiments.

`SPORF` performs as well as or better than the other algorithms on both the sparse parity (Figure 2C) and orthant problems (Figure 2F). `RF` performs relatively poorly on the sparse parity problem. Although the optimal decision boundary is a union of axis-aligned hyperplanes, each dimension is completely uninformative on its own. Since axis-aligned partitions are chosen one at a time in a greedy fashion, the trees in `RF` struggle to learn the correct partitioning. On the other hand, oblique splits are informative, which substantially improves the generalizability of `SPORF` and `F-RC`. While `F-RC` performs well on the sparse parity problem, it performs much worse than `RF` and `SPORF` on the orthant problem. On the orthant problem, in which `RF` is is designed to do exceptionally well, `SPORF` performs just as well. `CCF` performs poorly on both problems, which may be because CCA is not optimal for the particular data distributions. For instance, in the sparse parity problem, the projection found by CCA at the first node is approximately the difference in class-conditional means, which is zero. Furthermore, `CCF` only evaluates $d = min(l, C - 1)$ projections at each split node, where l is the number of dimensions subsampled and C is the number of classes. On the other hand, `SPORF` evaluates d random projections, and d could be as large as 3^p (each of the p elements can be either 0 or ± 1. Overall, `SPORF` is the only method of the four that performs relatively well on all of the simulated data settings.

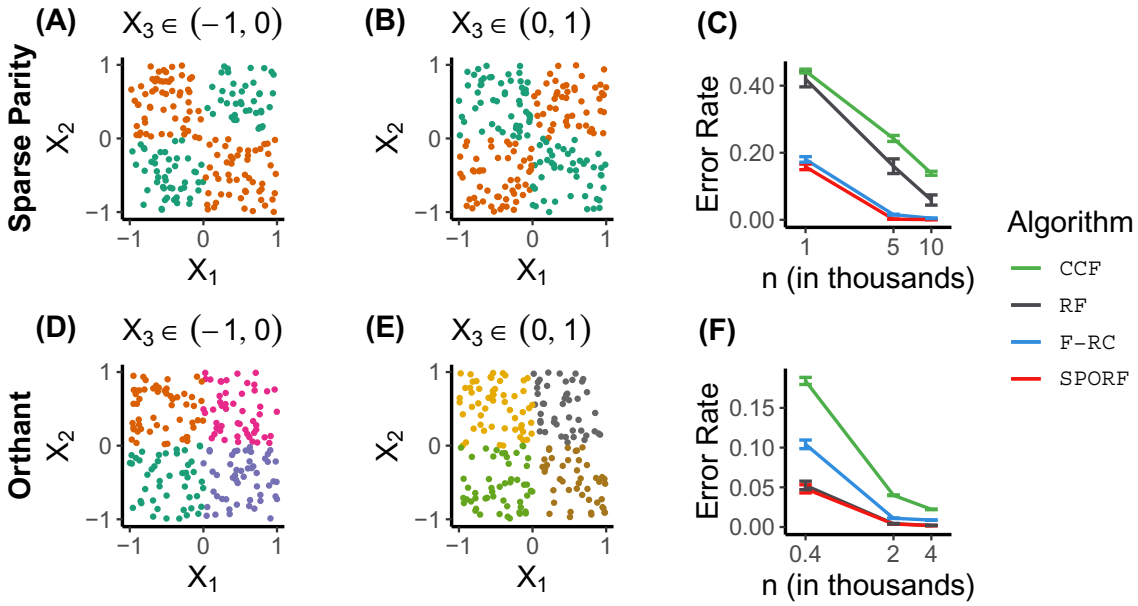

Figure 2: Classification performance on the sparse parity ($p_s = 20$) and orthant ($p_o = 6$) problems for various numbers of training samples. In both cases, we sample $X_1, \ldots, X_p \overset{iid}{\sim} U(-1, 1)$. `F-RC` has been known to perform much better than `RF` on the sparse parity problem (Tomita et al., 2017). The orthant problem is designed for `RF` to perform well because the optimal splits are axis-aligned. (A): A cross-section of the first two dimensions of sparse parity when $X_3 \in (-1, 0)$. Only the first three dimensions are informative w.r.t. class label. (B): The same as (A), except that the cross-section is taken over $X_3 \in (0, 1)$. (C): Error rate plotted against the number of training samples for sparse parity. Error rate is the average over ten repeated experiments. Error bars indicate the standard error of the mean. (D-F): Same as (A-C) except for the orthant problem. `SPORF` is the only method of the four that performs well across all simulated data settings.

4.4. `SPORF` is Robust to Hyperparameter Selection

One key difference between the random projection distribution of `SPORF` and `F-RC` is that `F-RC` requires that a hyperparameter be specified to fix the sparsity of the sampled univariate projections (individual linear combinations). Breiman denoted this hyperparameter L. `SPORF` on the other hand, requires that sparsity be specified on the entire random matrix $\mathbf{A}$, and hence, only an *average* sparsity on the univariate projections (details are in Section 3.1). In other words, `SPORF` induces a probability distribution with positive variance on the sparsity of univariate projections, whereas in `F-RC` that distribution is a point mass. If the Bayes optimal decision boundary is locally sparse, mis-specification of the hyperparameter controlling the sparsity of $\mathbf{A}$ may be more detrimental to `F-RC` than `SPORF`. Therefore, we examine the sensitivity of classification performance of `SPORF` and `F-RC` to the sparsity hyperparameter λ on the simulated data sets described previously. For `SPORF`, λ is defined as

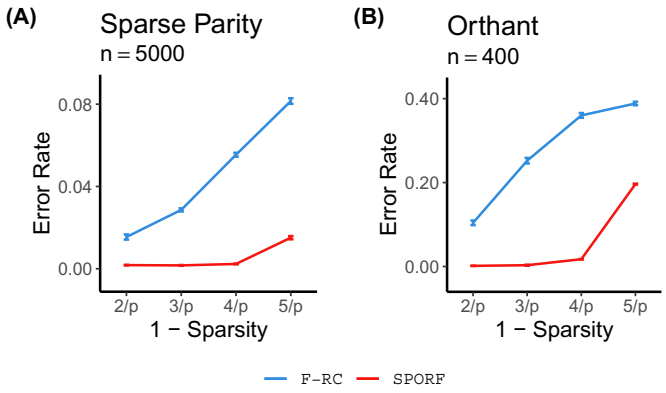

Figure 3: Dependence of error rate on the hyperparameter λ, which controls the average density (1 - sparsity) of projections for two different simulation settings. (A): Error rate as a function of λ on sparse parity ($n = 5000$, $p = 20$). (B): The same as (A) except on orthant ($n = 400$, $p = 6$). In both cases, `SPORF` is less sensitive to different values of λ than is `F-RC`.

in Section 3.1. For `F-RC`, we note that $\lambda = L/p$; in other words, the density of a univariate projection is the number of features to combine divided by the total number of features. For each of $\lambda \in \{\frac{2}{p}, \ldots, \frac{5}{p}\}$, the best performance for each algorithm is selected with respect to the hyperparameter based on minimum out-of-bag error. Error rate on the test set is computed for each of the four hyperparameter values for the two algorithms. Figure 3 shows the dependence of error rates of `SPORF` and `F-RC` on λ for the Sparse Parity ($n = 5{,}000$) and Orthant ($n = 400$) settings. The $n = 5{,}000$ setting for Sparse Parity was chosen because both `F-RC` and `SPORF` perform well above chance (see Figure 2C). The $n = 400$ setting for Orthant was chosen for the same reason and also because it displays the largest difference in classification performance in Figure 2F. In both settings, `SPORF` is more robust to the choice of sparsity level than `F-RC`.

4.5. `SPORF` Learns Important Features

For many data science applications, understanding which features are important is just as critical as finding an algorithm with excellent predictive performance. One of the reasons `RF` is so popular is that it can learn good predictive models that simultaneously lend themselves to extracting suitable feature importance measures. One such measure is the mean decrease in Gini impurity (hereafter called Gini importance) (Devroye et al., 1996). This measure of importance is popular because of its computational efficiency: it can be computed during training with minimal additional computation. For a particular feature, it is defined as the sum of the reduction in Gini impurity over all splits of all trees made on that feature. With this measure, features that tend to yield splits with relatively pure nodes will have large importance scores. When using `RF`, features with low marginal information about the class label, but high pairwise or other higher-order joint distributional information, will likely receive relatively low importance scores. Since splits in `SPORF` are linear

combinations of the original features, such features have a better chance of being identified. For SPORF, we compute Gini importance for each unique univariate projection (i.e. single linear combination). Of note, two projections that differ only by a sign project into the same subspace. However, in the experiment that follows we do not check whether any two projections used in the grown forest differ only by a sign.

Another measure of feature importance which we do not consider here is the permutation importance. Permutation importance of a particular feature is computed by shuffling the values along that feature and subsequently assessing how much the error rate increases using the shuffled feature to predict (relative to intact). This measure is considerably slower to compute than is Gini importance in high-dimensional settings because predictions are made for each permuted feature. Furthermore, it is unclear how to appropriately compute permutation importance for linear combinations of features.

SPORF is advantageous over methods such as F-RC ,RR-RF and CCF because those methods do not lend themselves to computation of Gini importance. The reason for this is that a particular univariate projection must be sampled and chosen many times over many trees in order to compute a stable estimate of its Gini importance. Since the aforementioned algorithms randomly sample continuous coefficients, it is extremely improbable that the same exact univariate projections will be sampled more than once across trees. On the other hand, the projections sampled in SPORF are sparse and only contain coefficients of ± 1, making it much more likely to sample any given univariate projection repeatedly. Furthermore, RR-RF and CCF split on dense univariate projections, which are less interpretable.

Gini importance was computed for each feature for both RF and SPORF on the Trunk problem with $n = 1,000$. Figure 4A,B depict the features that define each of the top ten split node projections for SPORF and RF, respectively. Projections are sorted from highest to lowest Gini importance. The top ten projections in SPORF are all linear combinations of dimensions, whereas in RF the projections can only be along single dimensions. The linear combinations in SPORF tend to include the first few dimensions, which contain most of the "true" signal. The best possible projection that SPORF could sample is the vector of all ones. However, since $\lambda = 1/2$ for this experiment, the probability of sampling such a dense projection is almost negligible. Figure 4C shows the normalized Gini importance of the top ten projections for each algorithm. The top ten most important features according to SPORF are all more important (in terms of Gini) than any of the RF features, except the very first one. Figure 4D shows the Bayes error rate of the top ten projections for each algorithm. Again, the top ten features according to SPORF are more informative than any of those according to RF. In other words, SPORF learns features that are more important than any of the observed features, and those features are interpretable, as they are sparse linear combinations of the observed features. The ability of SPORF to learn new identifiable features distinguishes it from RF, which cannot learn new features.

5. Real Data Empirical Performance

In this section we assess performance of SPORF on a large suite of data sets from the UCI machine learning repository. Based on a grid sweep over hyperparamter settings on these data sets, we identify a default hyperparameter setting which performs best on average. We then add various numbers of noise dimensions to these data sets and show that SPORF is

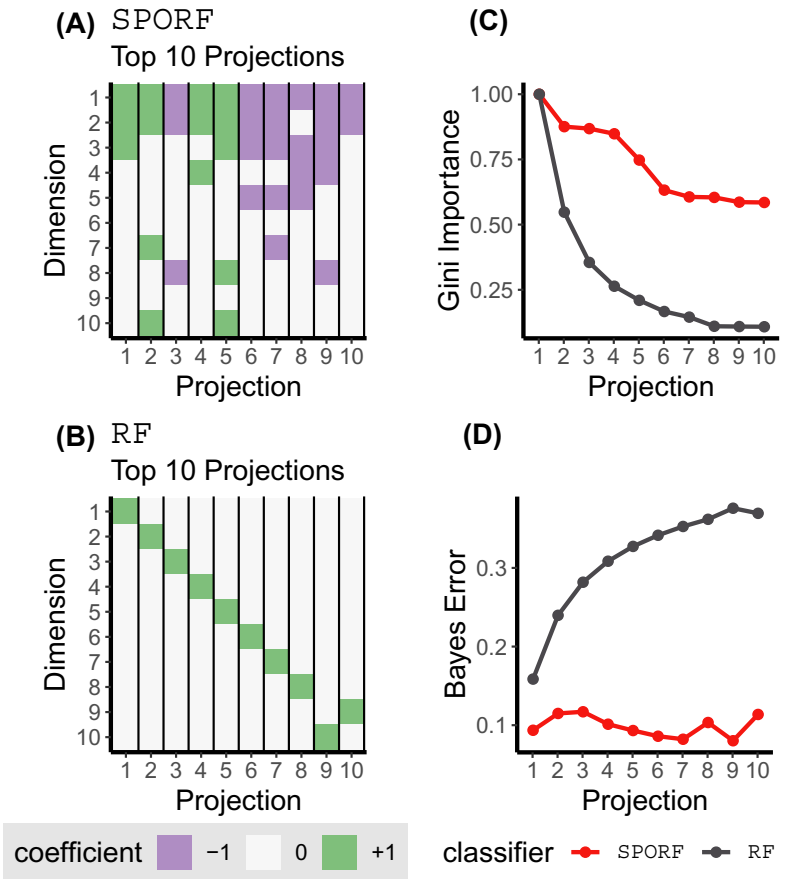

Figure 4: The ten projections with the highest Gini importance found by RF and SPORF on the Trunk problem with $p = 10, n = 1000$. (A): Visual representation of the top 10 projections identified by SPORF. The x-axis indicates the projection, sorted from highest Gini importance to lowest. The y-axis indicates the index of the ten canonical (observed) dimensions. The colors in the heat map indicate the linear coefficients of each canonical dimension that define each of the projections. (B): The same as (A), except for RF. (C): Comparison of the Gini importances of the 10 best projections found by each algorithm. (D): Comparison of the Bayes error rate of the 10 best projections found by each algorithm. The top 10 projections used in SPORF all have substantially lower Bayes error than those used in RF, indicating that SPORF learns interpretable informative features.

robust to a large number of noise dimensions while other oblique tree ensemble methods are not.

5.1. SPORF Exhibits Best Overall Classification Performance on a Large Suite of Benchmark Data Sets

SPORF compares favorably to RF, XGBoost, RR-RF, and CCF on a suite of 105 benchmark data sets from the UCI machine learning repository (Figure 5). This benchmark suite is a subset of the same problem sets previously used to conclude that RF outperformed >100 other algorithms (Fernández-Delgado et al., 2014) (16 were excluded for various reasons such as lack of availability; see Appendix C for preprocessing details).

Figure 5A shows pairwise comparisons of RF with SPORF (red), XGBoost (yellow), RR-RF (purple), and CCF (green) on the UCI data sets. Specifically, let $\kappa(\cdot)$ denote Cohen's kappa (fractional decrease in error rate over the chance error rate) for a particular classification algorithm. Here, error rates are estimated for each algorithm for each data set via five-fold cross-validation. Error rates for each data set are reported in Appendix D. Let $\Delta(\mathcal{A}) = \kappa(\texttt{RF}) - \kappa(\mathcal{A})$ be the difference between κ for some algorithm $\mathcal{A}$—either SPORF, XGBoost, RR-RF, or CCF—with κ(RF). Each beeswarm plot in 5(A) represents the distribution of $\Delta(\mathcal{A})$, denoted "Effect Size," over data sets. Comparisons are shown for the 65 numeric data sets (top), the 40 data sets having at least one categorical feature (middle), and all 105 data sets (bottom). A positive value on the x-axis indicates that RF performed better than the algorithm it is being compared to on a particular data set, while a negative value indicates it performed worse. Values on the y-axis greater than 10% were squashed to 10% and values less than -10% were squashed to -10% in order to improve visualization. Mean values are indicated by a black "x." As indicated by the downward skewing distribution, SPORF tends to outperform RF over all data sets, due in particular to its relative performance on the numeric data sets. RR-RF and CCF also tend to perform similar to or better than RF on the numeric data sets, but unlike SPORF they perform worse than RF on the categorical data sets; the oblique methods are likely sensitive to the one-hot encoding of categorical features. κ values for individual data sets and algorithms can be found in Table 1.

Additionally, we examined how frequently each algorithm ranked in terms of κ across the data sets. A rank of one indicates first place (best) on a particular data set and a rank of five indicates last place (worst). Histograms (in fraction of data sets) of the relative ranks are shown in Figure 5B. Overall, SPORF tends to outperform the other algorithms. This is despite the fact that XGBoost is tuned significantly more than SPORF in these comparisons (see Section 3.2 for details). Surprisingly, we find that RR-RF, one of the most recent methods to be proposed, has a strong tendency to perform the worst. One-sided Wilcoxon signed-rank tests were performed to determine whether SPORF performed significantly better than each of the other algorithms. Specifically, the null hypothesis was that the median κ value of SPORF is greater than that of each algorithm being compared to. P-values are shown for each algorithm compared with SPORF to the right of each histogram in Figure 5B. Over all data sets, we found that p-values were < 0.005 for every algorithm comparison with SPORF.

5.2. Identifying Default Hyperparameter Settings

While the hyperparameters λ and d of SPORF were tuned in this comparison, default hyperparameters can be of great value to researchers who use SPORF out of the box. This is especially true for those not familiar with the details of a particular algorithm or those

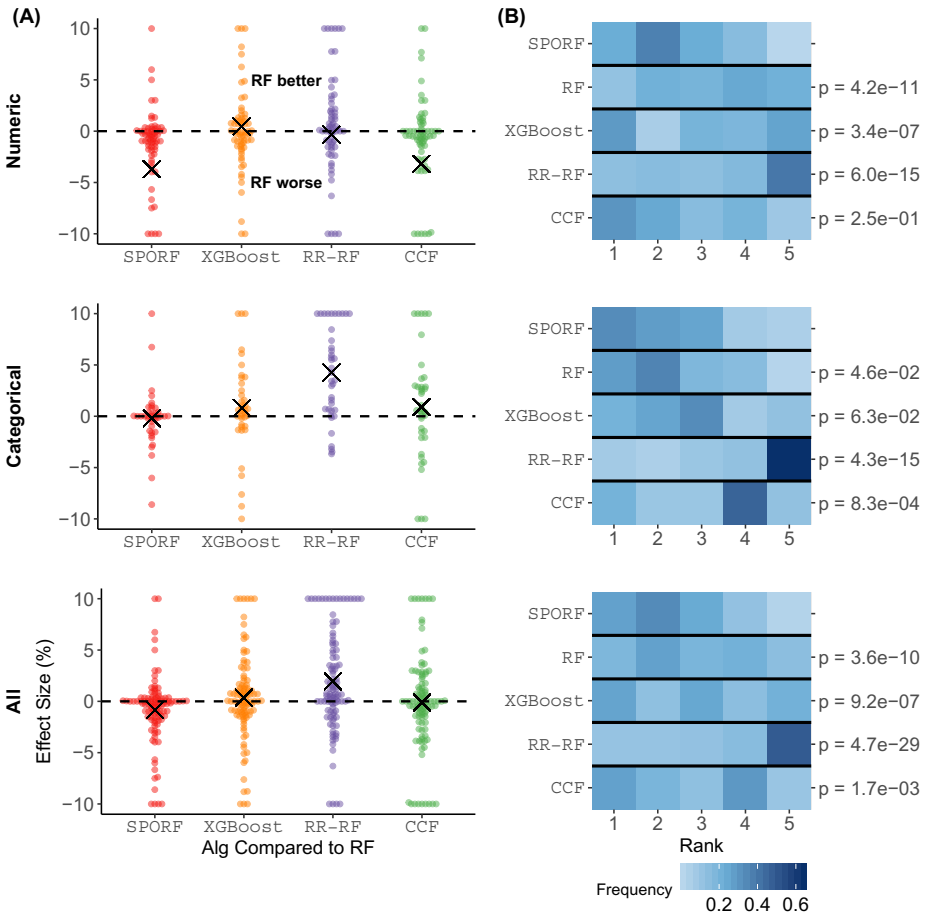

Figure 5: Pairwise comparisons of `RF` with `SPORF`, `XGBoost`, `RR-RF`, and `CCF` on the numeric data sets (top), categorical data sets (middle), and all data sets (numeric and categorical combined; bottom) from the UCI Machine Learning Repository (105 data sets total). (A): Beeswarm plots showing the distributions of classification performance relative to `RF` for various decision forest algorithms. Classification performance is measured by effect size, which is defined as $\kappa(\texttt{RF}) - \kappa(\mathcal{A})$, where κ is Cohen's kappa and $\mathcal{A}$ is one of the algorithms compared to `RF`. Each point corresponds to a particular data set. Mean effect sizes are indicated by a black "x." A negative value on the y-axis indicates `RF` performed worse than a particular algorithm. (B): Histograms of the relative ranks of the different algorithms, where a rank of 1 indicates best relative classification performance and 5 indicates worst. Color indicates frequency, as fraction of data sets. P-values correspond to testing that `RF`, `XGBoost`, `RR-RF`, and `CCF` performed worse than `SPORF`, using one-sided Wilcoxon signed-rank tests. Overall, `SPORF` tends to perform better than the other algorithms.

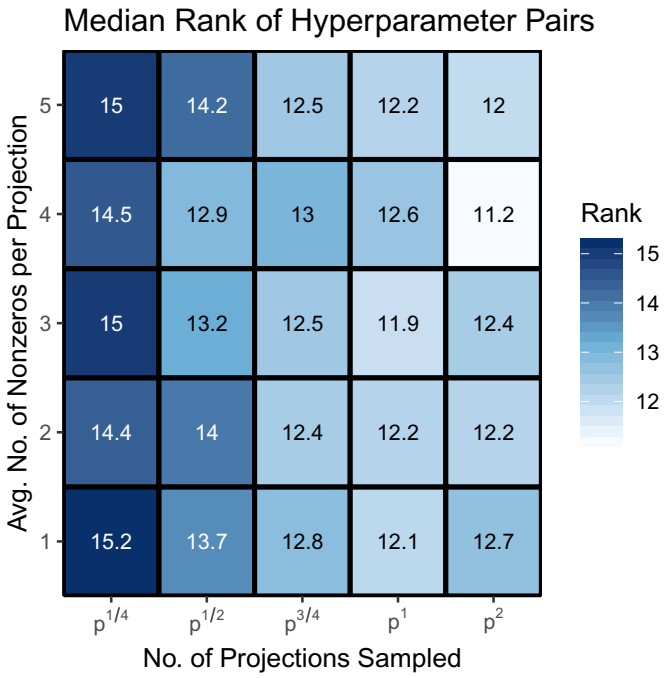

Figure 6: Median rank of SPORF's (d, λ) hyperparameter pairs on the UCI classification data sets (lower is better). Although $(p^2, 4/p)$ is the best performance-wise, we select $(p, 3/p)$ as the default because of a good balance between accuracy and training time.

having limited time and computational budget. Therefore, we sought suitable default values for λ and d based on classification performance on the UCI data sets. For each data set, for each fold the hyperparameter settings are ranked based on κ computed on the held out set. A rank of n indicates n^{th} place (i.e. first place indicates largest κ). Ties in the ranking procedure are handled by assigning all ties the same averaged rank. For example, consider the set of real numbers $\{a_1, a_2, a_3\}$ such that $a_1 > a_2 = a_3$. Then a_1 would be assigned a rank of three and a_2 and a_3 would both be assigned a rank of $(1+2)/2 = 1.5$. The rank of each hyperparameter pair was averaged over the five folds. Finally, for each hyperparameter pair, the median rank is computed over the data sets. The median rank for each hyperparameter setting is depicted in Figure 6. The results here suggest that $d = p^2$ and $\lambda = 4/p$ is the best default setting for SPORF with respect to classification performance. However, we choose the setting $d = p$ and $\lambda = 3/p$ as the default values in our implementation because it requires substantially less training time for moderate to large p at the expense of only a slightly greater tendency to perform worse on the UCI data sets.

5.3. SPORF is Robust to Many Noise Dimensions

Next, we investigated the effect of adding a varying number of noise dimensions to the UCI benchmark data sets. For each of the 105 UCI data sets used in the previous experiment,

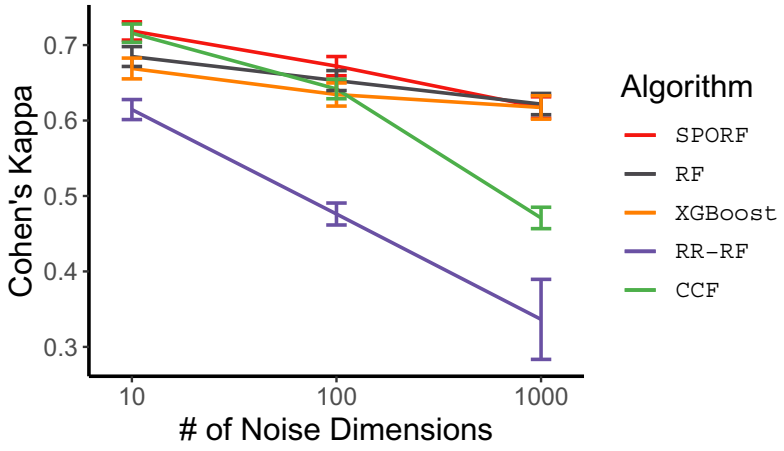

Figure 7: Comparison of classification performance on the UCI benchmark data sets with a varying number of Gaussian noise dimensions added. The x-axis represents the number of noise dimensions added. The y-axis represents the average of Cohen's kappa value over all data sets ($\pm SEM$). `SPORF` is always tied for the best performance. `CCF` and `RR-RF` are more sensitive to additional noise dimensions.

D_{noise} standard Gaussian dimensions were appended to the input matrix, for $D_{noise} \in \{10, 100, 1000\}$. Algorithm comparisons were then performed in the same way as before.

Figure 7 shows the overall classification performance of `SPORF`, `RF`, `XGBoost`, and `CCF` for each value of D_{noise}. Each of the points plotted represents the mean Cohen's kappa ($\pm$ SEM) over all data sets. For all values of D_{noise}, `SPORF` ties for best classification performance. Notably, `CCF` performs about as well as `SPORF` when there is little additional noise, but degrades substantially when many noise dimensions are added. This suggests that using supervised linear procedures to compute splits may lead to poor generalization, likely because the learned features have overfit to the noise dimensions. `RR-RF` degrades even more rapidly than does `CCF` with increasing numbers of noise dimensions. This can be explained by the fact that features derived from random rotations, which are *dense* linear projections, have very low probability of being informative in the presence of many noise dimensions.

6. Computational Efficiency and Scalability of `SPORF`

Computational efficiency and scalability are often as important as accuracy in the choice of machine learning algorithms, especially for big data. In this section we demonstrate that `SPORF`, with an appropriate choice of hyperparameter settings, scales similarly to `RF` with respect to sample size and number of features. Furthermore, we show that our open source implementation is computationally competitive with leading implementations of decision tree ensemble algorithms.

6.1. Theoretical Time Complexity

The time complexity of an algorithm characterizes how the theoretical processing time for a given input relies on both the hyper-parameters of the algorithm and the characteristics of the input. Let T be the number of trees, n the number of training samples, p the number of features in the training data, and d the number of features sampled at each split node. The average case time complexity of constructing an `RF` is $\mathcal{O}(Tdn\log^2 n)$ (Louppe, 2014). The $dn\log n$ accounts for the sorting of d features at each node. The additional $\log n$ accounts for both the reduction in node size at lower levels of the tree and the average number of nodes produced. `RF`'s near linear complexity shows that a good implementation will scale nicely with large input sizes, making it a suitable algorithm to process big data. `SPORF`'s average case time complexity is similar to `RF`'s, the only difference being that there is an additional term representing a sparse matrix multiplication that is required in each node. This makes `SPORF`'s complexity $\mathcal{O}(Tdn\log^2 n + Tdnp\lambda))$, where λ is the fraction of nonzeros in the $p \times d$ random projection matrix. We generally let λ be close to $1/p$, giving a complexity of $\mathcal{O}(Tdn\log^2 n)$, which is the same as for `RF`. Of note, in `RF` d is constrained to be no greater than p, the dimensionality of the data. `SPORF`, on the other hand, does not have this restriction on d. Therefore, if d is selected to be greater than p, `SPORF` may take longer to train. However, $d > p$ often results in improved classification performance.

6.2. Theoretical Space Complexity

The space complexity of an algorithm describes how the theoretical maximum memory usage during runtime scales with the number of inputs and hyperparameters. Let c be the number of classes and T, p, and n be defined as in Section 6.1. Building a single tree requires the data matrix to be kept in memory, which is $\mathcal{O}(np)$. During an attempt to split a node, two c-length arrays store the counts of each class to the left and to the right of the candidate split point. These arrays are used to evaluate the decrease in Gini impurity or entropy. Additionally, a series of random sparse projection vectors are sequentially assessed. Each vector has less than p nonzeros. Therefore this term is dominated by the np term. Assuming trees are fully grown, meaning each leaf node contains a single data point, the tree has $2n$ nodes in total. This term gets dominated by the np term as well. Therefore, the space complexity to build a `SPORF` is $\mathcal{O}(T(np + c))$. This is the same as that of `RF`.

6.3. Theoretical Storage Complexity

Storage complexity is the disk space required to store a forest, given the inputs and hyperparameters. Assume that trees are fully grown. For each leaf node, only the class label of the training data point contained within the node is stored, which is $\mathcal{O}(1)$. For each split node, the split dimension index and threshold are stored, which are also both $\mathcal{O}(1)$. Therefore, the storage complexity of a `RF` is $\mathcal{O}(Tn)$.

For a `SPORF`, the only aspect that differs is that a (sparse) vector is stored at each split node rather than a single split dimension index. Let z denote the average number of nonzero entries in a vector projection stored at each split node. Storage of this vector at each split node requires $\mathcal{O}(z)$ memory. Therefore, the storage complexity of a `SPORF` is $\mathcal{O}(Tnz)$. z is a random variable whose prior is governed by λ, which is typically set to $1/p$. The posterior

mean of z is determined also by the data; empirically it is close to $z = 1$. Therefore, in practice, the storage complexity of SPORF is close to that of RF.

6.4. Empirical Computational Efficiency and Scalability

Below we assess computational performance of SPORF. We do so by first comparing it to RF and F-RC. In order to fairly compare, all methods use our own R implementation. Then we compare our R implementation to other highly optimized implementations of decision tree ensembles.

6.4.1. Implementation Details

We use our own R implementation for evaluations of RF, SPORF, F-RC, and RR-RF (Browne et al., 2018). It was more difficult to modify one of the existing popular tree learning implementations due to the particular way in which they operate on the input data. In all of the popular axis-aligned tree learning implementations, each feature in the input data matrix is sorted just once prior to inducing a tree, and the tree induction procedure operates directly on this presorted data. Since trees in a SPORF include splitting on new features consisting of linear combinations of the original features, pre-sorting the data is not an option. Therefore our implementation is written from scratch in mostly native R. The code has been extensively profiled and optimized for speed and memory performance. Profiling revealed the primary performance bottleneck to be the portion of code responsible for finding the best split. In order to improve speed, this portion of code was implemented in C++ and integrated into R using the Rcpp package (Eddelbuettel, 2018). Further speedup is achieved through multicore parallelization of tree construction and byte-compilation via the R compiler package.

XGBoost is evaluated using the R implementation available on CRAN (Chen, 2018). CCF is evaluated using the authors' openly available MATLAB implementation (Rainforth and Wood, 2015).

6.4.2. Comparison of Algorithms Using the Same Implementation

Figure 8A shows the training times of RF, F-RC, and SPORF on the sparse parity problem. The reported training times correspond to the best hyperparameter settings for each algorithm. Experiments are run using an Intel Xeon E5-2650 v3 processors clocked at 2.30GHz with 10 physical cores, 20 threads, and 250 GB DDR4-2133 RAM. The operating system is Ubuntu 16.04. F-RC is the slowest, RF is the fastest, and SPORF is in between. While not shown, we note that a similar trend holds for the orthant problem. Figure 8B shows that when the hyperparameter d of SPORF and F-RC is the same as that for RF, training times are comparable. However, training time continues to increase as d exceeds p for SPORF and F-RC, which largely accounts for the trend seen in Figure 8A. Figure 8C indicates that this additional training time comes with the benefit of substantially improved accuracy. Restricting d to be no greater than p for SPORF in this setting would still perform noticeably better than RF at no additional cost in training time. Therefore, SPORF does not trade off accuracy for time. Rather, for a fixed computational budget, it achieves better accuracy, and if allowed to use more computation, further improves accuracy.

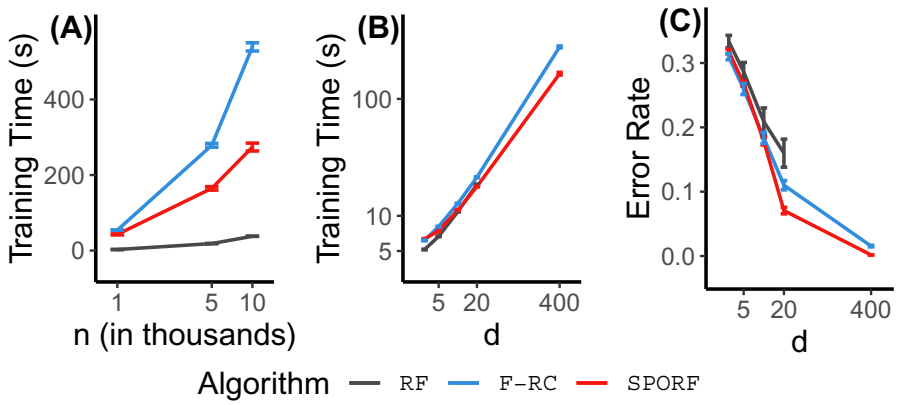

Figure 8: Comparison of training times of RF, SPORF, and F-RC on the 20-dimensional sparse parity setting. (A): Dependency of training time using the best set of hyperparameters (y-axis) on the number of training samples (x-axis) for the sparse parity problem. (B): Dependency of training time (y-axis) on the number of projections sampled at each split node (x-axis) for the sparse parity problem with $n = 5000$. (C): Dependency of error rate (y-axis) on the number of projections sampled at each split node (x-axis) for the sparse parity problem with $n = 5000$. SPORF and F-RC can sample many more than p projections, unlike RF. As seen in panels (B) and (C), increasing d above p meaningfully improves classification performance at the expense of larger training times. However, comparing error rates and training times at $d = 20$, SPORF can classify substantially better than RF even with no additional cost in training time.

6.4.3. Comparison of Training and Prediction Times for Different Implementations

We developed and maintain an open multi-core R implementation of SPORF , which is hosted on CRAN (Browne et al., 2018). We compare both speed of training and strong scaling of our implementation to those of the R Ranger (Wright, 2018) and XGBoost (Chen, 2018) packages, which are currently two of the fastest, parallelized decision tree ensemble software packages available. Strong scaling is the time needed to train a forest with one core divided by the time needed to train a forest with multiple cores. Ranger offers a fast multicore version of RF that has been extensively optimized for runtime performance. XGBoost offers a fast multicore version of gradient boosted trees, and computational performance is optimized for shallow trees. Both Ranger and XGBoost are C++ implementations with R wrappers, whereas our SPORF implementation is almost entirely native R. Hyperparameters are chosen for each implementation so as to make the comparisons fair. For all implementations, trees are grown to full depth, 100 trees are constructed, and $d = \sqrt{p}$ features sampled at each node. For SPORF, $\lambda = 1/p$. Experiments are run using four Intel Xeon E7-4860 v2 processors clocked at 2.60GHz, each processor having 12 physical cores and 24 threads. The amount of available memory is 1 TB DDR3-1600. The operating system is Ubuntu 16.04. Comparisons use three openly available large data sets:

MNIST The MNIST data set (Lecun et al.) has 60,000 training observations and 784 (28x28) features. For a small number of cores, SPORF is faster than XGBoost but slower than Ranger (Figure 9A). However, when 48 cores are used, SPORF is as fast as Ranger and still faster than XGBoost.

Higgs The Higgs data set (https://www.kaggle.com/c/higgs-boson) has 250,000 training observations and 31 features. SPORF is as fast as ranger and faster than XGBoost when using 48 cores (Figure 9B).

p53 The p53 data set (https://archive.ics.uci.edu/ml/datasets/p53+Mutants) has 31,159 training observations and 5,409 features. Figure 9C shows a similar trend as for MNIST. For this data set, utilizing additional resources with SPORF does not provide as much benefit due to the classification task being too easy (all algorithms achieve perfect classification accuracy)—the trees are shallow, causing the overhead cost of multithreading to outweigh the speed increase as a result of parallelism.

Strong scaling is the relative increase in speed of using multiple cores over that of using a single core. In the ideal case, the use of N cores would produce a factor N speedup. SPORF has the best strong scaling on MNIST (Figure 9D) and Higgs (Figure 9E), while it has strong scaling in between that of Ranger and XGBoostON the p53 data set (Figure 9F). This is due to the simplicity of the p53 data set, as discussed above.

Prediction times can be just as, or even more important than training times in certain applications. For example, electron microscopy-based connectomics can acquire multi-petabyte data sets that require classification of each voxel (Motta et al., 2019). Moreover, recent automatic hyperparameter tuning suites incorporate runtime in their evaluations, which leverage out-of-sample prediction accuracy (Falkner et al., 2018). Thus, accelerating prediction times can improve the effectiveness of hyperparameter sweeps.

Figure 10 compares the prediction times of the various implementations on the same three data sets. In addition to our standard SPORF prediction implementation, we also compare a "Forest Packing" prediction implementation (Browne et al., 2019). Briefly, Forest Packing is a procedure performed after a forest has been grown that reduces prediction latency by reorganizing and compacting the forest data structure. The number of test points used for the Higgs, MNIST, and p53 data sets is 50,000, 10,000, and 6,000, respectively. Predictions were made sequentially without batching using a single core. SPORF is significantly faster than Ranger on the Higgs and MNIST data sets, and only marginally slower on the p53 data set. XGBoost is much faster than both SPORF and Ranger, which is due to the fact that the XGBoost algorithm constructs much shallower trees than the other methods. Most notably, the Forest Packing procedure, which "packs" the trees learned by SPORF, makes predictions roughly ten times faster than XGBoost and over 100 times faster than the standard SPORF on all three data sets.

7. Conclusion

In this work we showed that existing oblique splitting extensions to RF forfeit some of the nice properties of RF, while achieving improved performance in certain settings. We therefore introdced SPORF which was designed to preserve the desirable properties of both RF and

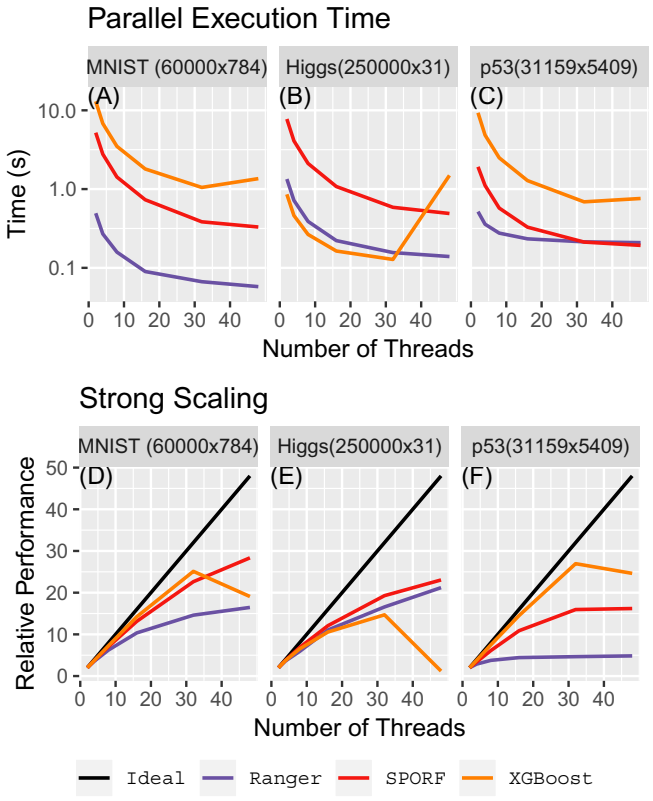

Figure 9: (A-C): The per-tree training time for three large real world data sets. Training was performed using matching parameters where possible, and default parameters otherwise. `SPORF`'s performance—even though it is written mostly in native R, as compared to the other optimized C++ codes—is comparable to the highly optimized `XGBoost` and Ranger and even outperforms `XGBoost` on two of the data sets. (D-F): Strong scaling is the time needed to train a forest with one core divided by the time needed to train a forest with multiple cores. This is a measurement of a system's ability to efficiently use additional resources. `SPORF` is able to scale well over the entire range of tested cores, whereas `XGBoost` has sharp drops in scalability during which it is unable to use additional threads due to characteristics of the given data sets. The p53 data set, despite having a large number of dimensions, is easily classifiable, which leads to short trees. The p53 strong scaling plot shows that when trees are short, the overhead of multithreading prevents `SPORF` from efficiently using the additional resources.

oblique forest methods, rendering it statistically robust, computationally efficient, scalable, and interpretable. This work only focused on classification; we also have a preliminary implementation for regression, which seems to perform similarly to `RF` on a suite of regression benchmark data sets. Future work will investigate the behavior and performance of `SPORF` on univariate and multivariate regression tasks.

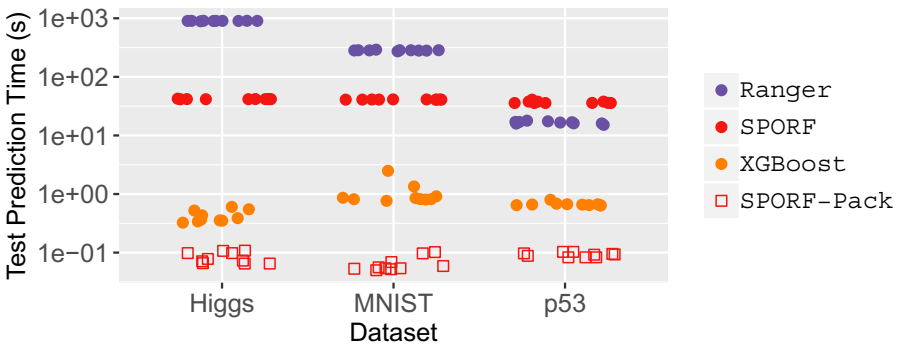

Figure 10: Comparison of test set prediction times. Forest Packing results show a 10x speed up in real time prediction scenarios. Test set sizes: Higgs, 50,000 observations; MNIST, 10,000 observations; p53, 6,000 observations. Predictions were made sequentially without batching.

One limitation of using sparse random projections to generate the candidate oblique splits is that it will never find informative splits in cases for which the signal is contained in a dense linear combination of features or nonlinear combinations of features. In such cases, supervised computation of split directions may be more suitable. Perhaps a decision forest method that evaluates both sparse random projections and dense supervised projections at each split node could further improve performance in such settings.

On a more theoretical note, we demonstrated that `SPORF` achieves almost perfect classification accuracy on a problem for which Biau et al. (2008) proved that `RF` cannot achieve better than an error rate of 1/6. This raises a question as to whether it is possible to construct a problem in which `RF` is consistent and `SPORF` is not. Or could it be the case that `SPORF` is always consistent when `RF` is? The consistency theorems by Scornet et al. (2015) for `RF` in the case of additive regression models should be extendable to `SPORF` with some minor modifications—their proofs rely on clever adaptations of classical consistency results for data-independent partitioning classifiers, which are agnostic to whether the splits are axis-aligned or not. Another factor that dictates the lower bound of error rate, as Breiman (2001) proved, is the relative balance between the strength and correlation of trees. Our investigation of strength and correlation on the Sparse Parity, Orthant, and Trunk simulations is offered in Appendix E. The results suggest that `SPORF` can outperform other algorithms because of stronger trees and/or less correlated trees. Therefore, `SPORF` perhaps offers more flexible control over the balance between tree strength and correlation, thereby allowing it to adapt better to different problems.

Our implementation of `SPORF` is as computationally efficient and scalable or more so than existing tree ensemble implementations. Additionally, our implementation can realize many previously proposed tree ensemble methods by allowing the user to define how random projections are generated. Open source code is available at `https://neurodata.io/sporf/`,

Algorithm 1 Learning a `SPORF` classification tree.

Input: (1) $\mathcal{D}_n = (\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{n \times p} \times \mathcal{Y}^n$: training data (2) Θ: set of split eligibility criteria
Output: A `SPORF` decision tree T

function T = GROWTREE$(\mathbf{X}, \mathbf{y}, \Theta)$
  $c = 1$ ▷ c is the current node index
  $M = 1$ ▷ M is the number of nodes currently existing
  $S^{(c)} = \text{bootstrap}(\{1, ..., n\})$ ▷ $S^{(c)}$ is the indices of the observations at node c
  while $c < M + 1$ **do** ▷ visit each of the existing nodes
    $(\mathbf{X}', \mathbf{y}') = (\mathbf{x}_i, y_i)_{i \in S^{(c)}}$ ▷ data at the current node
    for $k = 1, \ldots, K$ **do** $n_k^{(c)} = \sum_{i \in S^{(c)}} I[y_i = k]$ **end for** ▷ class counts
    if Θ satisfied **then** ▷ do we split this node?
      $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_d] \sim f_{\mathbf{A}}$ ▷ sample random $p \times d$ matrix as defined in 3.1
      $\widetilde{\mathbf{X}} = \mathbf{X}'\mathbf{A} = (\widetilde{\mathbf{x}}_i)_{i \in S^{(c)}}$ ▷ random projection into new feature space
      $(j^*, t^*) = \text{findbestsplit}(\widetilde{\mathbf{X}}, \mathbf{y}')$ ▷ Algorithm 2
      $S^{(M+1)} = \{i : \widetilde{\mathbf{x}}_i \cdot \mathbf{a}_{j^*} \leq t^* \quad \forall i \in S^{(c)}\}$ ▷ assign to left child node
      $S^{(M+2)} = \{i : \widetilde{\mathbf{x}}_i \cdot \mathbf{a}_{j^*} > t^* \quad \forall i \in S^{(c)}\}$ ▷ assign to right child node
      $\mathbf{a}^{*(c)} = \mathbf{a}_{j^*}$ ▷ store best projection for current node
      $\tau^{*(c)} = t^*$ ▷ store best split threshold for current node
      $\kappa^{(c)} = \{M + 1, M + 2\}$ ▷ node indices of children of current node
      $M = M + 2$ ▷ update the number of nodes that exist
    else
      $(\mathbf{a}^{*(c)}, \tau^{*(c)}, \kappa^{*(c)}) = \text{NULL}$
    end if
    $c = c + 1$ ▷ move to next node
  end while
  return $(S^{(1)}, \{\mathbf{a}^{*(c)}, \tau^{*(c)}, \kappa^{(c)}, \{n_k^{(c)}\}_{k \in \mathcal{Y}}\}_{c=1}^{m-1})$
end function

including both the R package discussed here, and a C++ version with both R and Python bindings that we are actively developing.

Acknowledgments

This work is graciously supported by the Defense Advanced Research Projects Agency (DARPA) SIMPLEX program through SPAWAR contract N66001-15-C-4041, DARPA GRAPHS N66001-14-1-4028, and DARPA Lifelong Learning Machines program through contract FA8650-18-2-7834.

Appendix A. Algorithms

Algorithm 2 Finding the best node split. This function is called by growtree (Alg 1) at every split node. For each of the p dimensions in $\mathbf{X} \in \mathbb{R}^{n \times p}$, a binary split is assessed at each location between adjacent observations. The dimension j^* and split value τ^* in j^* that best split the data are selected. The notion of "best" means maximizing some choice in scoring function. In classification, the scoring function is typically the reduction in Gini impurity or entropy. The increment function called within this function updates the counts in the left and right partitions as the split is incrementally moved to the right.

Input: (1) $(\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{n \times p} \times \mathcal{Y}^n$, where $\mathcal{Y} = \{1, \ldots, K\}$
Output: (1) dimension j^*, (2) split value τ^*

function $(j^*, \tau^*) =$ FINDBESTSPLIT$(\mathbf{X}, \mathbf{y})$
  for $j = 1, \ldots, p$ **do**
    Let $\mathbf{x}^{(j)} = (x_1^{(j)}, \ldots, x_n^{(j)})^T$ be the jth column of $\mathbf{X}$.
    $\{m_i^j\}_{i \in [n]} = \text{sort}(\mathbf{x}^{(j)})$ ▷ m_i^j is the index of the i^{th} smallest value in $\mathbf{x}^{(j)}$
    $t = 0$ ▷ initialize split to the left of all observations
    $n' = 0$ ▷ number of observations left of the current split
    $n'' = n$ ▷ number of observations right of the current split
    for $k = 1, \ldots, K$ **do**
      $n_k = \sum_{i=1}^{n} I[y_i = k]$ ▷ total number of observations in class k
      $n'_k = 0$ ▷ number of observations in class k left of the current split
      $n''_k = n_k$ ▷ number of observations in class k right of the current split
    end for
    for $t = 1, \ldots, n-1$ **do** ▷ assess split location, moving right one at a time
      $(\{(n'_k, n''_k)\}, n', n'', y_{m_t^j}) = \text{increment}(\{(n'_k, n''_k)\}, n', n'', y_{m_t^j})$
      $Q^{(j,t)} = \text{score}(\{(n'_k, n''_k)\}, n', n'')$ ▷ measure of split quality
    end for
  end for
  $(j^*, t^*) = \underset{j,t}{\text{argmax}}\, Q^{(j,t)}$
  for $i = 0, 1$ **do** $c_i = m_{t^*+i}^{j^*}$ **end for**
  $\tau^* = \frac{1}{2}(x_{c_0}^{(j^*)} + x_{c_1}^{(j^*)})$ ▷ compute the actual split location from the index j^*
  return (j^*, τ^*)
end function

Appendix B. Hyperparameter Tuning

Hyperparameters in `XGBoost` are tuned via grid search using the R caret package. The values tried for each hyperparameter are based on suggestions by Owen Zhang (`https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions`), a research data scientist who has had many successes in data science competitions using `XGBoost`:

- nrounds: 100, 1000
- subsample: 0.5, 0.75, 1
- eta: 0.001, 0.01
- colsample_bytree: 0.4, 0.6, 0.8, 1
- min_child_weight: 1
- max_depth: 4, 6, 8, 10, 100000
- gamma: 0

Selection of the hyperparameter values is based on minimization of a five-fold cross-validation error rate.

Appendix C. Real Benchmark Data Sets

We use 105 benchmark data sets from the UCI machine learning repository for classification. These data sets are most of the data sets used in Fernández-Delgado et al. (2014); some were removed due to licensing or unavailability issues. We noticed certain anomalies in Fernández-Delgado et al. (2014)'s pre-processed data, so we pre-processed the raw data again as follows.

1. **Remove of nonsensical features**. Some features, such as unique sample identifiers, or features that were the same value for every sample, were removed.

2. **Impute missing values**. The R randomForest package was used to impute missing values. This method was chosen because it is nonparametric and is one of the few imputation methods that can natively impute missing categorical entries.

3. **One-hot-encode categorical features**. Most classifiers cannot handle categorical data natively. Given a categorical feature with possible values $\{c_1, \ldots, c_m\}$, we expand to m binary features. If a data point has categorical value $c_k, \forall k \in 1, \ldots, m$ then the k^{th} binary feature is assigned a value of one and zero otherwise.

4. **Integer encoding of ordinal features**. Categorical features having order to them, such as "cold", "luke-warm", and "hot", were numerically encoded to respect this ordering with integers starting from 1.

5. **Standardization of the format**. Lastly, all data sets were stored as CSV files, with rows representing observations and columns representing features. The class labels were placed as the last column.

6. **Five-fold parition**. Each data set was randomly divided into five partitions for five-fold cross-validation. Partitions preserved the relative class frequencies by stratification. Each partition included a different 20% of the data for testing.

Appendix D. Data Tables

Dataset	n	p_{num}	p_{cat}	C	5-fold CV Cohen's κ SPORF	RF	XGBoost	RR-RF	CCF
abalone	4177	7	1	28	11.2 ± 0.8	11 ± 0.8	$\mathbf{11.8 \pm 0.4}$	9.1 ± 0.5	10.4 ± 0.7
acute_inflammation_task_1	120	6	0	2	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$	95 ± 2	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
acute_inflammation_task_2	120	6	0	2	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$	87 ± 9	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
adult	32561	7	7	2	82.38 ± 0.29	82.42 ± 0.28	$\mathbf{83.45 \pm 0.28}$	78.16 ± 0.39	80.2 ± 0.23
annealing	798	27	5	5	$\mathbf{98 \pm 1}$	$\mathbf{98 \pm 1}$	97 ± 1	91 ± 2	97 ± 1
arrhythmia	452	279	0	13	72 ± 2	$\mathbf{74 \pm 2}$	72 ± 2	61 ± 2	66 ± 1
audiology_std	200	68	1	24	75 ± 4	73 ± 5	73 ± 5	65 ± 4	$\mathbf{78 \pm 5}$
balance_scale	625	4	0	3	$\mathbf{94 \pm 1}$	77 ± 3	83 ± 1	82 ± 1	90 ± 1
balloons	16	4	0	2	40 ± 20	50 ± 20	10 ± 30	$\mathbf{60 \pm 10}$	40 ± 20
bank	4521	11	5	2	91.7 ± 0.1	$\mathbf{91.9 \pm 0.1}$	$\mathbf{91.9 \pm 0.1}$	91.3 ± 0.4	91.6 ± 0.3
blood	748	4	0	2	71 ± 1	$\mathbf{72 \pm 1}$	$\mathbf{72 \pm 1}$	71 ± 1	70 ± 1
breast_cancer	286	7	2	2	$\mathbf{61 \pm 2}$	60 ± 3	58 ± 1	54 ± 4	57 ± 3
breast_cancer-wisconsin	699	9	0	2	$\mathbf{96 \pm 2}$	$\mathbf{96 \pm 2}$	95 ± 2	$\mathbf{96 \pm 2}$	$\mathbf{96 \pm 2}$
breast_cancer-wisconsin-diag	569	30	0	2	96 ± 1	94 ± 1	94 ± 1	96 ± 1	$\mathbf{97 \pm 1}$
breast_cancer-wisconsin-prog	198	33	0	2	$\mathbf{73 \pm 2}$	69 ± 3	72 ± 2	72 ± 2	72 ± 3
car	1728	6	0	4	96.5 ± 0.2	93.1 ± 0.8	96.5 ± 0.4	81.5 ± 1.4	$\mathbf{96.9 \pm 0.6}$
cardiotocography_task_1	2126	21	0	10	83.5 ± 0.5	82.5 ± 0.6	$\mathbf{84.4 \pm 0.5}$	74.7 ± 0.8	81.3 ± 1.1
cardiotocography_task_2	2126	21	0	3	93.9 ± 0.5	93.6 ± 0.6	$\mathbf{94.5 \pm 0.4}$	90.1 ± 0.8	92.2 ± 0.5
chess_krvk	28056	0	6	18	84.01 ± 0.18	77.99 ± 0.18	$\mathbf{86.76 \pm 0.35}$	59.17 ± 0.13	82.62 ± 0.22
chess_krvkp	3196	35	1	2	$\mathbf{99.1 \pm 0.2}$	98.8 ± 0.2	98.8 ± 0.3	95.7 ± 0.6	98.8 ± 0.1
congressional_voting	435	16	0	2	94 ± 2	94 ± 2	$\mathbf{96 \pm 1}$	94 ± 1	95 ± 2
conn_bench-sonar-mines-rocks	208	60	0	2	72 ± 4	72 ± 6	$\mathbf{77 \pm 7}$	70 ± 3	75 ± 5
conn_bench-vowel-deterding	528	11	0	11	97 ± 0	96 ± 1	90 ± 2	97 ± 0	$\mathbf{98 \pm 1}$
contrac	1473	8	1	3	29.3 ± 2.8	26.5 ± 1.9	$\mathbf{31.6 \pm 1.7}$	24.9 ± 1.4	28 ± 1.5
credit_approval	690	10	5	2	77 ± 1	$\mathbf{78 \pm 1}$	77 ± 1	74 ± 2	75 ± 2
dermatology	366	34	0	6	$\mathbf{98 \pm 1}$	$\mathbf{98 \pm 1}$	97 ± 1	96 ± 0	96 ± 1
ecoli	336	7	0	8	$\mathbf{83 \pm 1}$	81 ± 2	81 ± 1	$\mathbf{83 \pm 1}$	81 ± 1
flags	194	22	6	8	57 ± 3	$\mathbf{58 \pm 3}$	57 ± 4	46 ± 5	54 ± 2
glass	214	9	0	6	64 ± 5	$\mathbf{67 \pm 6}$	65 ± 5	56 ± 7	66 ± 7

haberman_survival	306	3	0	2	$\mathbf{63 \pm 3}$	61 ± 3	$\mathbf{63 \pm 2}$	57 ± 3	54 ± 4
hayes_roth	132	0	4	3	68 ± 7	68 ± 7	64 ± 8	66 ± 7	$\mathbf{69 \pm 7}$
heart_cleveland	303	10	3	5	47 ± 2	48 ± 2	47 ± 1	$\mathbf{51 \pm 1}$	45 ± 2
heart_hungarian	294	10	3	2	$\mathbf{89 \pm 3}$	87 ± 2	87 ± 3	81 ± 3	85 ± 4
heart_switzerland	123	10	3	5	-5 ± 6	1 ± 4	-5 ± 6	2 ± 10	$\mathbf{6 \pm 10}$
heart_va	200	10	3	5	13 ± 4	13 ± 6	15 ± 5	$\mathbf{17 \pm 5}$	15 ± 3
hepatitis	155	19	0	2	44 ± 7	42 ± 15	37 ± 7	38 ± 7	$\mathbf{54 \pm 10}$
hill_valley	606	100	0	2	$\mathbf{100 \pm 0}$	12 ± 2	22 ± 4	88 ± 2	$\mathbf{100 \pm 0}$
hill_valley-noise	606	100	0	2	$\mathbf{90 \pm 3}$	3 ± 6	6 ± 3	65 ± 2	89 ± 2
horse_colic	300	17	4	2	76 ± 3	74 ± 4	$\mathbf{80 \pm 1}$	77 ± 3	77 ± 3
ilpd_indian-liver	583	10	0	2	60 ± 2	59 ± 2	60 ± 1	62 ± 3	$\mathbf{63 \pm 1}$
image_segmentation	210	19	0	7	$\mathbf{93 \pm 3}$	92 ± 3	91 ± 2	87 ± 3	$\mathbf{93 \pm 3}$
ionosphere	351	34	0	2	85 ± 2	82 ± 2	81 ± 1	$\mathbf{88 \pm 1}$	86 ± 2
iris	150	4	0	3	91 ± 2	94 ± 3	92 ± 2	94 ± 2	$\mathbf{96 \pm 2}$
led_display	1000	7	0	10	68.2 ± 1.3	68.6 ± 1.6	$\mathbf{69.5 \pm 1.2}$	67.9 ± 1.4	67.6 ± 1.3
lenses	24	4	0	3	50 ± 20	40 ± 20	$\mathbf{60 \pm 20}$	30 ± 10	40 ± 20
letter	20000	16	0	26	96.85 ± 0.13	96.37 ± 0.11	96.32 ± 0.05	95.24 ± 0.22	$\mathbf{97.67 \pm 0.18}$
libras	360	90	0	15	85 ± 2	80 ± 2	76 ± 2	84 ± 2	$\mathbf{90 \pm 2}$
low_res-spect	531	100	1	48	59 ± 3	51 ± 2	48 ± 3	48 ± 1	$\mathbf{62 \pm 1}$
lung_cancer	32	13	43	3	30 ± 10	$\mathbf{40 \pm 10}$	30 ± 20	20 ± 10	0 ± 10
magic	19020	10	0	2	$\mathbf{82.65 \pm 0.3}$	81.48 ± 0.39	82.35 ± 0.24	79.55 ± 0.22	81.92 ± 0.37
mammographic	961	3	2	2	$\mathbf{69 \pm 2}$	$\mathbf{69 \pm 1}$	$\mathbf{69 \pm 1}$	57 ± 1	61 ± 2
molec_biol-promoter	106	0	57	4	$\mathbf{40 \pm 2}$	36 ± 5	32 ± 2	15 ± 4	19 ± 7
molec_biol-splice	3190	0	60	3	93 ± 0.7	93.2 ± 0.7	$\mathbf{94.2 \pm 0.5}$	68.9 ± 0.6	92.8 ± 0.9
monks_1	124	2	4	2	$\mathbf{98 \pm 2}$	$\mathbf{98 \pm 2}$	82 ± 3	70 ± 5	81 ± 5
monks_2	169	2	4	2	37 ± 5	37 ± 5	45 ± 6	30 ± 3	$\mathbf{61 \pm 4}$
monks_3	122	2	4	2	86 ± 4	86 ± 4	81 ± 3	$\mathbf{87 \pm 5}$	81 ± 4
mushroom	8124	7	15	2	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$	99.8 ± 0.1	99.9 ± 0	$\mathbf{100 \pm 0}$
musk_1	476	166	0	2	80 ± 2	79 ± 4	80 ± 3	79 ± 2	$\mathbf{83 \pm 2}$
musk_2	6598	166	0	2	97.4 ± 0.3	97.3 ± 0.4	$\mathbf{98.2 \pm 0.2}$	95.1 ± 0.5	97.7 ± 0.4
nursery	12960	6	2	5	$\mathbf{99.97 \pm 0.02}$	99.71 ± 0.05	99.91 ± 0.05	96.2 ± 0.08	99.92 ± 0.04
optical	3823	64	0	10	98.1 ± 0.2	97.9 ± 0.3	97.8 ± 0.3	97.7 ± 0.2	$\mathbf{98.6 \pm 0.1}$

ozone	2534	72	0	2	94 ± 0.3	94.1 ± 0.3	$\mathbf{94.4 \pm 0.3}$	93.9 ± 0.1	94.3 ± 0.2
page_blocks	5473	10	0	5	$\mathbf{97.3 \pm 0.2}$	97.2 ± 0.1	$\mathbf{97.3 \pm 0.2}$	96.9 ± 0.1	$\mathbf{97.3 \pm 0.2}$
parkinsons	195	22	0	2	69 ± 8	$\mathbf{75 \pm 3}$	67 ± 9	67 ± 10	$\mathbf{75 \pm 5}$
pendigits	7494	16	0	10	99.5 ± 0.1	99.1 ± 0.1	99.1 ± 0.1	99.3 ± 0.1	$\mathbf{99.6 \pm 0.1}$
pima	768	8	0	2	$\mathbf{66 \pm 4}$	64 ± 4	63 ± 5	65 ± 2	64 ± 3
pittsburgh_bridges-MATERIAL	106	4	3	3	$\mathbf{55 \pm 5}$	$\mathbf{55 \pm 5}$	51 ± 5	12 ± 5	38 ± 10
pittsburgh_bridges-REL-L	103	4	3	3	58 ± 6	58 ± 7	59 ± 7	52 ± 3	$\mathbf{62 \pm 4}$
pittsburgh_bridges-SPAN	92	4	3	3	40 ± 10	40 ± 10	40 ± 10	40 ± 10	40 ± 10
pittsburgh_bridges-T-OR-D	102	4	3	2	7 ± 7	7 ± 12	$\mathbf{33 \pm 11}$	7 ± 7	27 ± 12
pittsburgh_bridges-TYPE	106	4	3	7	$\mathbf{39 \pm 4}$	37 ± 4	27 ± 8	11 ± 7	24 ± 5
planning	182	12	0	2	60 ± 2	60 ± 4	48 ± 5	$\mathbf{62 \pm 2}$	59 ± 4
post_operative	90	8	0	3	$\mathbf{60 \pm 0}$	50 ± 10	$\mathbf{60 \pm 0}$	50 ± 0	50 ± 10
ringnorm	7400	20	0	2	96.1 ± 0.2	92.1 ± 0.5	$\mathbf{96.3 \pm 0.4}$	95.9 ± 0.1	95.6 ± 0.3
seeds	210	7	0	3	$\mathbf{91 \pm 3}$	90 ± 4	89 ± 4	88 ± 4	90 ± 3
semeion	1593	256	0	10	93.4 ± 0.4	93.7 ± 0.7	93.7 ± 0.5	91 ± 0.9	$\mathbf{94.2 \pm 0.4}$
soybean	307	22	13	19	90 ± 1	90 ± 2	90 ± 2	90 ± 1	$\mathbf{92 \pm 2}$
spambase	4601	57	0	2	92.9 ± 0.7	92.2 ± 0.6	92.6 ± 0.5	90.4 ± 0.5	$\mathbf{93.3 \pm 0.7}$
spect	80	22	0	2	30 ± 10	$\mathbf{40 \pm 10}$	$\mathbf{40 \pm 10}$	$\mathbf{40 \pm 20}$	30 ± 10
spectf	80	44	0	2	$\mathbf{60 \pm 10}$	50 ± 10	40 ± 10	$\mathbf{60 \pm 10}$	50 ± 10
statlog_australian-credit	690	10	4	2	77 ± 2	$\mathbf{78 \pm 2}$	76 ± 3	72 ± 1	74 ± 1
statlog_german-credit	1000	14	6	2	$\mathbf{66.4 \pm 1.5}$	65 ± 1.7	63.6 ± 2.9	61.6 ± 1.7	64.3 ± 1.6
statlog_heart	270	10	3	2	68 ± 1	70 ± 2	$\mathbf{71 \pm 4}$	69 ± 3	67 ± 2
statlog_image	2310	19	0	7	98 ± 0.5	97.7 ± 0.4	98.3 ± 0.4	96.5 ± 0.5	$\mathbf{98.2 \pm 0.4}$
statlog_landsat	4435	36	0	6	88.5 ± 0.5	88.3 ± 0.6	$\mathbf{89.2 \pm 0.5}$	87.8 ± 0.4	88.9 ± 0.6
statlog_shuttle	43500	9	0	7	$\mathbf{99.98 \pm 0.01}$	99.97 ± 0.01	99.97 ± 0.01	99.87 ± 0.01	99.97 ± 0.01
statlog_vehicle	846	18	0	4	74 ± 1	68 ± 2	69 ± 1	69 ± 2	$\mathbf{77 \pm 0}$
steel_plates	1941	27	0	7	68.1 ± 1	69.4 ± 0.6	$\mathbf{71.1 \pm 0.9}$	64.4 ± 1.7	66.4 ± 1.5
synthetic_control	600	60	0	6	98 ± 1	$\mathbf{99 \pm 1}$	98 ± 1	98 ± 1	$\mathbf{99 \pm 0}$
teaching	151	3	2	3	$\mathbf{39 \pm 6}$	36 ± 4	30 ± 3	$\mathbf{39 \pm 8}$	38 ± 8
thyroid	3772	21	0	3	96.8 ± 1	$\mathbf{97.2 \pm 1.4}$	96.5 ± 1.6	38.4 ± 1.8	93.7 ± 1.9
tic_tac-toe	958	0	9	2	96 ± 1	$\mathbf{97 \pm 1}$	$\mathbf{97 \pm 1}$	55 ± 3	95 ± 1
titanic	2201	2	1	2	$\mathbf{69.1 \pm 0.5}$	$\mathbf{69.1 \pm 0.5}$	68.5 ± 0.4	68.5 ± 0.4	68.5 ± 0.4

twonorm	7400	20	0	2	95.5 ± 0.3	94.8 ± 0.2	94.9 ± 0.3	$\mathbf{95.7 \pm 0.3}$	$\mathbf{95.7 \pm 0.3}$
vertebral_column_task_1	310	6	0	2	$\mathbf{78 \pm 2}$	75 ± 2	72 ± 1	$\mathbf{78 \pm 2}$	75 ± 1
vertebral_column_task_2	310	6	0	3	$\mathbf{68 \pm 5}$	66 ± 5	66 ± 4	63 ± 4	67 ± 1
wall_following	5456	24	0	4	99.3 ± 0.2	99.3 ± 0.2	$\mathbf{99.6 \pm 0.1}$	83.4 ± 0.9	96.3 ± 0.3
waveform	5000	21	0	3	$\mathbf{79.5 \pm 0.6}$	77.9 ± 0.7	79.2 ± 0.4	$\mathbf{79.5 \pm 0.4}$	79 ± 0.5
waveform_noise	5000	40	0	3	79.9 ± 0.5	78.9 ± 0.5	79.2 ± 0.7	79 ± 0.4	$\mathbf{80.3 \pm 0.6}$
wine	178	13	0	3	95 ± 3	94 ± 4	$\mathbf{97 \pm 2}$	96 ± 2	$\mathbf{97 \pm 2}$
wine_quality-red	1599	11	0	6	$\mathbf{47.3 \pm 3}$	46.2 ± 3.4	45.2 ± 2.9	47.1 ± 2.6	46.6 ± 2.7
wine_quality-white	4898	11	0	7	$\mathbf{43.8 \pm 2.2}$	42.8 ± 1.6	42 ± 1.2	43.4 ± 2.2	43.7 ± 1.9
yeast	1484	8	0	10	47.7 ± 1.9	$\mathbf{48 \pm 2.5}$	47.2 ± 2.4	46.5 ± 1.9	46.3 ± 2.4
zoo	101	16	0	7	93 ± 4	$\mathbf{94 \pm 3}$	93 ± 2	$\mathbf{94 \pm 3}$	$\mathbf{94 \pm 3}$

Table 1: Five-fold cross-validation Cohen's kappa values (mean±SEM) on the UCI datasets, along with summary statistics for each dataset. n is the number of examples, p_{num} is the number of numeric features, p_{cat} is the number of categorical features, and C is the number of classes. Best performing algorithm for each data set is highlighted in bold text.

Appendix E. Strength and Correlation of Trees

One of the most important and well-known results in ensemble learning theory for classification states that the generalization error of an ensemble learning procedure is bounded above by the quantity $\bar{\rho}(1-s^2)/s^2$, where $\bar{\rho}$ is a particular measure of the correlation of the base learners and s is a particular measure of the strength of the base learners (Breiman, 2001). In both `SPORF` and `F-RC`, the set of possible splits that can be sampled is far larger in size than that for `RF`, which may lead to more diverse trees. Moreover, the ability to sample a more diverse set of splits may increase the likelihood of finding good splits and therefore boost the strength of the trees. To investigate the strength and correlation of trees using different projection distributions, we evaluate `RF`, `F-RC`, and `SPORF` on the three simulation settings described above. Scatter plots of tree strength vs tree correlation are shown in Figure 11 for sparse parity ($n = 1000$), orthant ($n = 400$), Trunk ($n = 10$), and Trunk ($n = 100$). In all four settings, `SPORF` classifies as well as or better than `RF` and `F-RC`.

On the sparse parity setting, `SPORF` and `F-RC` produce significantly stronger trees than does `RF`, at the expense of an increase in correlation among the trees (Figure 11A). Both `SPORF` and `F-RC` are much more accurate than `RF` in this setting, so any performance degradation due to the increase in correlation relative to `RF` is outweighed by the increased strength. `SPORF` produces slightly less correlated trees than does `F-RC`, which may explain why `SPORF` has a slightly lower error rate than does `F-RC` on this setting.

On the orthant setting, `F-RC` produces trees of roughly the same strength as those in `RF`, but significantly more correlated (Figure 11B). This may explain why `F-RC` has substantially worse prediction accuracy than does `RF`. `SPORF` also produces trees more correlated than those in `RF`, but to a lesser extent than `F-RC`. Furthermore, the trees in `SPORF` are stronger than those in `RF`. Observing that `SPORF` has roughly the same error rate as `RF` does, it seems that any contribution of greater tree strength in `SPORF` is canceled by a contribution of greater tree correlation.

On the Trunk setting with $p = 10$ and $n = 10$, `SPORF` and `F-RC` produces trees that are comparable in strength to those in `RF` but less correlated (Figure 11C). However, when increasing n to 100, the trees in `SPORF` and `F-RC` become both stronger and more correlated. In both cases, `SPORF` and `F-RC` have better classification performance than `RF`.

These results suggest a possibly general phenomenon. Namely, for smaller training set sizes, tree correlation may be a more important factor than tree strength because there is not enough data to induce strong trees, and thus, the only way to improve performance is through increasing the diversity of trees. Likewise, when the training set is sufficiently large, tree correlation matters less because there is enough data to induce strong trees. Since `SPORF` has the ability to produce both stronger and more diverse trees than `RF`, it is adaptive to both regimes In all four settings, `SPORF` never produces more correlated trees than does `F-RC`, and sometimes produces less correlated trees. A possible explanation for this is that the splits made by `SPORF` are linear combinations of a random number of dimensions, whereas in `F-RC` the splits are linear combinations of a fixed number of dimensions. Thus, in some sense, there is more randomness in `SPORF` than in `F-RC`.

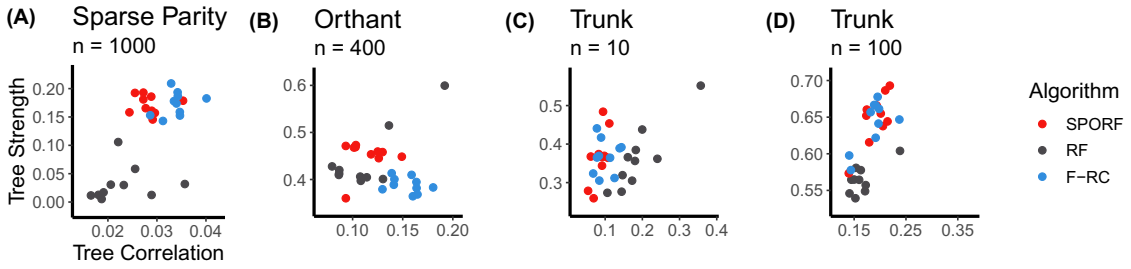

Figure 11: Comparison of tree strength and correlation of SPORF, RF, and F-RC on four of the simulated data sets. (A): sparse parity with $p = 10, n = 1000$, (B): orthant with $p = 6, n = 400$, (C): Trunk with $p = 10, n = 10$, and (D): Trunk with $p = 10, n = 100$. For a particular algorithm, there are ten dots, each corresponding to one of ten trials. Note in all settings, SPORF beats RF and/or F-RC. However, the mechanism by which it does varies across the different settings. In sparse parity SPORF wins because the trees are substantially stronger, even though the correlation increases. In Trunk for small sample size, it is purely because of less correlated trees. However, when sample size increases 10-fold, it wins purely because of stronger trees. This suggests that SPORF can effectively trade-off strength for correlation on the basis of sample complexity to empirically outperform RF and F-RC.

Appendix F. Understanding the Bias and Variance of SPORF

The crux of supervised learning tasks is to optimize the trade-off between bias and variance. As a first step in understanding how the choice of projection distribution effects the balance between bias and variance, we estimate bias, variance, and error rate of the various algorithms on the sparse parity problem. Universally agreed upon definitions of bias and variance for 0-1 loss do not exist, and several such definitions have been proposed for each. Here we adopt the framework for defining bias and variance for 0-1 loss proposed by James (2003). Under this framework, bias and variance for 0-1 loss have similar interpretations to those for mean squared error. That is, bias is a measure of the distance between the expected output of a classifier and the true output, and variance is a measure of the average deviation of a classifier output around its expected output. Unfortunately, these definitions (along with the term for Bayes error) do not provide an additive decomposition for the expected 0-1 loss. Therefore, James (2003) provides two additional statistics that do provide an additive decomposition. In this decomposition, the so-called "systematic effect" measures the contribution of bias to the error rate, while the "variance effect" measures the contribution of variance to the error rate. For completeness, we restate these definitions below.

Let $\bar{h}(X) = \underset{k}{\operatorname{argmax}} P_{\mathcal{D}_n}(h(X|\mathcal{D}_n) = k)$ be the most common prediction (mode) with respect to the distribution of $\mathcal{D}_n$. This is referred to as the "systematic" prediction by James (2003). Furthermore, let $P^*(X) = P_{Y|X}(Y = h^*(X)|X)$ and $\bar{P}(X) = P_{\mathcal{D}_n}(h(X|\mathcal{D}_n) =$

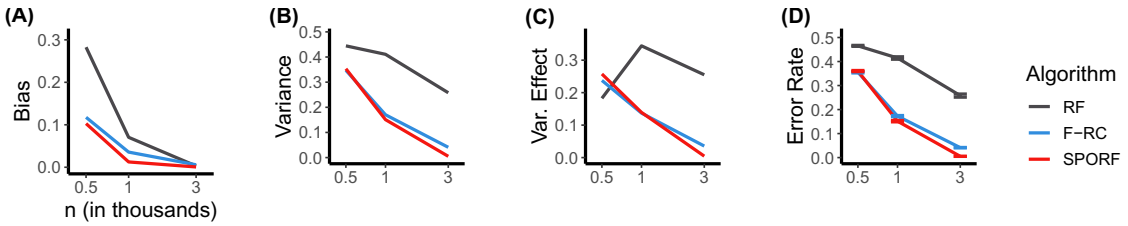

Figure 12: (A-D): Bias, variance, variance effect, and error rate, respectively, on the sparse parity problem as a function of the number of training samples. Error rate is the sum of systematic effect and variance effect, which roughly measure the contributions of bias and variance to the error rate, respectively. In this example, bias and systematic effect are identical because the Bayes error is zero (refer to James, 2003). For smaller training sets, `SPORF` wins primarily through lower bias/systematic effect, while for larger training sets it wins primarily through lower variance effect.

$\bar{h}(X)$). The bias, variance, systematic effect (SE), and variance effect (VE) are defined as

$$\begin{aligned} Bias &= P_X(\bar{h}(X) = h^*(X)), \\ Var &= 1 - E_X[\bar{P}(X)], \\ SE &= E_X[P^*(X) - P_{Y|X}(Y = \bar{h}(X)|X)], \\ VE &= E_X[P_{Y|X}(Y = \bar{h}(X)|X) \\ &\quad - \sum_k P_{Y|X}(Y = k|X) P_{\mathcal{D}_n}(h(X|\mathcal{D}_n) = k)]. \end{aligned}$$

Figure 12 compares estimates of bias, variance, variance effect, and error rate for `SPORF`, `RF`, and `F-RC` as a function of number of training samples. Since the Bayes error is zero in these settings, systematic effect is the same as bias. The four metrics are estimated from 100 repeated experiments for each value of n. In Figure 12A, `SPORF` has lower bias than both `RF` and `F-RC` for all training set sizes. All algorithms converge to approximately zero bias after about 3000 samples. Figure 12B shows that `RF` has substantially more variance than do `SPORF` and `F-RC`, and `SPORF` has slightly less variance than `F-RC` at 3,000 samples. The trend in Figure 12C is similar to that in Figure 12B, which is not too surprising since VE measures the contribution of the variance to the error rate. Interestingly, although `RF` has noticeably more variance at 500 samples than do `SPORF` and `F-RC`, it has slightly lower VE. It is also surprising that the VE of `RF` increases from 500 to 1000 training samples. It could be that this is the result of the tradeoff of the substantial reduction in bias. In Figure 12D, the error rate is shown for reference, which is the sum of bias and VE. Overall, these results suggest that `SPORF` wins on the sparse parity problem with a small sample size primarily through lower bias/SE, while with a larger sample size it wins mainly via lower variance/VE. A similar trend holds for the orthant problem (not shown).

References

Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003.

Gérard Biau, Luc Devroye, and Gábor Lugosi. Consistency of random forests and other averaging classifiers. *The Journal of Machine Learning Research*, 9:2015–2033, 2008.

Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *International Conference on Knowledge Discovery and Data Mining*, pages 245–250, 2001.

Rico Blaser and Piotr Fryzlewicz. Random rotation ensembles. *Journal of Machine Learning Research*, 17(4):1–26, 2016.

Leo Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

Leo Breiman. Random forests. *Machine Learning*, 4(1):5–32, October 2001.

Leo Breiman and Adele Cutler. *Random forests*, 2002. URL `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`.

James Browne, Tyler Tomita, and Joshua Vogelstein. *rerf: Randomer forest*, 2018. URL `https://cran.r-project.org/web/packages/rerf/`.

James Browne, Disa Mhembere, Tyler Tomita, Joshua T. Vogelstein, and Randal Burns. Forest packing: fast parallel, decision forests. In *SIAM International Conference on Data Mining*, pages 46–54. Society for Industrial and Applied Mathematics, 2019.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *International Conference on Machine Learning*, pages 161–168, 2006.

Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *International Conference on Machine learning*, pages 96–103, 2008.

Tianqi Chen. *xgboost: Extreme gradient boosting*, 2018. URL `https://cran.r-project.org/web/packages/xgboost/`.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *ACM Symposium on Theory of Computing*, pages 537–546, New York, NY, USA, 2008. ACM.

Sanjoy Dasgupta and Yoav Freund. Random projection trees for vector quantization. *IEEE Transactions on Information Theory*, 55(7), 2009.

Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for exact nearest neighbor search. *Conference on Learning Theory*, 2013.

Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer Science & Business Media, 1996.

Dirk Eddelbuettel. *Rcpp: seamless R and C++ integration*, 2018. URL `https://cran.r-project.org/web/packages/Rcpp/index.html`.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 2018.

Xiaoli Z. Fern and Carla E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *International Conference on Machine Learning*, pages 186–193, 2003.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *International Conference on Knowledge Discovery and Data Mining*, pages 517–522, 2003.

Jerome H. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, pages 1189–1232, 2001.

David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *International Joint Conferences on Artificial Intelligence*, pages 1002–1007, 1993.

Chinmay Hegde, Michael Wakin, and Richard Baraniuk. Random projections for manifold learning. In *Advances in Neural Information Processing Systems*, pages 641–648, 2008.

Gareth M. James. Variance and bias for general loss functions. *Machine Learning*, 51(2): 115–135, 2003.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 3146–3154. Curran Associates, Inc., 2017.

Yann Lecun, Corinna Cortes, and Christopher J. C. Burges. *The MNIST database of handwritten digits*. URL `http://yann.lecun.com/exdb/mnist/`.

Donghoon Lee, Ming-Hsuan Yang, and Songhwai Oh. Fast and accurate head pose estimation via random projection forests. In *IEEE International Conference on Computer Vision*, pages 1958–1966, 2015.

Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *International Conference on Knowledge Discovery and Data Mining*, pages 287–296, 2006.

Gilles Louppe. *Understanding Random Forests: From Theory to Practice*. PhD thesis, University of Liège, 2014.

Bjoern H. Menze, B. Michael Kelm, Daniel N. Splitthoff, Ullrich Koethe, and Fred A. Hamprecht. On oblique random forests. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6912 of *Lecture Notes in Computer Science*, pages 453–469. Springer, Berlin, Heidelberg, 2011.

Alessandro Motta, Meike Schurr, Benedikt Staffler, and Moritz Helmstaedter. Big data in nanoscale connectomics, and the greed for training labels. *Current Opinion in Neurobiology*, 55:180–187, 2019.

Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.

Tom Rainforth and Frank Wood. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*, 2015.

Juan J. Rodriguez, Ludmila I. Kuncheva, and Carlos J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, 2006.

Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. Consistency of random forests. *The Annals of Statistics*, 43(4):1716–1741, 2015.

Tyler M. Tomita, Mauro Maggioni, and Joshua T. Vogelstein. Roflmao: Robust oblique forests with linear matrix operations. In *SIAM International Conference on Data Mining*, 2017.

Gerard V. Trunk. A problem of dimensionality: A simple example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(3):306–307, 1979.

Roman Vershynin. *High Dimensional Probability: An Introduction with Applications in Data Science*. 2019. URL `https://www.math.uci.edu/~rvershyn/papers/HDP-book/HDP-book.pdf`.

Marvin N. Wright. *ranger: A fast implementation of random forests*, 2018. URL `https://cran.r-project.org/web/packages/ranger/`.

Abraham J. Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *Journal of Machine Learning Research*, 18(1), 2017.