

Sparse Projection Oblique Randomer Forests

Tyler M. Tomita¹, James Browne³, Cencheng Shen⁴, Jaewon Chung², Jesse L. Patsolic², Benjamin Falk², Jason Yim⁵, Carey E. Priebe⁶, Randal Burns³, Mauro Maggioni⁶, and Joshua T. Vogelstein^{2,3,7}

¹Department of Psychological and Brain Sciences, Johns Hopkins University, Baltimore, MD

²Center for Imaging Science, Johns Hopkins University, Baltimore, MD

³Department of Computer Science, Johns Hopkins University, Baltimore, MD

⁴Department of Applied Economics and Statistics, University of Delaware, Newark, DE

⁵DeepMind, London, UK

⁶Department of Mathematics, Johns Hopkins University, Baltimore, MD

⁷Institute for Computational Medicine, Kavli Neuroscience Discovery Institute, Department of Biomedical Engineering, Johns Hopkins University, Baltimore, MD

Decision forests, including Random Forests and Gradient Boosting Trees, have recently demonstrated superior 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 axis-aligned splits, such as 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 on a standard benchmark suite for classification with > 100 problems of varying dimension, sample size, and number of classes. To illustrate how, why, and when SPORF outperforms other methods, we conduct extensive simulated experiments. SPORF typically yields improved performance over existing decision forests, while mitigating computational efficiency and scalability and maintaining interpretability. SPORF can easily be incorporated into other ensemble methods such as boosting to obtain potentially similar gains.

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 [1–3]. 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 [4]. GBTs have been a key component of many Kaggle competition-winning solutions, and was part of the Netflix Prize winning solution [4].

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 trees require very deep trees with complicated 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 [5]. 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) [6], and the Canonical Correlation Forest (CCF) [7]. 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 inputs or noise, 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 [8], rather than axis-aligned splits. Very sparse random projections preserve many of the desirable properties of axis-aligned decision trees, while mitigating their issues.

Our statistically- and computationally-efficient parallelized implementations are available from <https://neurodata.io/rerf/> 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/2.0.2/>), and is sklearn API compliant.

2 Background & Related Work

2.1 Statistical Learning: Classification

Here we briefly describe the classification problem. Let $X \in \mathbb{R}^p$ be a random real-valued vector and $Y \in \mathcal{Y} = \{c_1, \dots, c_K\}$ be a random variable denoting a class label associated with X . Suppose the pair (X, Y) has joint distribution f_{XY} . Let the lower-case counterpart (x, y) denote a particular realization of the random variable pair (X, Y) . Assume n pairs of (x_i, y_i) are sampled identically and independently (i.i.d.) from f_{XY} . Denote an arbitrary training set of n i.i.d samples by $\mathcal{D}_n = \{X_i, Y_i\}_1^n$; the goal is to learn a classifier $h(\cdot; \mathcal{D}_n) : \mathbb{R}^p \rightarrow \mathcal{Y}$ that correctly predicts the unobserved class label y associated with an observed x . Specifically, we would like to minimize the probability of misclassification $P_{XY}(h(X; \mathcal{D}_n) \neq Y)$. It is well-known that the Bayes classifier $h^*(X) = \underset{k}{\operatorname{argmax}} P_{Y|X}(Y = k|X)$ minimizes the probability of misclassification. Generally in real-world settings, $h^*(X)$ is unknown because $P_{Y|X}$ is unknown.

2.2 Random Forests

The original random forest (RF) procedure popularized by Leo Breiman is one of the most commonly employed classification learning algorithms [5]. 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 partitions. 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 $I(S) = \sum_{k=1}^K f_k(1 - f_k)$, where $f_k = \frac{1}{|S|} \sum_{i \in S} \mathbb{I}[y_i = c_k]$.

More concretely, let $\theta = (j, \tau)$, where j is an index selecting a dimension and τ is a splitting threshold. Furthermore, let $S_\theta^L = \{i : x_i^{(j)} \leq \tau, \forall i \in S\}$ and $S_\theta^R = \{i : x_i^{(j)} > \tau, \forall i \in S\}$ be the subsets of S to the left and right of the splitting threshold, respectively. Here, $x_i^{(j)}$ denotes the value of the j th feature

for the i th observation. Let n_S , n_L , and n_R denote the number of points in the parent, left, and right child nodes, respectively. 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 carried out by exhaustively searching 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 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. Let $m(x)$ be the index of the leaf node that x falls into. Then the tree prediction is $h(l_{m(x)})$. Let $\hat{y}^{(t)}$ 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:

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

Breiman [5] 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 [5].

2.3 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 [9]. Various approaches have been proposed for constructing oblique forests. Breiman [5] 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

[5] and Tomita et al. [10] indicate that F-RC tends to empirically outperform RF on a wide variety of datasets. Heath et al. [11] 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. [12] attempted to find discriminative split directions via PCA. Menze et al. [13] perform supervised learning of linear discriminative models at each node. Blaser and Fryzlewicz [6] 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 [7]’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. [14]’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.

2.4 Random Projections

Given a data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, one can construct a random projection matrix $\mathbf{A} \in \mathbb{R}^{p \times d}$ whose entries a_{ij} are i.i.d. with zero mean and constant variance and multiply it by \mathbf{X} to obtain:

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

With high probability, the much smaller matrix $\tilde{\mathbf{X}}$ preserves all pairwise distances of \mathbf{X} with small distortion.

Due to theoretical guarantees, random projections are commonly employed as a dimensionality reduction tool in machine learning applications [15–19]. Different probability distributions over the entries lead to different average errors and error tail bounds. Li et al. [8] 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:

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

Dasgupta and Freund [20] 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 [21] and nearest neighbor search [22]. Our work is inspired by this work, but in a supervised setting.

2.5 Gradient Boosted Trees

Gradient boosted trees (GBTs) are another tree ensemble method commonly used for regression and classification tasks. Unlike in RF, GBTs are learned in an iterative stage-wise manner by directly minimizing a cost function via gradient descent [23, 24]. Despite the obvious differences in the learning procedures between GBT and RF, they tend to perform comparably. A study by Wyner et al. [25] 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 [4] and LIGHTGBM

[26], 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.

2.6 Desirable Properties for Ensembles of Trees

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} from distribution $f_{\mathbf{A}}$, where $\mathbf{A} \in \mathbb{R}^{p \times d}$, possibly in a data-dependent or supervised fashion. This matrix is used to randomly project the feature matrix, yielding $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{A} \in \mathbb{R}^{n \times d}$, 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. See Algorithms 1 and 2 for details. 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 in this projected subspace amounts to searching over a random subset of the original features.

While the best specification of a distribution over random projections (if one exists) is dataset-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 reduce tree diversity, causing the model to overfit noise (we demonstrate this in high-dimensional settings in Section 5.3). This is akin to 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 [27] 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. [10] 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 [28]. 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 [29] 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.

3 Methods

3.1 Sparse Projection Oblique Randomer Forests

Sparse Projection Oblique Randomer Forests (SPORF) searches for splits over sparse random projections [8]. 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 $\lceil \lambda p d \rceil$ 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.¹ These nonzeros are then distributed uniformly at random in \mathbf{A} . See Algorithm 1 for details on how to grow a decision tree using any arbitrary f_A , which includes SPORF as a special case.

3.2 Training and Hyperparameter Tuning

Unless otherwise stated, model training and tuning for all algorithms except for XGBOOST and CCF is performed in the following way. Each algorithm uses 500 trees, which was empirically determined to be sufficient for convergence of out-of-bag error in 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). Two hyperparameters are tuned via minimization of out-of-bag error. The first parameter 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 $\{1/p, \dots, 5/p\}$. 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). Projection bootstrapping is used rather than bagging [7]. The only hyperparameter tuned is the number of features subsampled prior to performing CCA. We optimize this hyperparameter over the set $\{p^{1/4}, p^{1/2}, p^{3/4}, p\}$. The projection bootstrapping procedure requires each tree to be trained on the full dataset. 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.

Hyperparameters in XGBOOST are tuned via grid search using the R caret package (see Appendix A 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.

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

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.

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. [30] 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 1(A) shows a schematic illustration (because we cannot show countably infinite rows or columns). On this problem, Biau et al. [30] 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 1(B) shows that SPORF, RR-RF, and CCF approach perfect classification. This is because, although it is also greedy, it 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 statistical 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 across on a wider set of classification problems. It also highlights why, from a theoretical standpoint, oblique forests are advantageous.

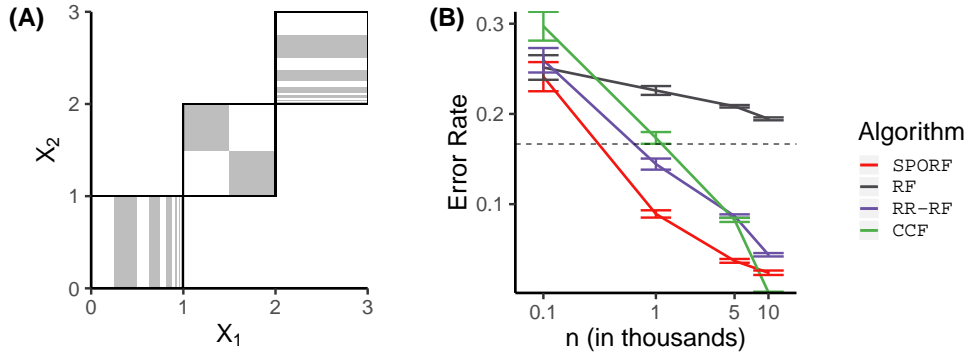


Figure 1: Classification performance on the consistency ($p = 2$) problem as a function of the number of training samples. The consistency problem is designed such that RF has a theoretical lower bound of error of $1/6$. (A) The joint distribution of (X, Y) . X is uniformly distributed in the three unit 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 datasets

In the sections that follow, we perform a variety of experiments on three carefully constructed simulated classification problems, referred 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 subsets of dimensions contains any information. Specifically, let $X = (X_1, \dots, X_p)$ be a p -dimensional feature vector, where each $X_1, \dots, X_p \stackrel{iid}{\sim} U(-1, 1)$. Furthermore, let $Q = \sum_{j=1}^{p^*} \mathbb{I}(X_j > 0)$, where $p^* < p$ and $\mathbb{I}(X_j > 0)$ is the indicator that the j^{th} feature has a value greater than zero. A sample's class label Y is equal to the parity of Q . That is, $Y = \text{odd}(Q)$, 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 2 (A,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 datapoint 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 $X_1, \dots, X_p \stackrel{iid}{\sim} U(-1, 1)$. 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 2 (D,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.

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 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.

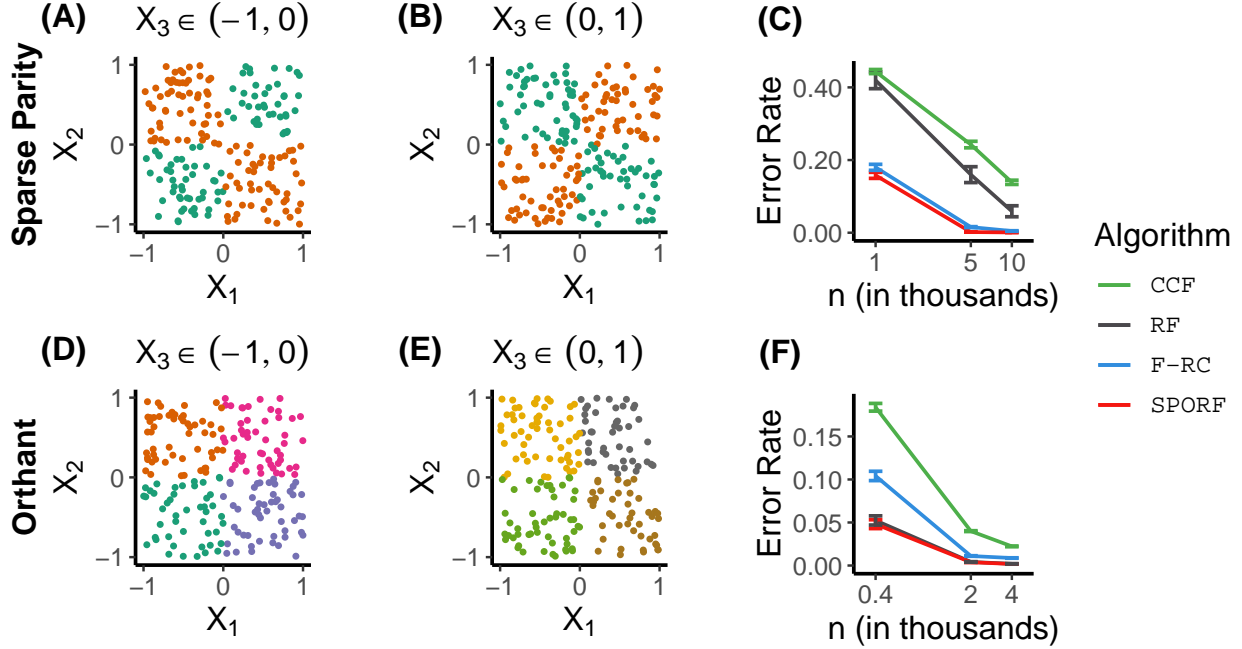


Figure 2: Classification performance on the sparse parity ($p_s = 20$) and orthant ($p_o = 6$) problems for various numbers of training samples. In both case, we sample $X_1, \dots, X_p \stackrel{iid}{\sim} U(-1, 1)$. F-RC has been known to perform much better than RF on the sparse parity problem [10]. 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 (i.e., individual linear combinations). Breiman denoted this hyperparameter as 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 datasets described previously. For SPORF, λ is defined as in Section 3.1. For F-RC, we note that $\lambda = L/p$ (i.e. density of a univariate projection is the number of features to combine, divided by the total number of features). For each of $\lambda \in \{\frac{2}{p}, \dots, \frac{5}{p}\}$, 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.

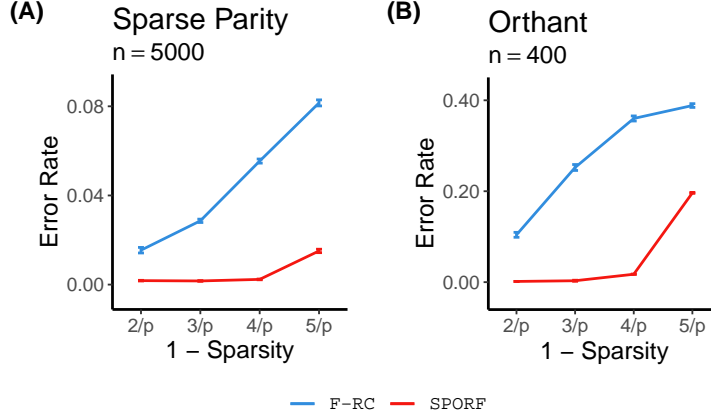


Figure 3: Dependence of error rate on the hyperparameter λ , which controls the average density (1 - sparsity) of projections for two different simulation settings. **(A)** Error rate as a function of λ on sparse parity ($n = 5000$, $p = 20$). **(B)** The same as (A) except on orthant ($n = 400$, $p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

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 of Gini importance (hereafter called Gini importance) [9]. 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.

We evaluate feature importance on a simulation we call *Trunk*. Trunk is a balanced, two-class problem in which each class is distributed as a p -dimensional multivariate Gaussian with identity covariance matrices [31]. Every dimension is informative, but each subsequent dimension is less informative than the last. The class 1 mean is $\mu_1 = (1, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{3}}, \dots, \frac{1}{\sqrt{p}})$, and $\mu_0 = -\mu_1$. The Bayes optimal decision boundary is the hyperplane $(\mu_1 - \mu_0)^T X = 0$. We set $p = 10$ in the following experiments. Gini importance is computed for each feature on both RF and SPORF on the Trunk problem with $n = 1,000$. Methods such as F-RC, RR-RF and CCF 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 CCFsplit on dense univariate projections, which tend to be uninterpretable.

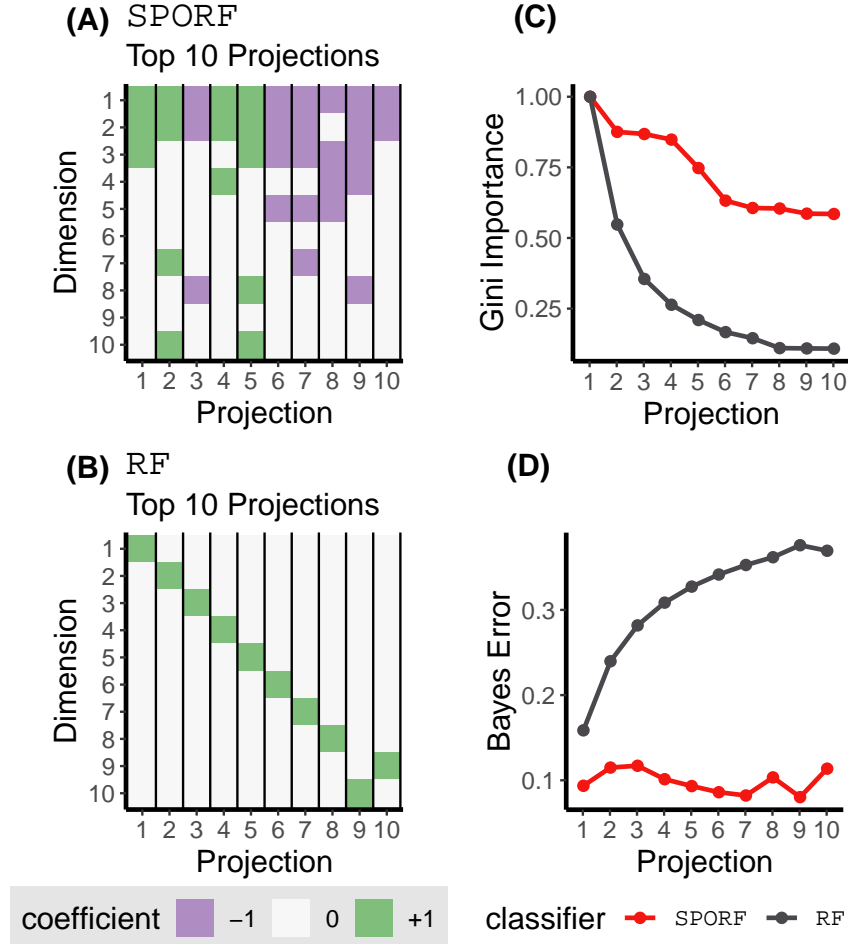


Figure 4: The ten projections with the highest Gini importance found by RF and SPORF on the Trunk problem with $p = 10, n = 1000$. **(A)** Visual representation of the top 10 projections identified by SPORF. The x-axis indicates the projection. The y-axis indicates the index of the ten canonical dimensions. 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.

Figure 4 depicts the features that define each of the top ten split node projections for SPORF (A) and RF (B). 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 4(C) 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 4(D) 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

5.1 SPORF Exhibits Best Overall Classification Performance on a Large Suite of Benchmark Datasets

SPORF compares favorably to RF, XGBOOST, RR-RF, and CCF on a suite of 105 benchmark datasets 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 [1] (16 were excluded for various reasons such as lack of availability; see Appendix B for preprocessing details).

Figure 5(A) shows pairwise comparisons of RF with SPORF (red), XGBOOST (yellow), RR-RF (purple), and CCF (green) on the UCI datasets. 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 dataset via five-fold cross-validation. Error rates for each data set are reported in Appendix F. Let $\Delta(\mathcal{A}) = \kappa(\text{RF}) - \kappa(\mathcal{A})$ be the difference between κ for some algorithm \mathcal{A} —either SPORF, XGBOOST, RR-RF, or CCF—with $\kappa(\text{RF})$. Each beeswarm plot in 5(A) represents the distribution of $\Delta(\mathcal{A})$ over data sets. Comparisons are shown for the 65 numeric datasets (top), the 40 datasets having at least one categorical feature (middle), and all 105 datasets (bottom). A positive value on the x-axis indicates that RF performed better than the algorithm it is being compared to on a particular dataset, 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 squared 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 datasets, due in particular to its relative performance on the numeric datasets. XGBOOST and CCF also tend to outperform RF on the numeric datasets, but unlike SPORF they perform worse than RF on the categorical datasets.

Additionally, we examined how frequently each algorithm ranked in terms of Cohen's kappa across the datasets. A rank of one indicates first place (best) on a particular dataset and a rank of five indicates last place (worst). Histograms (in fraction of data sets) of the relative ranks are shown in Figure 5(B). 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. Wilcoxon signed-rank tests were performed to determine whether SPORF performed significantly better than each of the other algorithms. We found that p-values were < 0.005 for all datasets and just the numeric ones against each of the compared algorithms.

Interestingly, XGBOOST and CCF tend to perform better than RF on the numeric datasets, but worse on the categorical datasets. Categorical datasets had to undergo more processing due to one-hot encoding of the categorical features. Also, prior to preprocessing, the categorical datasets tended to have relatively more missing data than the other datasets. It is possible that RF and SPORF are more

robust to this heavier processing of the categorical datasets.

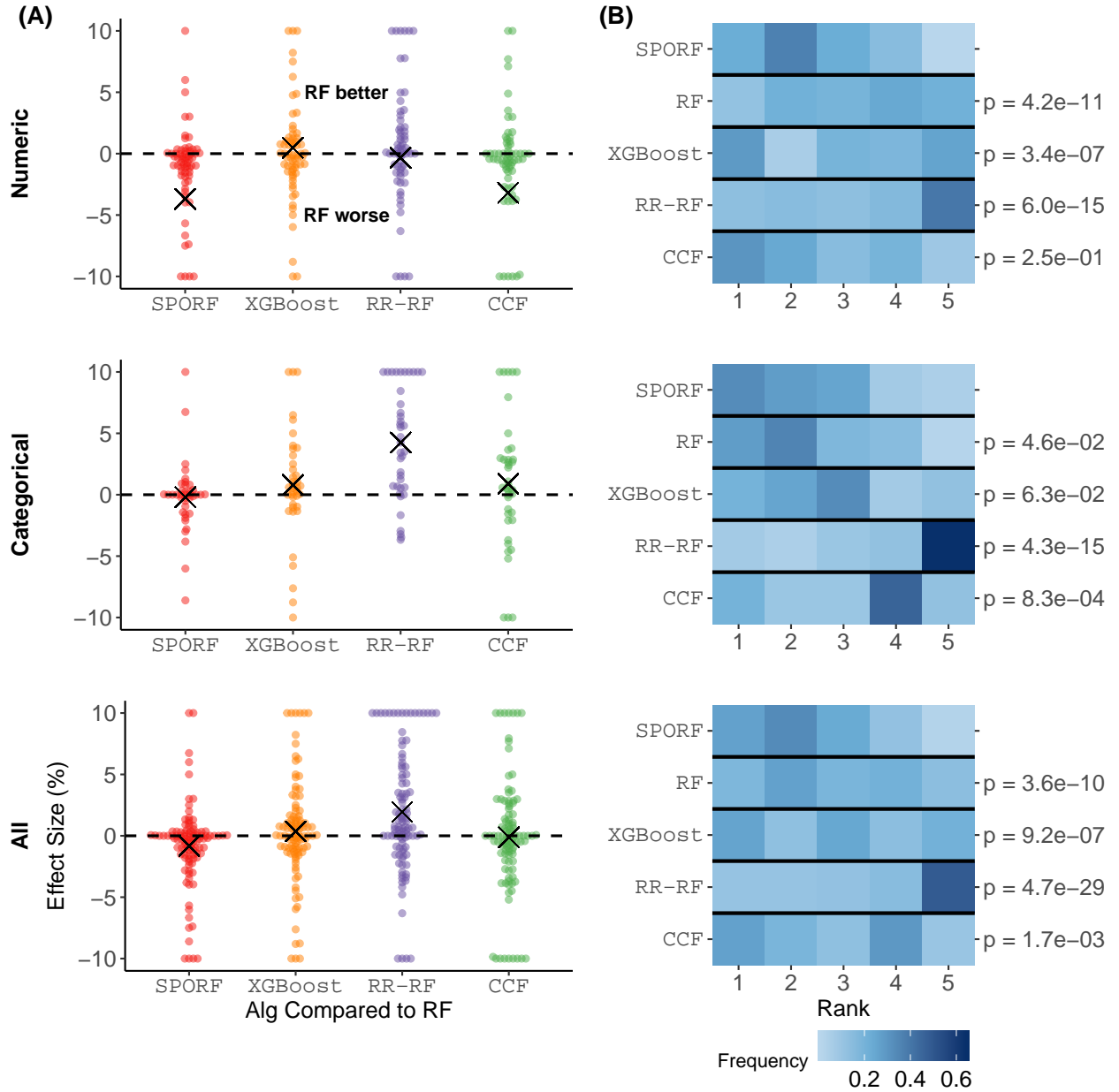


Figure 5: Pairwise comparisons of RF with SPORF, XGBoost, RR-RF, and CCF on the numeric datasets (top), categorical datasets (middle), and all datasets (numeric and categorical combined; bottom) from the UCI Machine Learning Repository (105 datasets total). **(A)** Beeswarm plots showing the distributions of classification performance relative to RF for various decision forest algorithms. Classification performance is measured by effect size, which is defined as $\kappa(\text{RF}) - \kappa(\mathcal{A})$, 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.

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 having limited time and computational budget. Therefore, we sought suitable default values for λ and d based on classification performance on the UCI datasets. For each dataset, for each fold the hyperparameter settings are ranked based on Cohen's kappa computed on the held out set. A rank of n indicates n^{th} place (i.e. first place indicates largest kappa). 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 datasets. 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 datasets.

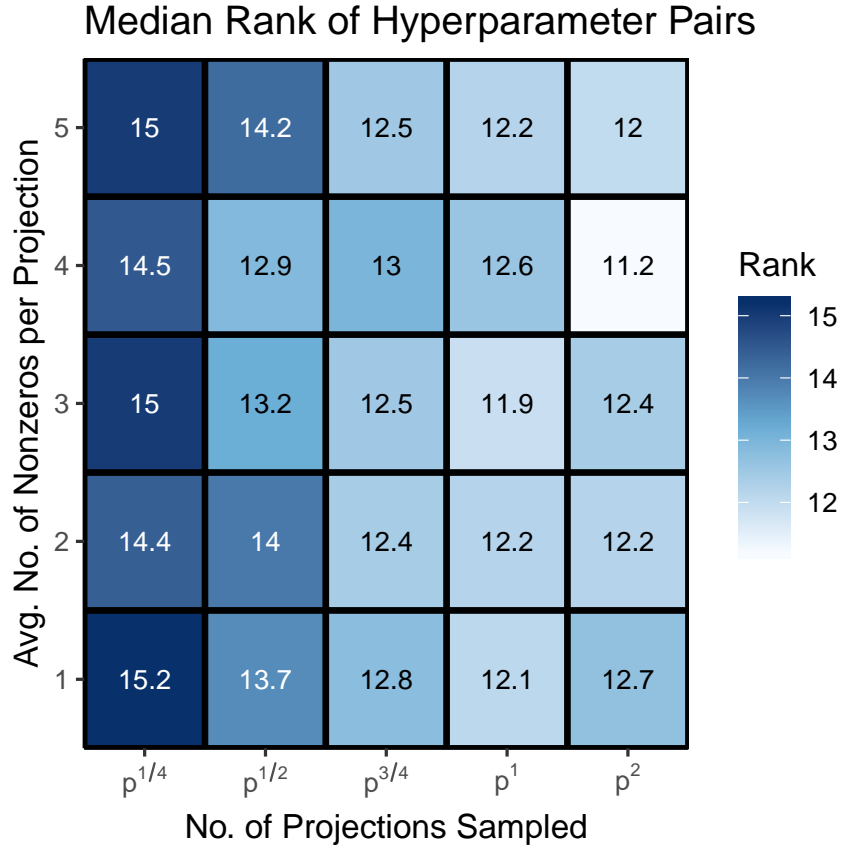


Figure 6: Median rank of SPORF's (d, λ) hyperparameter pairs on the UCI classification datasets (lower is better). Although $(p^2, 4/p)$ is the best performance-wise, we select $(p, 3/p)$ as the default because of a good balance between accuracy and training time.

5.3 SPORF is Robust to High-Dimensional Noise

Next, we investigated the effect of adding a varying number of noise dimensions to the UCI benchmark datasets. For each of the 105 UCI datasets used in the previous experiment, 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 datasets. For all values of D_{noise} , SPORF ties for best classification performance. Notably, CCF performs about as well as SPORF when there is little or no 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 out-of-sample performance, 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.

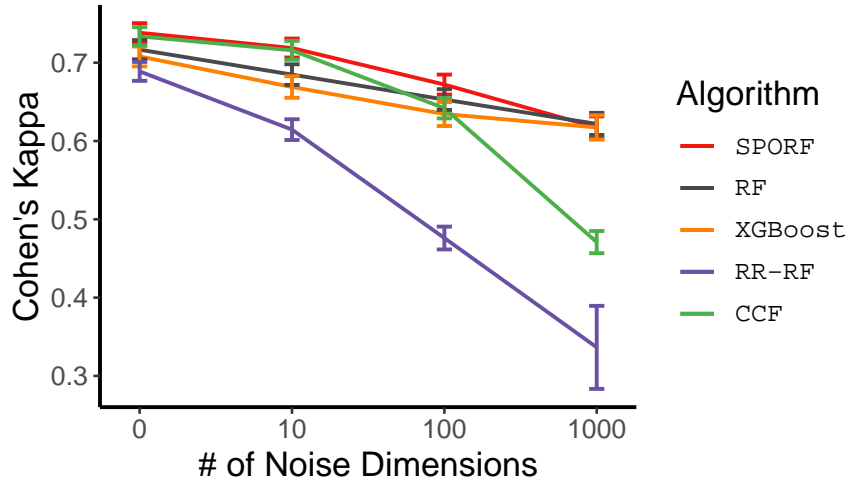


Figure 7: Comparison of classification performance on the UCI benchmark datasets with a varying number of Gaussian noise dimensions added. The x-axis represents the number of noise dimensions added. The y-axis represents the average of Cohen's kappa value over all datasets (\pm SEM). SPORF is always tied for the best performance. CCF and RR-RF are more sensitive to additional 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 $\mathcal{O}(Tdn \log^2 n)$ [32]. 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 $\mathcal{O}(Tdn \log^2 n + Tdn p \lambda)$, 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

6.5 Implementation Details

We use our own R implementation for evaluations of RF, SPORF, F-RC, and RR-RF [33]. 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 [34]. 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 [35]. CCF is evaluated using the authors' openly available MATLAB implementation [7].

6.5.1 Comparison of Algorithms Using the Same Implementation

Figure 8(A) 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 8(B) 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 8(A). Figure 8(C) 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.

6.5.2 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 [33]. We compare speed of training and strong scaling of our implementation to those of the R Ranger [36] and XGBOOST [35] packages, which are currently two of the fastest, parallelized decision tree ensemble software packages available. 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

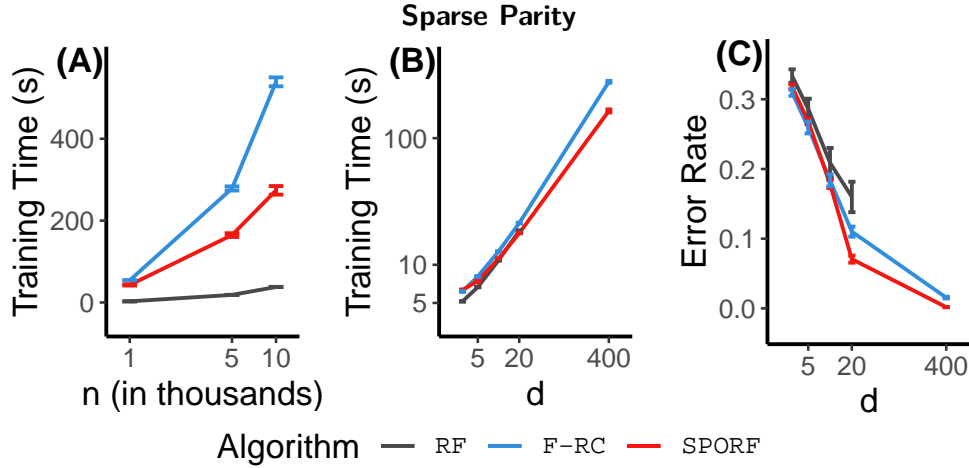


Figure 8: Comparison of training times of RF, SPORF, and F-RC on the 20-dimensional sparse parity setting. **(A)** Dependency of training time using the best set of hyperparameters (y-axis) on the number of training samples (x-axis) for the sparse parity problem. **(B)** Dependency of training time (y-axis) 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.

available large datasets:

MNIST The MNIST dataset [37] 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 9(A)). However, when 48 cores are used, SPORF is as fast as Ranger and still faster than XGBOOST.

Higgs The Higgs dataset (<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 9(B)).

p53 The p53 dataset (<https://archive.ics.uci.edu/ml/datasets/p53+Mutants>) has 31,159 training observations and 5,409 features. Figure 9(C) shows a similar trend as for MNIST. For this dataset, 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 for this MNIST (Figure 9(D)) and Higgs (Figure 9(E)) Figure 9(F) indicates that SPORF has strong scaling in between that of Ranger and XGBOOST. This is due to the simplicity of the p53 dataset, 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 datasets that require classification of each voxel [38]. Moreover, recent automatic hyperparameter tuning suites incorporating runtime in their evaluations, which leverage from out-of-sample prediction accuracy [39]. Thus, accelerating prediction times can improve the effectiveness of hyperparameter sweeps.

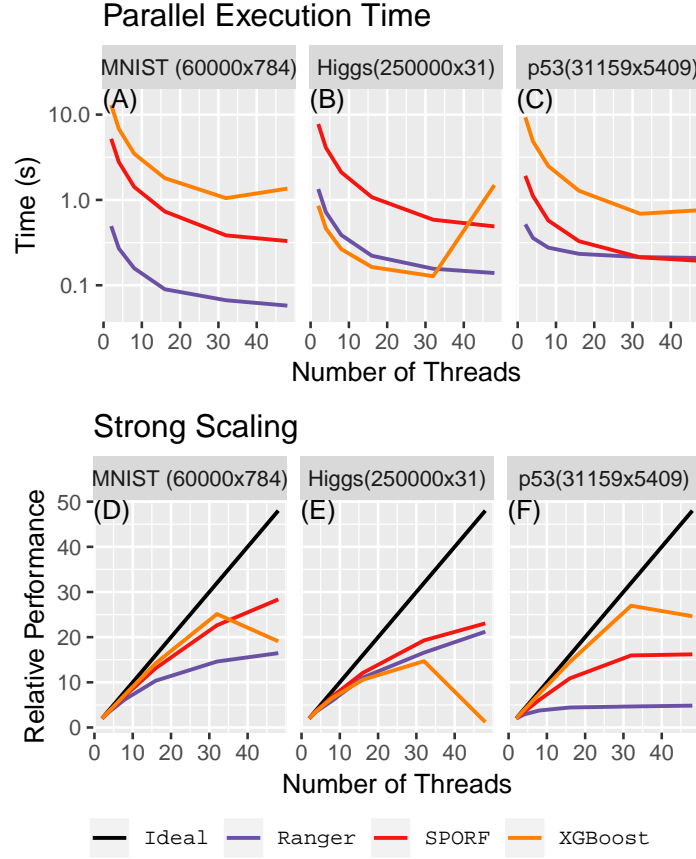


Figure 9: **(A-C)** The per-tree training time for three large real world datasets. Training was performed using matching parameters where possible, and default parameters otherwise. SPORF’s performance—even though it is written mostly in native R, as compared to the other optimized C++ codes—is comparable to the highly optimized XGBoost and Ranger and even outperforms XGBoost on two of the datasets. **(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 utilize 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 datasets. The p53 dataset, 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.

Figure 10 compares the prediction times of the various implementations on the same three datasets. In addition to our standard SPORF prediction implementation, we also compare a "Forest Packing" prediction implementation [?]. 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 datasets 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 datasets, and only marginally slower on the p53 dataset. 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 datasets.

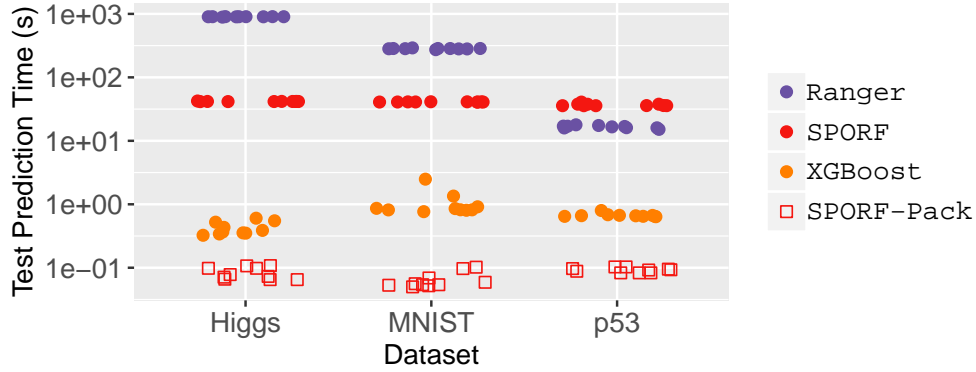


Figure 10: Comparison of test set prediction times. Forest Packing results show a 10x speed up in real time prediction scenarios. Test set sizes: Higgs, 50,000 observations; MNIST, 10,000 observations; p53, 6,000 observations. Predictions were made sequentially without batching.

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 introduced SPORF which was designed to preserve the desirable properties of both RF and 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.

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 empirical statistical consistency on a classification problem for which Biau et al. [30] proved that RF cannot achieve better than an error rate of $1/6$, even though the Bayes optimal error rate is zero in this setting. 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 in Biau et al. [40] 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 [5] 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 C. 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/>, 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.

References

- [1] M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, October 2014. 1, 12, 24
- [2] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. *Proceedings of the 25th International Conference on Machine Learning*, 2008.
- [3] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006. 1
- [4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016. 1, 4
- [5] L. Breiman. Random forests. *Machine Learning*, 4(1):5–32, October 2001. 1, 2, 3, 4, 20, 24
- [6] Rico Blaser and Piotr Fryzlewicz. Random rotation ensembles. *Journal of Machine Learning Research*, 17(4):1–26, 2016. 2, 4
- [7] Tom Rainforth and Frank Wood. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*, 2015. 2, 4, 6, 17
- [8] P. Li, T. J. Hastie, and K. W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296. ACM, 2006. 2, 4, 6
- [9] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. 1996. 3, 10
- [10] Tyler M Tomita, Mauro Maggioni, and Joshua T Vogelstein. Roflmao: Robust oblique forests with linear matrix operations. In *SIAM Data Mining*, 2017. 4, 5, 9
- [11] D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993. 4
- [12] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1619–1630, 2006. 4

- [13] B. H. Menze, B.M Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht. On oblique random forests. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgianis, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6912 of *Lecture Notes in Computer Science*, pages 453–469. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23782-9. doi: 10.1007/978-3-642-23783-6_29. URL http://dx.doi.org/10.1007/978-3-642-23783-6_29. 4
- [14] Donghoon Lee, Ming-Hsuan Yang, and Songhwai Oh. Fast and accurate head pose estimation via random projection forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1958–1966, 2015. 4
- [15] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM, 2001. 4
- [16] Xiaoli Z Fern and Carla E Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 186–193, 2003.
- [17] Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522. ACM, 2003.
- [18] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.
- [19] Chinmay Hegde, Michael Wakin, and Richard Baraniuk. Random projections for manifold learning. In *Advances in neural information processing systems*, pages 641–648, 2008. 4
- [20] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 537–546, New York, NY, USA, 2008. ACM. 4
- [21] S Dasgupta and Y Freund. Random projection trees for vector quantization. *IEEE Transactions on Information*, 2009. 4
- [22] S Dasgupta and K Sinha. Randomized partition trees for exact nearest neighbor search. *Conference on Learning Theory*, 2013. 4
- [23] Leo Breiman et al. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998. 4
- [24] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. 4
- [25] A J Wyner, M Olson, J Bleich, and D Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *J. Mach. Learn. Res.*, 2017. 4
- [26] 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 30*, pages 3146–3154. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf>. 5

- [27] Roman Vershynin. *High Dimensional Probability: An Introduction with Applications in Data Science*. 2019. 5
- [28] 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. 5
- [29] Leo Breiman and Adele Cutler. *Random Forests*. 5
- [30] G. Biau, L. Devroye, and G. Lugosi. Consistency of random forests and other averaging classifiers. *The Journal of Machine Learning Research*, 9:2015–2033, 2008. 7, 20
- [31] G. V. Trunk. A problem of dimensionality: A simple example. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (3):306–307, 1979. 10
- [32] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014. 16
- [33] James Browne, Tyler Tomita, and Joshua T. Vogelstein. *rerf: Randomer Forest*, 2018. URL <https://cran.r-project.org/web/packages/rerf/>. 17
- [34] Dirk Eddelbuettel. *Rcpp: Seamless R and C++ Integration*, 2018. URL <https://cran.r-project.org/web/packages/Rcpp/index.html>. 17
- [35] Tianqi Chen. *xgboost: Extreme Gradient Boosting*, 2018. URL <https://cran.r-project.org/web/packages/xgboost/>. 17
- [36] Marvin N. Wright. *ranger: A fast Implementation of Random Forests*, 2018. URL <https://cran.r-project.org/web/packages/ranger/>. 17
- [37] Yann Lecun, Corinna Cortes, and Christopher J.C. Burges. *The MNIST Database of Handwritten Digits*. 18
- [38] Alessandro Motta, Meike Schurr, Benedikt Staffler, and Moritz Helmstaedter. Big data in nanoscale connectomics, and the greed for training labels. *Curr. Opin. Neurobiol.*, 55:180–187, May 2019. 18
- [39] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. July 2018. 18
- [40] Gérard Biau, Erwan Scornet, and Johannes Welbl. Neural random forests. *arXiv preprint arXiv:1604.07143*, 2016. 20
- [41] Gareth M James. Variance and bias for general loss functions. *Machine Learning*, 51(2):115–135, 2003. 26, 27

Appendix A 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 B Real Benchmark datasets

We use 105 benchmark datasets from the UCI machine learning repository for classification. These datasets are most of the datasets used in Fernandez-Delgado et al. [1]; some were removed due to licensing or unavailability issues. We noticed certain anomalies in [1]’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 $\{c_1, \dots, c_m\}$, we expand to m binary features. If a data point has categorical value $c_k, \forall k \in 1, \dots, m$ 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 datasets 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 partition.** Each dataset 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 C 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 [5]. 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 11(A)). 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 11(B)). 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 11(C)). 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.

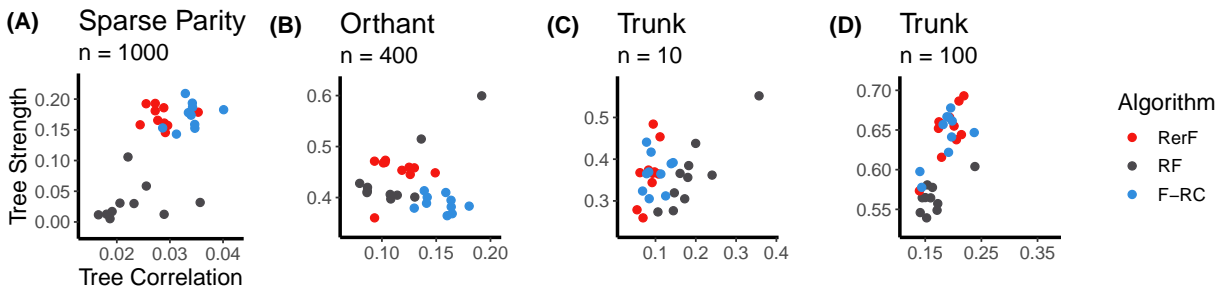


Figure 11: Comparison of tree strength and correlation of SPORF, RF, and F-RC on four of the simulated datasets: (A) sparse parity with $p = 10, n = 1000$, (B) orthant with $p = 6, n = 400$, (C) Trunk with $p = 10, n = 10$, and (D) Trunk with $p = 10, n = 100$. For a particular algorithm, there are ten 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 D 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 in James [41]. 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 [41] 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 in James [41]. Furthermore, let $P^*(X) = P_{Y|X}(Y = h^*(X)|X)$ and $\bar{P}(X) = P_{\mathcal{D}_n}(h(X|\mathcal{D}_n) = \bar{h}(X))$. 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 12(A), 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 12(B) 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 12(C) is similar to that in Figure 12(B), 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 12(D), 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).

Appendix E Algorithms

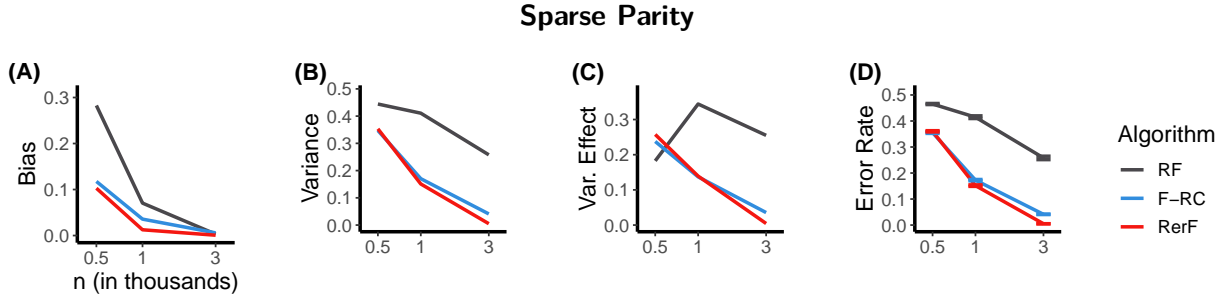


Figure 12: **(A-D)** Bias, variance, variance effect, and error rate, respectively, on the sparse parity problem as a function of the number of training samples. Error rate is the sum of systematic effect and variance effect, which roughly measure the contributions of bias and variance to the error rate, respectively. In this example, bias and systematic effect are identical because the Bayes error is zero (refer to [41]). For smaller training sets, SPORF wins primarily through lower bias/systematic effect, while for larger training sets it wins primarily through lower variance effect.

Algorithm 1 Learning a Random Projection Forest decision tree.

Input: (1) \mathcal{D}_n : training data (2) d : dimensionality of the projected space, (3) f_A : distribution of the random projection matrix, (4) Θ : set of split eligibility criteria

Output: A RPF decision tree T

```

1: function  $T = \text{GROWTREE}(\mathbf{X}, \mathbf{y}, f_A, \Theta)$ 
2:    $c = 1$  ▷  $c$  is the current node index
3:    $M = 1$  ▷  $M$  is the number of nodes currently existing
4:    $S^{(c)} = \text{bootstrap}(\{1, \dots, n\})$  ▷  $S^{(c)}$  is the indices of the observations at node  $c$ 
5:   while  $c < M + 1$  do ▷ visit each of the existing nodes
6:      $(\mathbf{X}', \mathbf{y}') = (\mathbf{x}_i, y_i)_{i \in S^{(c)}}$  ▷ data at the current node
7:     for  $k = 1, \dots, K$  do  $n_k^{(c)} = \sum_{i \in S^{(c)}} I[y_i = k]$  end for ▷ class counts (for classification)
8:     if  $\Theta$  satisfied then ▷ do we split this node?
9:        $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_d] \sim f_A$  ▷ sample random  $p \times d$  matrix
10:       $\tilde{\mathbf{X}} = \mathbf{A}^T \mathbf{X}' = (\tilde{\mathbf{x}}_i)_{i \in S^{(c)}}$  ▷ random projection into new feature space
11:       $(j^*, t^*) = \text{findbestsplit}(\tilde{\mathbf{X}}, \mathbf{y}')$  ▷ Algorithm 2
12:       $S^{(M+1)} = \{i : \mathbf{a}_{j^*} \cdot \tilde{\mathbf{x}}_i \leq t^* \mid i \in S^{(c)}\}$  ▷ assign to left child node
13:       $S^{(M+2)} = \{i : \mathbf{a}_{j^*} \cdot \tilde{\mathbf{x}}_i > t^* \mid i \in S^{(c)}\}$  ▷ assign to right child node
14:       $\mathbf{a}^{*(c)} = \mathbf{a}_{j^*}$  ▷ store best projection for current node
15:       $\tau^{*(c)} = t^*$  ▷ store best split threshold for current node
16:       $\kappa^{(c)} = \{M + 1, M + 2\}$  ▷ node indices of children of current node
17:       $M = M + 2$  ▷ update the number of nodes that exist
18:     else
19:        $(\mathbf{a}^{*(c)}, \tau^{*(c)}, \kappa^{(c)}) = \text{NULL}$ 
20:     end if
21:      $c = c + 1$  ▷ move to next node
22:   end while
23:   return  $(S^{(1)}, \{\mathbf{a}^{*(c)}, \tau^{*(c)}, \kappa^{(c)}\}_{c=1}^{m-1})$ 
24: end function

```

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 $\mathbf{X} \in \mathbb{R}^{p \times n}$, 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) $(\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{p \times n} \times \mathcal{Y}^n$, where $\mathcal{Y} = \{1, \dots, K\}$

Output: (1) dimension j^* , (2) split value τ^*

```

1: function  $(j^*, \tau^*) = \text{FINDBESTSPLIT}(\mathbf{X}, \mathbf{y})$ 
2:   for  $j = 1, \dots, p$  do
3:     Let  $\mathbf{x}^{(j)} = (x_1^{(j)}, \dots, x_n^{(j)})$  be the  $j$ th row of  $\mathbf{X}$ .
4:      $\{m_i^j\}_{i \in [n]} = \text{sort}(\mathbf{x}^{(j)})$  ▷  $m_i^j$  is the index of the  $i^{\text{th}}$  smallest value in  $\mathbf{x}^{(j)}$ 
5:      $t = 0$  ▷ initialize split to the left of all observations
6:      $n' = 0$  ▷ number of observations left of the current split
7:      $n'' = n$  ▷ number of observations right of the current split
8:     if (task is classification) then
9:       for  $k = 1, \dots, K$  do
10:         $n_k = \sum_{i=1}^n I[y_i = k]$  ▷ total number of observations in class  $k$ 
11:         $n'_k = 0$  ▷ number of observations in class  $k$  left of the current split
12:         $n''_k = n_k$  ▷ number of observations in class  $k$  right of the current split
13:      end for
14:    end if
15:    for  $t = 1, \dots, n - 1$  do ▷ assess split location, moving right one at a time
16:       $(\{n'_k, n''_k\}, n', n'', y_{m_t^j}) = \text{increment}(\{n'_k, n''_k\}, n', n'', y_{m_t^j})$ 
17:       $Q^{(j,t)} = \text{score}(\{n'_k, n''_k\}, n', n'')$  ▷ measure of split quality
18:    end for
19:  end for
20:   $(j^*, t^*) = \underset{j,t}{\text{argmax}} Q^{(j,t)}$ 
21:  for  $i = 0, 1$  do  $c_i = m_{t^*+i}^{j^*}$  end for
22:   $\tau^* = \frac{1}{2}(x_{c_0}^{(j^*)} + x_{c_1}^{(j^*)})$  ▷ compute the actual split location from the index  $j^*$ 
23:  return  $(j^*, \tau^*)$ 
24: end function

```

Appendix F Data Tables

Table 1: Five-fold cross-validation error rates (mean \pm 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, and p_{cat} is the number of categorical features.

Dataset	n	p_{num}	p_{cat}	5-fold CV Error Rate				
				SPORF	RF	XGBoost	RR-RF	CCF
abalone	4177	7	1	0.888 ± 0.019	0.89 ± 0.03	0.882 ± 0.03	0.909 ± 0.004	0.906 ± 0.009
acute_inflammation_task_1	120	6	0	0 ± 0.018	0 ± 0.1	0.05 ± 0.06	0 ± 0.003	0 ± 0.008
acute_inflammation_task_2	120	6	0	0 ± 0.009	0 ± 0.13	0.13 ± 0.05	0 ± 0.004	0 ± 0.011
adult	32561	7	7	0.1762 ± 0.012	0.1758 ± 0.15	0.1655 ± 0.07	0.2184 ± 0.18	0.1972 ± 0.01
annealing	798	27	5	0.02 ± 0.023	0.02 ± 0.07	0.03 ± 0.04	0.09 ± 0.06	0.06 ± 0.011
arrhythmia	452	279	0	0.28 ± 0	0.26 ± 0.11	0.28 ± 0.04	0.39 ± 0.19	0.34 ± 0.013
audiology_std	200	68	1	0.25 ± 0	0.27 ± 0	0.27 ± 0.04	0.35 ± 0.23	0.24 ± 0.012
balance_scale	625	4	0	0.06 ± 0.04	0.23 ± 0.02	0.17 ± 0.028	0.18 ± 0.14	0.1 ± 0.009
balloons	16	4	0	0.6 ± 0	0.5 ± 0.04	0.9 ± 0.035	0.4 ± 0.002	0.6 ± 0.012
bank	4521	11	5	0.083 ± 0	0.081 ± 0.01	0.081 ± 0.027	0.087 ± 0.002	$0.084 \pm 3e - 04$
blood	748	4	0	0.29 ± 0	0.28 ± 0.02	0.28 ± 0.031	0.29 ± 0.003	$0.3 \pm 3e - 04$
breast_cancer	286	7	2	0.39 ± 0	0.4 ± 0.063	0.42 ± 0.025	0.46 ± 0.001	$0.43 \pm 3e - 04$
breast_cancer-wisconsin	699	9	0	0.04 ± 0.19	0.04 ± 0.043	0.05 ± 0.5	0.04 ± 0.002	$0.04 \pm 3e - 04$
breast_cancer-wisconsin-diag	569	30	0	0.04 ± 0	0.06 ± 0.039	0.06 ± 0.4	0.04 ± 0.1	$0.03 \pm 2e - 04$
breast_cancer-wisconsin-prog	198	33	0	0.27 ± 0	0.31 ± 0.031	0.28 ± 0.5	0.28 ± 0.09	0.28 ± 0.02
car	1728	6	0	0.035 ± 0.0065	0.069 ± 0.031	0.035 ± 0.3	0.185 ± 0.11	0.031 ± 0.04
cardiotocography_task_1	2126	21	0	0.165 ± 0.0064	0.175 ± 0.03	0.156 ± 0.4	0.253 ± 0.04	0.185 ± 0.03
cardiotocography_task_2	2126	21	0	0.061 ± 0.0063	0.064 ± 0.01	0.055 ± 0.0029	0.099 ± 0.08	0.077 ± 0.05
chess_krvk	28056	0	6	0.1599 ± 0.0087	0.2201 ± 0.03	0.1324 ± 0.0025	0.4083 ± 0.1	0.149 ± 0.01
chess_krvkp	3196	35	1	0.009 ± 0.0063	0.012 ± 0.04	0.012 ± 0.0011	0.043 ± 0.1	0.014 ± 0.023
congressional_voting	435	16	0	0.06 ± 0.02	0.06 ± 0.04	0.04 ± 0.0049	0.06 ± 0.12	0.06 ± 0.014
conn_bench-sonar-mines-rocks	208	60	0	0.28 ± 0.03	0.28 ± 0.03	0.23 ± 0.0037	0.3 ± 0.11	0.24 ± 0.021
conn_bench-vowel-deterding	528	11	0	0.03 ± 0.03	0.04 ± 0.02	0.1 ± 0.05	0.03 ± 0.11	0.03 ± 0.038
contrac	1473	8	1	0.707 ± 0.04	0.735 ± 0.01	0.684 ± 0.04	0.751 ± 0.14	0.703 ± 0.028
credit_approval	690	10	5	0.23 ± 0.02	0.22 ± 0.01	0.23 ± 0.04	0.26 ± 0.15	0.23 ± 0.01
dermatology	366	34	0	0.02 ± 0.04	0.02 ± 0.02	0.03 ± 0.05	0.04 ± 0.15	0.03 ± 0.01
ecoli	336	7	0	0.17 ± 0.05	0.19 ± 0.02	0.19 ± 0.03	0.17 ± 0.07	0.19 ± 0.02
flags	194	22	6	0.43 ± 0.04	0.42 ± 0.05	0.43 ± 0.06	0.54 ± 0.14	0.42 ± 0.02
glass	214	9	0	0.36 ± 0.05	0.33 ± 0.03	0.35 ± 0.05	0.44 ± 0.2	0.32 ± 0.01
haberman_survival	306	3	0	0.37 ± 0.04	0.39 ± 0.03	0.37 ± 0.06	0.43 ± 0.2	0.46 ± 0.12

hayes_roth	132	0	4	0.32 ± 0.08	0.32 ± 0.03	0.36 ± 0.03	0.34 ± 0.2	0.32 ± 0.1
heart_cleveland	303	10	3	0.53 ± 0.11	0.52 ± 0.07	0.53 ± 0.03	0.49 ± 0.2	0.51 ± 0.08
heart_hungarian	294	10	3	0.11 ± 0.12	0.13 ± 0.08	0.13 ± 0.2	0.19 ± 0.2	0.15 ± 0.17
heart_switzerland	123	10	3	1.05 ± 0.09	0.99 ± 0.09	1.05 ± 0.2	0.98 ± 0.15	0.97 ± 0.23
heart_va	200	10	3	0.87 ± 0.12	0.87 ± 0.11	0.85 ± 0.4	0.83 ± 0.28	0.81 ± 0.023
hepatitis	155	19	0	0.56 ± 0.03	0.58 ± 0.09	0.63 ± 0.2	0.62 ± 0.24	0.46 ± 0.032
hill_valley	606	100	0	0 ± 0.06	0.88 ± 0.12	0.78 ± 0.3	0.12 ± 0.15	0 ± 0.035
hill_valley-noise	606	100	0	0.1 ± 0.02	0.97 ± 0.13	0.94 ± 0.0067	0.35 ± 0.3	0.1 ± 0.039
horse_colic	300	17	4	0.24 ± 0.02	0.26 ± 0.1	0.2 ± 0.0087	0.23 ± 0.1	0.25 ± 0.039
ilpd_indian-liver	583	10	0	0.4 ± 0.02	0.41 ± 0.15	0.4 ± 0.0054	0.38 ± 0.09	0.38 ± 0.02
image_segmentation	210	19	0	0.07 ± 0.5	0.08 ± 0.12	0.09 ± 0.0049	0.13 ± 0.17	0.07 ± 0.02
ionosphere	351	34	0	0.15 ± 0.4	0.18 ± 0.07	0.19 ± 0.0095	0.12 ± 0.17	0.14 ± 0.02
iris	150	4	0	0.09 ± 0.7	0.06 ± 0.07	0.08 ± 0.04	0.06 ± 0.07	0.03 ± 0.06
led_display	1000	7	0	0.318 ± 0.2	0.314 ± 0.04	0.305 ± 0.03	0.321 ± 0.03	0.319 ± 0.03
lenses	24	4	0	0.5 ± 0.7	0.6 ± 0.07	0.4 ± 0.03	0.7 ± 0.08	0.6 ± 0.01
letter	20000	16	0	0.0315 ± 0.002	0.0363 ± 0.08	0.0368 ± 0.03	0.0476 ± 0.11	0.0234 ± 0.01
libras	360	90	0	0.15 ± 0.003	0.2 ± 0.16	0.24 ± 0.04	0.16 ± 0.05	0.1 ± 0.009
low_res-spect	531	100	1	0.41 ± 0.003	0.49 ± 0.16	0.52 ± 0.04	0.52 ± 0.07	0.35 ± 0.009
lung_cancer	32	13	43	0.8 ± 0.009	0.6 ± 0.17	0.7 ± 0.12	0.8 ± 0.1	0.9 ± 0.009
magic	19020	10	0	0.1735 ± 0.002	0.1852 ± 0.15	0.1765 ± 0.04	0.2045 ± 0.2	0.179 ± 0.007
mammographic	961	3	2	0.31 ± 0.02	0.31 ± 0.14	0.31 ± 0.08	0.43 ± 0	0.39 ± 0.005
molec_biol-promoter	106	0	57	0.6 ± 0.02	0.64 ± 0.04	0.68 ± 0.15	0.85 ± 0.1	0.81 ± 0.006
molec_biol-splice	3190	0	60	0.07 ± 0.02	0.068 ± 0.04	0.058 ± 0.015	0.311 ± 0.2	0.069 ± 0.007
monks_1	124	2	4	0.02 ± 0.02	0.02 ± 0.03	0.18 ± 0.015	0.3 ± 0.005	0.19 ± 0.007
monks_2	169	2	4	0.63 ± 0.03	0.63 ± 0.01	0.55 ± 0.011	0.7 ± 0.012	0.36 ± 0.05
monks_3	122	2	4	0.14 ± 0.05	0.14 ± 0.03	0.19 ± 0.013	0.13 ± 0.009	0.18 ± 0.05
mushroom	8124	7	15	0 ± 0.06	0 ± 0.07	0.002 ± 0.016	0.001 ± 0.003	0 ± 0.03
musk_1	476	166	0	0.2 ± 0.02	0.21 ± 0.06	0.2 ± 0.03	0.21 ± 0.007	0.19 ± 0.04
musk_2	6598	166	0	0.026 ± 0.08	0.027 ± 0.08	0.018 ± 0.04	0.049 ± 0.07	0.023 ± 0.03
nursery	12960	6	2	$3e - 04 \pm 0.03$	0.0029 ± 0.06	$9e - 04 \pm 0.06$	0.038 ± 0.09	$5e - 04 \pm 0.11$
optical	3823	64	0	0.019 ± 0.03	0.021 ± 0.07	0.022 ± 0.12	0.023 ± 0.08	0.013 ± 0.1
ozone	2534	72	0	0.06 ± 0.04	0.059 ± 0.14	0.056 ± 0.09	0.061 ± 0.08	0.057 ± 0.08
page_blocks	5473	10	0	0.027 ± 0.04	0.028 ± 0.09	0.027 ± 0.11	0.031 ± 0.06	0.026 ± 0.09
parkinsons	195	22	0	0.31 ± 0.04	0.25 ± 0.14	0.33 ± 0.1	0.33 ± 0.01	0.21 ± 0.04

pendigits	7494	16	0	0.005 ± 0.04	0.009 ± 0.21	0.009 ± 0.12	0.007 ± 0.015	0.004 ± 0.005
pima	768	8	0	0.34 ± 0.03	0.36 ± 0.2	0.37 ± 0.07	0.35 ± 0.012	0.37 ± 0.005
pittsburgh_bridges-MATERIAL	106	4	3	0.45 ± 0.02	0.45 ± 0.09	0.49 ± 0.06	0.88 ± 0.02	0.59 ± 0.003
pittsburgh_bridges-REL-L	103	4	3	0.42 ± 0.03	0.42 ± 0.14	0.41 ± 0.1	0.48 ± 0.01	0.46 ± 0.02
pittsburgh_bridges-SPAN	92	4	3	0.6 ± 0.03	0.6 ± 0.11	0.6 ± 0.1	0.6 ± 0.03	0.5 ± 0.007
pittsburgh_bridges-T-OR-D	102	4	3	0.93 ± 0.04	0.93 ± 0.12	0.67 ± 0.07	0.93 ± 0.04	0.8 ± 0.013
pittsburgh_bridges-TYPE	106	4	3	0.61 ± 0.06	0.63 ± 0.1	0.73 ± 0.1	0.89 ± 0.04	0.65 ± 0.016
planning	182	12	0	0.4 ± 0.07	0.4 ± 0.15	0.52 ± 0.08	0.38 ± 0.03	0.42 ± 0.01
post_operative	90	8	0	0.4 ± 0.05	0.5 ± 0.33	0.4 ± 0	0.5 ± 0.06	0.5 ± 0.009
ringnorm	7400	20	0	0.039 ± 0.05	0.079 ± 0.16	0.037 ± 0	0.041 ± 0.016	0.044 ± 0.011
seeds	210	7	0	0.09 ± 0.05	0.1 ± 0.15	0.11 ± 0.002	0.12 ± 0.014	0.09 ± 0.012
semeion	1593	256	0	0.066 ± 0.005	0.063 ± 0.21	0.063 ± 0.001	0.09 ± 0.011	0.056 ± 0.012
soybean	307	22	13	0.1 ± 0.017	0.1 ± 0.01	0.1 ± 0	0.1 ± 0.011	0.09 ± 0.017
spambase	4601	57	0	0.071 ± 0.009	0.078 ± 0.05	0.074 ± 0.05	0.096 ± 0.011	0.066 ± 0.008
spect	80	22	0	0.7 ± 0.031	0.6 ± 0.09	0.6 ± 0.09	0.6 ± 0.3	0.7 ± 0.012
spectf	80	44	0	0.4 ± 0.008	0.5 ± 0.04	0.6 ± 0.07	0.4 ± 0.2	0.5 ± 0.07
statlog_australian-credit	690	10	4	0.23 ± 0.011	0.22 ± 0	0.24 ± 0.05	0.28 ± 0.3	0.25 ± 0.08
statlog_german-credit	1000	14	6	0.336 ± 0.013	0.35 ± 0.06	0.364 ± 0.06	0.384 ± 0.3	0.346 ± 0.04
statlog_heart	270	10	3	0.32 ± 0.011	0.3 ± 0.13	0.29 ± 0.007	0.31 ± 0.3	0.35 ± 0.04
statlog_image	2310	19	0	0.02 ± 0.018	0.023 ± 0.07	0.017 ± 0.008	0.035 ± 0.2	0.017 ± 0.04
statlog_landsat	4435	36	0	0.115 ± 0.023	0.117 ± 0.05	0.108 ± 0.005	0.122 ± 0.2	0.108 ± 0.067
statlog_shuttle	43500	9	0	$2e - 04 \pm 0.011$	$3e - 04 \pm 0.05$	$3e - 04 \pm 0.01$	0.0013 ± 0.1	$3e - 04 \pm 0.076$
statlog_vehicle	846	18	0	0.26 ± 0.014	0.32 ± 0.07	0.31 ± 0.01	0.31 ± 0.1	0.22 ± 0.065
steel_plates	1941	27	0	0.319 ± 0.009	0.306 ± 0.08	$0.289 \pm 5e - 04$	0.356 ± 0.3	0.336 ± 0.058
synthetic_control	600	60	0	0.02 ± 0.018	0.01 ± 0.02	0.02 ± 0.0012	0.02 ± 0.04	0.01 ± 0.059
teaching	151	3	2	0.61 ± 0.016	0.64 ± 0.07	0.7 ± 0.0011	0.61 ± 0.04	0.57 ± 0.048
thyroid	3772	21	0	0.032 ± 0.0041	0.028 ± 0.06	0.035 ± 0.0018	0.616 ± 0.06	0.053 ± 0.037
tic_tac-toe	958	0	9	0.04 ± 0.0041	0.03 ± 0.05	$0.03 \pm 7e - 04$	0.45 ± 0.03	0.05 ± 0.027
titanic	2201	2	1	0.309 ± 0.0079	0.309 ± 0.04	0.315 ± 0.005	0.315 ± 0.02	0.315 ± 0.048
twonorm	7400	20	0	0.045 ± 0.0028	0.052 ± 0.01	0.051 ± 0.006	0.043 ± 0.035	0.045 ± 0.038
vertebral_column_task_1	310	6	0	0.22 ± 0.0047	0.25 ± 0.06	0.28 ± 0.006	0.22 ± 0.037	0.24 ± 0.043
vertebral_column_task_2	310	6	0	0.32 ± 0.004	0.34 ± 0.01	0.34 ± 0.004	0.37 ± 0.065	0.31 ± 0.057
wall_following	5456	24	0	0.007 ± 0.005	0.007 ± 0.07	0.004 ± 0.004	0.166 ± 0.039	0.037 ± 0.054
waveform	5000	21	0	0.205 ± 0.006	0.221 ± 0.07	0.208 ± 0.006	0.205 ± 0.019	0.209 ± 0.042

waveform_noise	5000	40	0	0.201 ± 0.013	0.211 ± 0.05	0.208 ± 0.008	0.21 ± 0.03	0.197 ± 0.054
wine	178	13	0	0.05 ± 0.004	0.06 ± 0.06	0.03 ± 0.008	0.04 ± 0.05	0.03 ± 0.08
wine_quality-red	1599	11	0	0.527 ± 0.04	0.538 ± 0.05	0.548 ± 0.003	0.529 ± 0.1	0.523 ± 0.06
wine_quality-white	4898	11	0	0.562 ± 0.04	0.572 ± 0.05	0.58 ± 0.005	0.566 ± 0.06	0.57 ± 0.05
yeast	1484	8	0	0.523 ± 0.03	0.52 ± 0.04	0.528 ± 0.005	0.535 ± 0.02	0.538 ± 0.06
zoo	101	16	0	0.07 ± 0.02	0.06 ± 0.02	0.07 ± 0.003	0.06 ± 0.011	0.06 ± 0.08