Manifold Forests: Closing the Gap on Neural Networks

Ronan Perry,[1] Tyler M. Tomita,[1] Jesse Patsolic,[1] Benjamin Falk,[1]
Joshua T. Vogelstein[1,2,3]

Decision forests (DF), in particular random forests and gradient boosting trees, have demonstrated superior accuracy compared to other methods in many supervised learning scenarios. In particular, DFs dominate other methods in tabular data, that is, when the feature space is unstructured, so that the signal is invariant to permuting feature indices. However, in structured data lying on a manifold—such as images, text, and speech—neural nets (NN) tend to outperform DFs. We conjecture that at least part of the reason for this is that the input to NN is not simply the feature magnitudes, but also their indices (for example, the convolution operation uses "feature locality"). In contrast, naïve DF implementations fail to explicitly consider feature indices. A recently proposed DF approach demonstrates that DFs, for each node, implicitly sample a random matrix from some specific distribution. Here, we build on that to show that one can choose distributions in a *manifold aware fashion*. For example, for image classification, rather than randomly selecting pixels, one can randomly select contiguous patches. We demonstrate the empirical performance of data living on three different manifolds: images, time-series, and a torus. In all three cases, our Manifold Forest (MF) algorithm empirically dominates other state-of-the-art approaches that ignore feature space structure, achieving a lower classification error on all sample sizes. Moreover, both training and test time is significantly faster for manifold forests as compared to deep nets. This approach, therefore, has promise to enable DFs and other machine learning methods to close the gap with deep nets on manifold-valued data.

1 Introduction

Decision forests, including random forests and gradient boosting trees, have solidified themselves in the past couple decades as a powerful ensemble learning method in supervised settings [1, 2], including both classification and regression [3]. In classification, each forest is a collection of decision trees whose individual classifications of a data point are aggregated together using majority vote. One of the strengths of this approach is that each decision tree need only perform better than chance for the forest to be a strong learner, given a few assumptions [4, 5]. Additionally, decision trees are relatively interpretable because they can provide an understanding of which features are most important for correct classification [6]. Breiman originally proposed decision trees that partition the data set using hyperplanes aligned to feature axes [6]. Yet, this limits the flexibility of the forest and requires deep trees to classify some data sets, leading to overfitting. He also suggested that algorithms which partition based on sparse linear combinations of the coordinate axes can improve performance [6]. More recently, Sparse Projection Oblique Randomer Forest (SPORF), partitions a random projection of the data and has shown impressive improvement over other methods [7].

Yet random forests and other machine learning algorithms frequently operate in a tabular setting, viewing an observation $\vec{x} = (x_1, \ldots, x_p)^\mathsf{T} \in \mathbb{R}^p$ as an unstructured feature vector. In doing so, they neglect the indices in settings where the indices encode additional information. For structured data, e.g. images or time series, traditional decision forests are not able to incorporate known continuity between features to learn new features.

For decision forests to utilize known local structure in data, new features encoding this information must be manually constructed. Prior research has extended random forests to a variety of computer vision tasks [8–11] and augmented random forests with structured pixel label information [12]. Yet these methods either generate features a priori from individual pixels, and thus do not take advantage of the local topology, or lack the flexibility to *learn* relevant patches. Decision forests have been used to learn distance metrics on unknown manifolds [13], but such manifold forest algorithms are unsupervised and aim to learn a low dimensional representation of the data.

Inspired by SPORF, we propose a projection distribution that takes into account continuity between neighboring features while incorporating enough randomness to learn relevant projections. At each node in the decision tree, sets of random spatially contiguous features are randomly selected using knowledge of the underlying manifold. Summing the intensities of the sampled features yields a set of projections which can then be evaluated to partition the observations. We describe this proposed classification algorithm, Manifold Forests (MF) in detail and show its effectiveness in three simulation settings as compared to common classification algorithms. Furthermore, the optimized and parallelizable open source implementation of MF in R and Python is available (https://neurodata.com/sporf/). This addition makes for an effective and flexible learner across a wide range of manifold structures.

2 Background and Related Work

2.1 Classification

In the two-class classification setting, there is a data set $D = \{(x_i, y_i)\}_{i=1}^n$ of n pairs (x_i, y_i) drawn from an unknown distribution F_{XY} where $x_i \in \mathcal{X} \subset \mathbb{R}^p$ and $y_i \in \mathcal{Y} = \{0, 1\}$. Our goal is to train a classifier $h(x; D_n) : \mathcal{X} \times (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{Y}$ based on our observations that generalizes to correctly predict the class of an observed x. The performance of this classifier is evaluated via the 0-1 Loss function $L(h(x), y) = \mathbb{I}[h(x) \neq y]$ to find the optimal classifier $h^* = \operatorname{argmin}_h \mathbb{E}[L(h(x), y)]$, which minimizes the probability of an incorrect classification.

2.2 Random Forests

Originally popularized by Breiman, the random forest (RF) classifier is empirically very effective [1] while maintaining strong theoretical guarantees [6]. A random forest is an ensemble of decision trees whose individual classifications of a data point are aggregated together using majority vote. Each decision tree consists of split nodes and leaf nodes. A split node is associated with a subset of the data $S = \{(x_i, y_i)\} \subseteq D$ and splits into two child nodes, each associated with a binary partition of S. Let $e_j \in \mathbb{R}^p$ denote a unit vector in the standard basis (that is, a vector with a single one and the rest of the entries are zero) and τ a threshold value. Then S is partitioned into two subsets given the pair $\theta_j = \{e_j, \tau\}$.

$$S_\theta^L = \{(x_i, y_i) \mid e_j^\intercal x_i < \tau\}$$
$$S_\theta^R = \{(x_i, y_i) \mid e_j^\intercal x_i \geq \tau\}$$

To choose the partition, the optimal $\theta^* = (e_j^*, \tau^*)$ pair is selected via a greedy search from among a set of d randomly selected standard basis vectors e_j. The selected partition is that which maximizes some measure of information gain. A typical measure is a decrease in impurity, calculated by the Gini impurity score $I(S)$, of the resulting partitions [3]. Let $\hat{p}_k = \frac{1}{|S|} \sum_{y_i \in S} \mathbb{I}[y_i = k]$ be the fraction of elements of class k in partition S, then the optimal split is found as

$$I(S) = \sum_{k=1}^{K} \hat{p}_k (1 - \hat{p}_k)$$
$$\theta^* = \operatorname*{argmax}_\theta |S| I(S) - |S_\theta^L| I(S_\theta^L) - |S_\theta^R| I(S_\theta^R).$$

A leaf node is created once the partition reaches a stopping criterion, typically either falling below an impurity score threshold or a minimum number of observations [3]. The leaf nodes of the tree form a disjoint partition of the feature space in which each partition of observations S_k is assigned a class label c_k^* corresponding to the class majority.

$$c_k^* = \underset{c_k}{\operatorname{argmax}} \sum_{y_i \in S} \mathbb{I}(y_i = c_k).$$

A decision tree classifies a new observation by assigning it the class of the partition into which the observation falls. The forest averages the classifications over all decision trees to make the final classification [3]. For good performance of the ensemble and strong theoretical guarantees, the individual decision trees must be relatively uncorrelated from one another. Breiman's random forest algorithm does this in two ways:

1. At every node in the decision tree, the optimal split is determined over a random subset d of the total collection of features p.
2. Each tree is trained on a randomly bootstrapped sample of data points $D' \subset D$ from the full training data set.

Applying these techniques means that random forests do not overfit and lowers the upper bound of the generalization error [6].

2.3 Sparse Projection Oblique Randomer Forests

SPORF is a recent modification to random forest that has shown improvement over other versions [7, 14]. Recall that RF split nodes partition data along the coordinate axes by comparing the projection $e_j^\mathsf{T} x_i$ of observation x_i on standard basis e_j to a threshold value τ. SPORF generalizes the set of possible projections, allowing for the data to be partitioned along axes specified by any sparse vector a.

$$S_\theta^L = \{(x_i, y_i) \mid a^\mathsf{T} x_i < \tau\}$$
$$S_\theta^R = \{(x_i, y_i) \mid a^\mathsf{T} x_i \geq \tau\}$$

Rather than partitioning the data solely along the coordinate axes (i.e. the standard basis), SPORF creates partitions along axes specified by sparse vectors. In other words, let the dictionary $\mathcal{A}$ be the set of atoms $\{a\}$, each atom a p-dimensional vector defining a possible projection $a^\mathsf{T} x_i$. In axis-aligned forests, $\mathcal{A}$ is the set of standard basis vectors $\{e_j\}$. In SPORF, the dictionary $\mathcal{D}$ can be much larger, because it includes, for example, all 2-sparse vectors. At each split node, SPORF samples d atoms from $\mathcal{D}$ according to a specified distribution. By default, each of the d atoms are randomly generated with a sparsity level drawn from a Poisson distribution with a specified rate λ. Then, each of the non-zero elements are uniformly randomly assigned either $+1$ or -1. Note that the size of the dictionary for SPORF is 3^p (because each of the p elements could be -1, 0, or $+1$), although the atoms are sampled from a distribution heavily skewed towards sparsity.

3 Methods

3.1 Random Projection Forests on Manifolds

In the structured setting, the dictionary of projection vectors $\mathcal{A} = \{a\}$ is modified to take advantage of the underlying manifold on which the data lies. We term this method the Manifold Forest (MF).

Each atom a projects an observation to a real number and is designed with respect to prior knowledge of the data manifold. Nonzero elements of a effectively select and weight features. Since the feature space is structured, each element of a maps to a location on the underlying manifold. Thus, patterns of contiguous points on the manifold define the atoms of $\mathcal{A}$; the distribution of those patterns yields a distribution over the atoms. At each node in the decision tree, MF samples d atoms, yielding d new features per observation. MF proceeds just like SPORF by optimizing the best split according to the Gini index. Algorithm pseudocde, essentially equivalent to that of SPORF , can be found in the Appendix.

In the case of two-dimensional arrays, such as images, an observation $x_i \in \mathbb{R}^p$ is a vectorized representation of a data-matrix $X_i \in \mathbb{R}^{W \times H}$. To capture the relevance of neighboring pixels, MF creates projections by summing the intensities of pixels in rectangular patches. Thus the atoms of $\mathcal{A}$ are the vectorized representations of these rectangular patches.

A rectangular patch is fully parameterized by the location of its upper-left corner (u, v), its height h, and width w. To generate a patch, first the index of the upper left corner is sampled. Then its height and width are independently sampled from separate uniform distributions. MF hyperparameters determine the minimum and maximum heights heights $\{h_{min}, h_{max}\}$ and widths $\{w_{min}, w_{max}\}$, respectively, to sample from. Let $unif\{\alpha, \beta\}$ denote the discrete uniform distribution. An atom a is sampled as follows. Note that the patch cannot exceed the data-matrix boundaries.

$$\begin{aligned} &u \sim unif\{0, W - w_{min}\} &\qquad &v \sim unif\{0, H - h_{min}\} \\ &w \sim unif\{w_{min}, min(w_{max}, W - u)\} &\qquad &h \sim unif\{h_{min}, min(h_{max}, H - v)\} \end{aligned}$$

The vectorized atom a yields a projection of the data $a^{\mathsf{T}} x_i$, effectively selecting and summing pixel intensities in the sampled rectangular patch.

$$a^{\mathsf{T}} = (\underbrace{0,\ldots,0}_{W \times v}, \underbrace{0,\ldots,0}_{u}, \overbrace{\underbrace{1,\ldots,1}_{w}, \underbrace{0,\ldots,0}_{W-w}, \ldots, \underbrace{1,\ldots,1}_{w}, \underbrace{0,\ldots,0}_{W-w}}^{(h-1) \times W}, \underbrace{1,\ldots,1}_{w}, \underbrace{0,\ldots,0}_{W-w-u}, \quad \underbrace{0,\ldots,0}_{W \times (H-h-v)} \quad)$$

MF independently samples d atoms, generating d features per observation, and selects the best one to split on. By constructing features in this way, MF seeks to learn low-level features in the structured data, such as edges or corners in images. The forest can therefore learn the features that best distinguish a class. The structure of these atoms is flexible and task dependent. In the case of data lying on a cyclic manifold, the atoms can wrap-around borders to capture the added continuity. Atoms can also be used in one-dimensional arrays, such as univariate time-series data, in which case $h_{min} = h_{max} = 1$

3.2 Feature Importance

One of the benefits to decision trees is that their results are fairly interpretable in that they allow for estimation of the relative importance of each feature. Many approaches have been suggested [6, 15] and here a projection forest specific metric is used in which the number of times a given feature was used in projections across the ensemble of decision trees is counted. A decision tree T is composed of many nodes k, each one associated with an atom a^*_k and threshold that partition the feature space according to the projection $a^*_k \cdot x_i$. Thus, the indices corresponding to nonzero elements of a^*_k indicate important features used in the projection. For each feature j, the number of times π_j it is used in a projection, across all split nodes and decision trees, is counted.

$$\pi_j = \sum_{T} \sum_{k \in T} \mathbb{I}(a^*_{kj} \neq 0)$$

These normalized counts represent the relative importance of each feature in making a correct classification. Such a method applies to both SPORF and MF, although different results between them would be expected due to different projection distributions yielding different hyperplanes.

4 Simulation Results

To test MF, we evaluate its performance in three simulation settings as compared to logistic regression (Log. Reg), linear support vector machine (Lin. SVM), support vector machine with a radial basis function kernel (SVM), k-nearest neighbors (kNN), random forest (RF), Multi-layer Perceptron (MLP), and SPORF (SPORF). For each experiment, we used our open source implementation of MF and SPORF (available at `https://neurodata.io/sporf`. All decision forest algorithms used 100 decision trees on the simulations and 500 tree on the real data. Each of the other classifiers were run from the Scikit-learn Python package [16] with default parameters. Additionally, we tested against a Convolutional Neural Network (CNN) built using PyTorch [17] with two convolution layers, ReLU activations, and maxpooling, followed by dropout and two densely connected layers. The CNN results were averaged over 5 runs for the simulations and training was stopped early if the loss plateaued.

4.1 Simulation Settings

Experiment (A) is a non-Euclidean example inspired by [18]. Each observation is a discretization of a circle into 100 features with two non-adjacent segments of 1's in two differing patterns: class 1 features two segments of length five while class 2 features one segment of length four and one of length six. MF chose one-dimensional rectangles in this setting as the observations were one-dimensional in nature. These projection patches had a width between one and fifteen pixels and each split node of SPORF and MF considered 40 random projections. Figure (1) shows examples from the two classes and classification results across various sample sizes.

In experiment (B), a simple 28×28 binary image classification problem was constructed. Images in class 0 contain randomly sized and spaced horizontal bars while those in class 1 contain randomly sized and spaced vertical bars. For each sampled image, $k \sim Poisson(\lambda = 10)$ bars were distributed among the rows or columns, depending on the class. The distributions of the two classes are identical if a 90 degree rotation is applied to one of the classes. Projection patches were between one and four pixels in both width and height and each split node of SPORF and MF considered 28 random projections. Figure (1) shows examples from the two classes and classification results across various sample sizes.

Experiment (C) is a signal classification problem. One class consists of 100 values of Gaussian noise while the second class has an added exponentially decaying unit step beginning at time 20.

$$\begin{aligned} x_t^{(0)} &= \epsilon_t \\ x_t^{(1)} &= u(t-20)e^{(t-20)} + \epsilon_t \\ \epsilon_t &\sim \mathcal{N}(0,1) \end{aligned}$$

Projection patches were 1D with a width between one and five timesteps and each split node of SPORF and MF considered the default number of random projections. Figure (1) shows examples from the two classes and classification results across various sample sizes.

4.2 Classification Accuracy

In all three simulation settings, MF well outperforms all other classifiers, doing especially better at low sample sizes, except the CNN for which there is no clear winner. The performance of MF is particularly

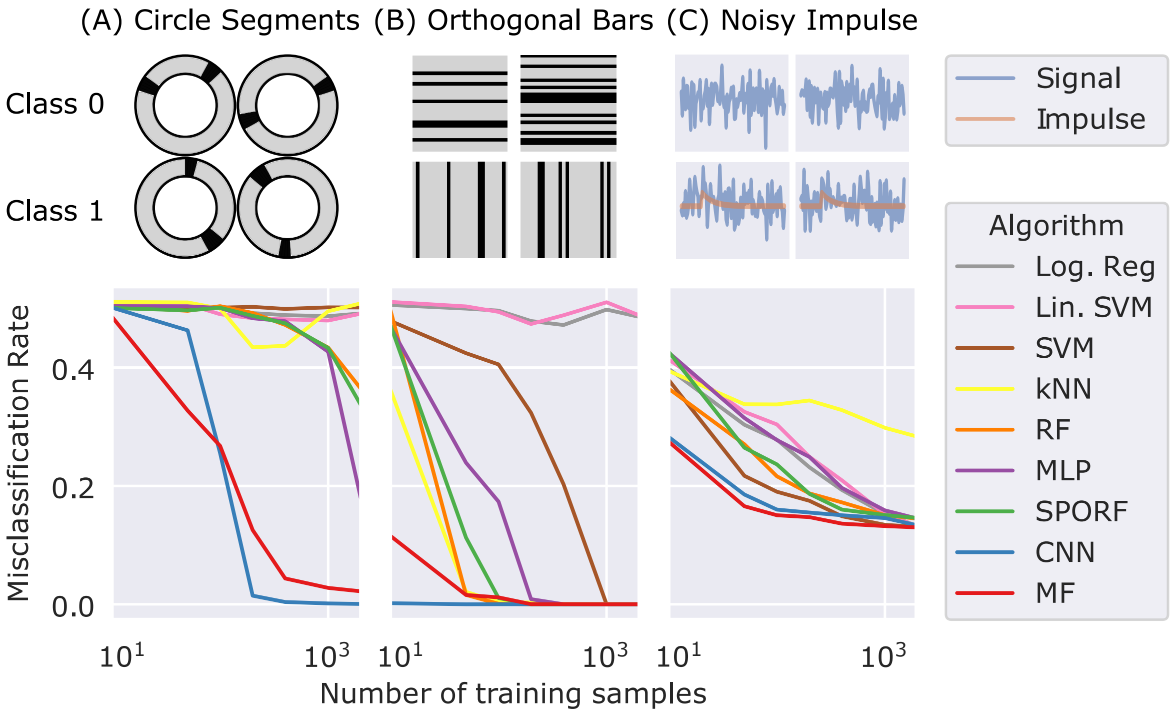

Figure 1: MF outperforms other algorithms in three two-class classification settings. Upper row shows examples of simulated data from each setting and class. Lower row shows misclassification rate in each setting, tested on 10,000 test samples. **(A)** Two segments in a discretized circle. Segment lengths vary by class. **(B)** Image setting with uniformly distributed horizontal or vertical bars. **(C)** White noise (class 0) vs. exponentially decaying unit impulse plus white noise (class 1).

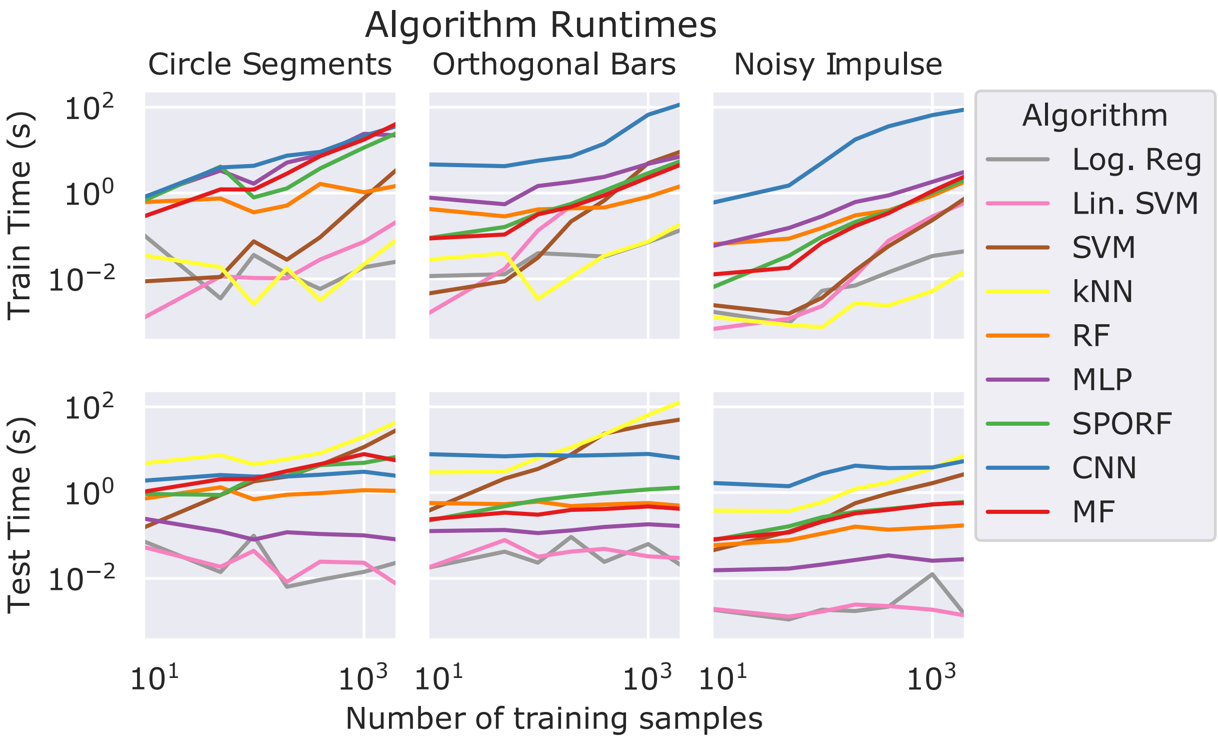

Figure 2: Algorithm train times (above) and test times (below). MF runtime is not particularly costly and well below CNN runtime in most examples.

good in the discretized circle simulation for which most other classifiers perform at chance levels. MF also performs well in the signal classification problem although all the classifiers are close in performance. This may be because the exponential signal is prevalent throughout most of the time-steps and so perfect continuity is less relevant.

4.3 Run Time

All experiments were run on a single core CPU. MF has train and test times on par with those of SPORF and RF and so is not particularly more computationally intensive to run. The CNN, however, took noticeably longer to run—especially in terms of training, but also in testing—in two of the three simulations. Thus its strong performance in those settings comes at an added computational cost, a typical issue for deep learning methods [19].

5 Real Data Results

MF was evaluated on a real world data set

5.1 Classification Accuracy

5.2 Feature Importance

To evaluate the capability of MF to identify importance features in manifold-valued data as compared to SPORF, both methods were run on a subset of the MNIST dataset, a collection of handwritten digits [20]. The two algorithms were trained in a two-class setting on handwritten threes and fives, 100 images from

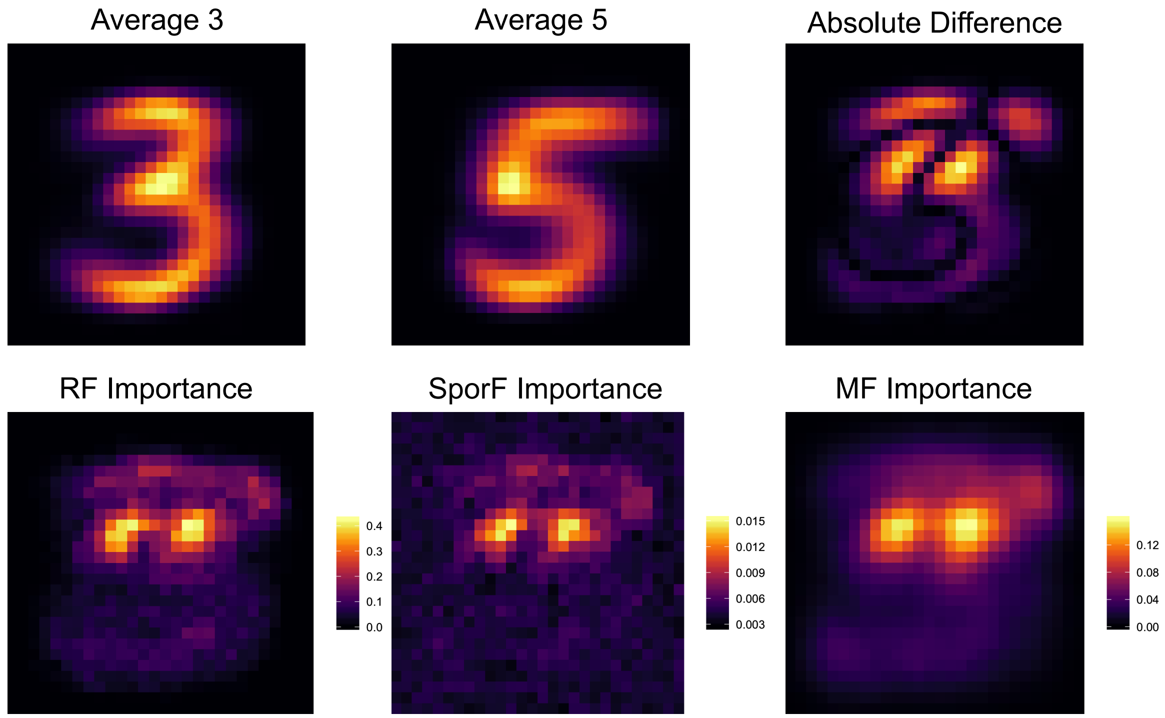

Figure 3: Averages of images in the two classes and their difference (above). Feature importance from MF (bottom left) shows less noise than SPORF (bottom middle) and is smoother than RF (bottom right).

each class.

The feature importance of each pixel is shown in Figure (3). MF visibly results in a smoother pixel importance, a result most likely from the continuity of neighboring pixels in selected projections. Although SPORF was shown to be an improvement over RF [7], its projection distribution yields scattered importance of unimportant background pixels whereas RF does not. Since projections in SPORF have no continuity constraint, those that select high importance pixels will also select pixels of low importance by chance. This may be a nonissue asymptotically, but is a relevant problem in low sample size settings. MF, however, shows little or no importance of these background pixels by virtue of the modified projection distribution.

6 Discussion

The success of sparse oblique projections in decision forests has opened up many possible ways to improve axis-aligned decision forests (including random forests and gradient boosting trees) by way of specialized projection distributions. Traditional decision forests have already been applied to structured data, using predefined features to classify images or pixels. Decision forest algorithms like that implemented in the Microsoft Kinect showed great success but ignore pixel continuity and specialize for a specific data modality, namely images. We expanded upon sparse oblique projections and introduced a structural projection distribution that uses prior knowledge of the topology of a feature space. Unlike some other manifold forest methods, ours does not attempt to learn the underlying manifold but rather uses a priori knowledge of the manifold to improve classification accuracy. The open source implementation[1] of SPORF has allowed for a relatively easy implementation of MF, creating a flexible classification method for a variety of data modalities. We showed in various simulated settings that appropriate domain knowledge

[1] https://neurodata.com/sporf/

can improve the projection distribution to yield impressive results that challenge the strength of deep learning techniques on manifold-valued data. Research into other, task-specific convolution kernels may lead to improved results in real-world computer vision tasks. Such structured projection distributions, while incorporated into SPORF here, may also be incorporated into other state of the art algorithms such as XGBOOST.

References

[1] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014. URL http://jmlr.org/papers/v15/delgado14a.html. 1, 2

[2] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143865. URL http://doi.acm.org/10.1145/1143844.1143865. 1

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. 1, 2, 3

[4] Robert E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5(2):197–227, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022648800760. URL https://doi.org/10.1023/A:1022648800760. 1

[5] Gérard Biau, Luc Devroye, and Gábor Lugosi. Consistency of random forests and other averaging classifiers. *J. Mach. Learn. Res.*, 9:2015–2033, June 2008. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1390681.1442799. 1

[6] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324. 1, 2, 3, 4

[7] Tyler M. Tomita, James Browne, Cencheng Shen, Jesse L. Patsolic, Jason Yim, Carey E. Priebe, Randal Burns, Mauro Maggioni, and Joshua T. Vogelstein. Random Projection Forests. *arXiv e-prints*, art. arXiv:1506.03410, Jun 2015. 1, 3, 8

[8] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 775–781 vol. 2, June 2005. doi: 10.1109/CVPR.2005.288. 1

[9] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky. Hough forests for object detection, tracking, and action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2188–2202, Nov 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2011.70.

[10] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, Oct 2007. doi: 10.1109/ICCV.2007.4409066.

[11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, June 2011. doi: 10.1109/CVPR.2011.5995316. 1

[12] P. Kontschieder, S. R. Bulò, H. Bischof, and M. Pelillo. Structured class-labels in random forests for semantic image labelling. In *2011 International Conference on Computer Vision*, pages 2190–2197, Nov 2011. doi: 10.1109/ICCV.2011.6126496. 1

[13] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Found. Trends. Comput. Graph. Vis.*, 7(2&#8211;3):81–227, February 2012. ISSN 1572-2740. doi: 10.1561/0600000035. URL http://dx.doi.org/10.1561/0600000035. 2

[14] Tyler M. Tomita, Mauro Maggioni, and Joshua T. Vogelstein. *ROFLMAO: Robust Oblique Forests with Linear MAtrix Operations*, pages 498–506. doi: 10.1137/1.9781611974973.56. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611974973.56. 3

[15] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *ArXiv*, abs/1705.07874, 2017. 4

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 5

[17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 5

[18] Laurent Younes. Diffeomorphic learning. *ArXiv*, abs/1806.01240, 2018. 5

[19] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 855–863, Cambridge, MA, USA, 2014. MIT Press. URL http://dl.acm.org/citation.cfm?id=2968826.2968922. 7

[20] Yann Lecun, Corinna Cortes, and Christopher J.C. Burges. *The MNIST Database of Handwritten Digits*. URL http://yann.lecun.com/exdb/mnist/. 7

7 Appendix

Algorithm 1 Learning a Manifold Forest decision tree.

```
Input: (1) $\mathcal{D}_n$: training data (2) $d$: dimensionality of the projected space, (3) $f_{\mathbf{A}}$: distribution of the
    atoms, (4) $\Theta$: set of split eligibility criteria
Output: A MF decision tree $T$
function $T$ = GROWTREE($\mathbf{X}, \mathbf{y}, f_{\mathbf{A}}, \Theta$)
    $c = 1$                                                   ▷ $c$ is the current node index
    $M = 1$                                                   ▷ $M$ is the number of nodes currently existing
    $S^{(c)}$ = bootstrap($\{1, ..., n\}$)                    ▷ $S^{(c)}$ is the indices of the observations at node $c$
    while $c < M + 1$ do                                      ▷ visit each of the existing nodes
        $(\mathbf{X}', \mathbf{y}') = (\mathbf{x}_i, y_i)_{i \in S^{(c)}}$          ▷ data at the current node
        for $k = 1, \ldots, K$ do $n_k^{(c)} = \sum_{i \in S^{(c)}} I[y_i = k]$ end for    ▷ class counts (for classification)
        if $\Theta$ satisfied then                            ▷ do we split this node?
            $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_d] \sim f_{\mathbf{A}}$          ▷ sample random $p \times d$ matrix of atoms
            $\widetilde{\mathbf{X}} = \mathbf{A}^T \mathbf{X}' = (\widetilde{\mathbf{x}}_i)_{i \in S^{(c)}}$    ▷ random projection into new feature space
            $(j^*, t^*)$ = findbestsplit($\widetilde{\mathbf{X}}, \mathbf{y}'$)     ▷ Algorithm 2
            $S^{(M+1)} = \{i : \mathbf{a}_{j^*} \cdot \widetilde{\mathbf{x}}_i \leq t^* \quad \forall i \in S^{(c)}\}$    ▷ assign to left child node
            $S^{(M+2)} = \{i : \mathbf{a}_{j^*} \cdot \widetilde{\mathbf{x}}_i > t^* \quad \forall i \in S^{(c)}\}$    ▷ assign to right child node
            $\mathbf{a}^{*(c)} = \mathbf{a}_{j^*}$                ▷ store best projection for current node
            $\tau^{*(c)} = t^*$                                   ▷ store best split threshold for current node
            $\kappa^{(c)} = \{M + 1, M + 2\}$                     ▷ node indices of children of current node
            $M = M + 2$                                           ▷ update the number of nodes that exist
        else
            $(\mathbf{a}^{*(c)}, \tau^{*(c)}, \kappa^{*(c)})$ = NULL
        end if
        $c = c + 1$                                           ▷ move to next node
    end while
    return $(S^{(1)}, \{\mathbf{a}^{*(c)}, \tau^{*(c)}, \kappa^{(c)}, \{n_k^{(c)}\}_{k \in \mathcal{Y}}\}_{c=1}^{m-1})$
end function
```

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

Affiliations

[1]Department of Biomedical Engineering, Johns Hopkins University

[2]Center for Imaging Science, Johns Hopkins University

[3]Kavli Neuroscience Discovery Institute, Johns Hopkins University