Canonical Correlation Forests

Tom Rainforth TWGR@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Parks Road, Oxford, OX1 3PJ, UK

Frank Wood FWOOD@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Parks Road, Oxford, OX1 3PJ, UK

Abstract

We introduce canonical correlation forests (CCFs), a new decision tree ensemble method for classification and regression. Individual canonical correlation trees are binary decision trees with hyperplane splits based on local canonical correlation coefficients calculated during training. Unlike axis-aligned alternatives, the decision surfaces of CCFs are not restricted to the coordinate system of the inputs features and therefore more naturally represent data with correlated inputs. CCFs naturally accommodate multiple outputs, provide a similar computational complexity to random forests, and inherit their impressive robustness to the choice of input parameters. As part of the CCF training algorithm, we also introduce projection bootstrapping, a novel alternative to bagging for oblique decision tree ensembles which maintains use of the full dataset in selecting split points, often leading to improvements in predictive accuracy. Our experiments show that, even without parameter tuning, CCFs out-perform axis-aligned random forests and other state-of-the-art tree ensemble methods on both classification and regression problems, delivering both improved predictive accuracy and faster training times. We further show that they outperform all of the 179 classifiers considered in a recent extensive survey.

Keywords: random forests, canonical correlation analysis, classification, regression, multiple output prediction

1. Introduction

Decision tree ensemble methods such as random forests (RF) (Breiman, 2001), extremely randomized trees (Geurts et al., 2006a), and boosted decision trees (Friedman, 2001) are widely employed methods for classification and regression due to their scalability, fast out of sample prediction, and tendency to require little parameter tuning. In many cases, they give predictive performance close to, or even equalling, state of the art when used in an out-of-the-box fashion (Fernández-Delgado et al., 2014), i.e. when parameters are set to default values. The individual trees used in such algorithms are, however, typically axis-aligned, restricting the ensemble decision surfaces to be piecewise axis-aligned, even when there is little evidence for this in the data. Canonical correlation forests (CCFs) overcome this problem by instead using carefully chosen hyperplane splits, leading to a more powerful classifier (or regressor) that naturally incorporates correlation between the features, an example for which shown in Figure 1. In this paper we demonstrate that this innovation regularly leads to a significant increase in out-of-sample predictive accuracy over previous state-of-

the-art tree ensemble methods, whilst maintaining speed and black-box applicability. Furthermore, we demonstrate that running CCFs without parameter tuning outperforms all of the 179 classifiers considered in the recent survey of Fernández-Delgado et al. (2014) over a large selection of datasets, even when parameter tuning is employed by the competing methods. We provide an open source code for CCFs,[1] delivering both training and testing in a single short line of code. Usage requires no expertise on the part of the user - all results presented in the paper are generated by using the package out-of-the-box, requiring only the data as input.

The two key factors underlying the performance of decision tree ensembles are the accuracy of the individual trees and the diversity in their predictions (Kuncheva and Whitaker, 2003; Elghazel et al., 2011). These often form conflicting aims as methods used for decorrelating tree predictions, such as bagging (Breiman, 1996) or random subspacing (Ho, 1998), typically rely on randomizing the tree training process, degrading the accuracy of individual trees. Not providing enough randomization in the tree training process causes high levels of correlation between tree predictions, diminishing the gains from aggregation and leading to reduced predictive accuracy and potentially overfitting (Brown et al., 2005). Inserting too much randomness degrades the predictive accuracy of the individual trees, again reducing the performance of the ensemble (Sollich and Krogh, 1996).

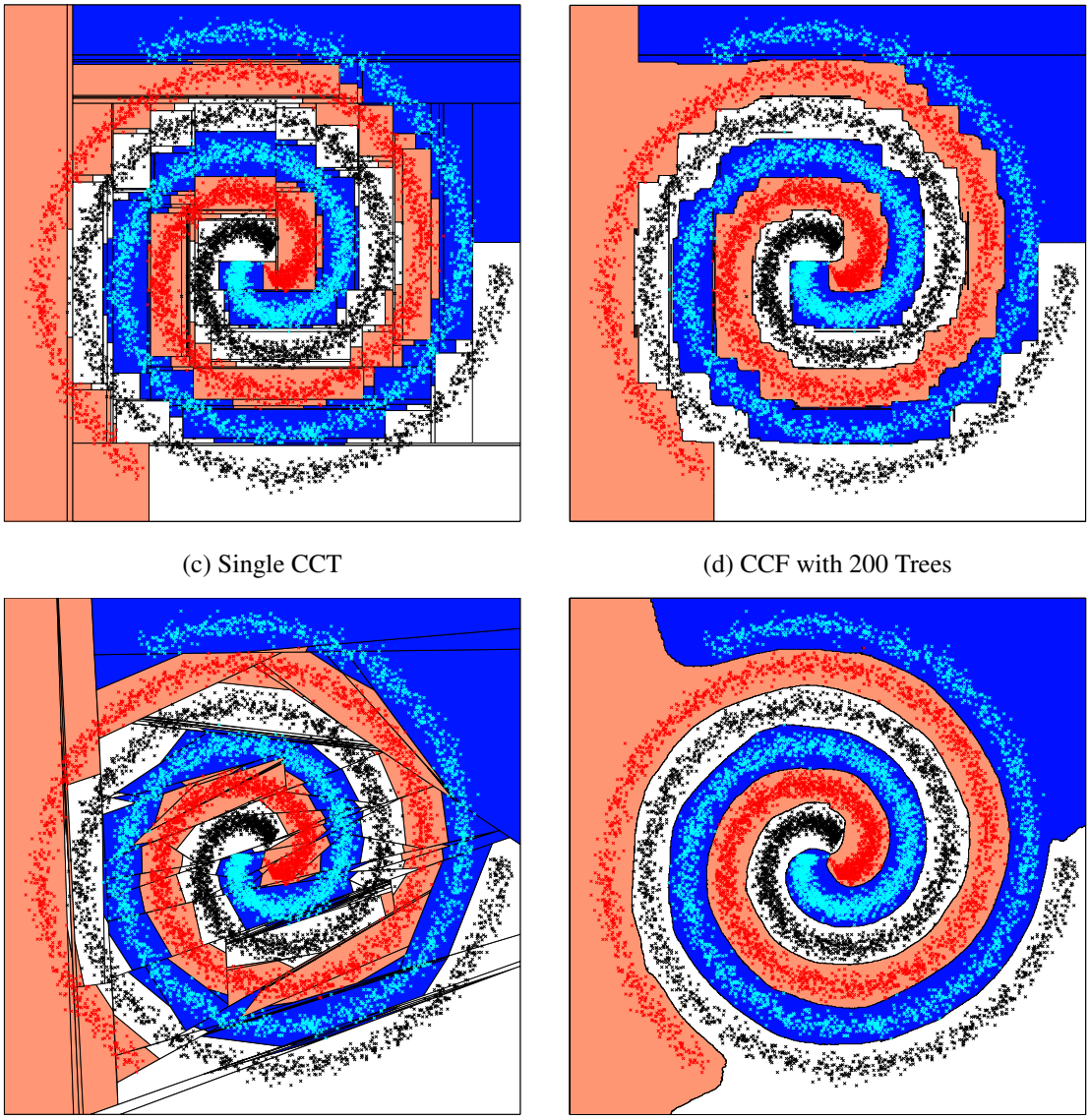
The use of axis-aligned splits by most decision tree ensemble methods can be detrimental to both these key factors. Individual tree performance can be hampered by an inability of axis-aligned splits to provide a good representation of the true decision surface, while the smaller set of possible splits reduces the ensemble diversity. The effect of the former on the ensemble performance can be particularly pronounced, resulting in sensitivity in the predictive accuracy to rotation of, and to correlation between, the input features (Menze et al., 2011). Recent work has looked to overcome this by randomly rotating or projecting the input features, either as preprocessing step before training each tree (Blaser and Fryzlewicz, 2016), or independently at each decision node (Breiman, 2001; Tomita et al., 2015, 2017). Such approaches can improve the ensemble diversity, alleviate sensitivity to rotation, and improve performance for many datasets. However, these gains are by no means ubiquitous, with the approaches ignoring potentially useful information in the choice of axes, such as when there are unordered categorical features. Furthermore, they do little to overcome the sensitivity of the performance to correlation between the input features.

CCFs instead use carefully chosen hyperplane splits based on applying canonical correlation analysis (CCA) (Hotelling, 1936) at each decision node during training. By using CCA, both output information and the correlation between the input features is naturally incorporated into the choice of projection. Further, as the hyperplane splits are calculated at each node separately, CCFs are better at incorporating local correlations than previous approaches. This also has the effect of reducing correlation between trees compared to axis-aligned approaches, as successive splits are no longer constrained to be orthogonal. CCFs maintain a very similar computational complexity to RFs and our empirical timings suggest that they are typically faster to train, due to evaluating less candidates splits on average at each node. Given that similar overall accuracy can often be achieved with only a small fraction of the number of trees compared with existing approaches, CCFs offer the potential for both substantial improvements in predictive accuracy and training time.

1. `https://github.com/twgr/ccfs/`

(a) Single CART

(b) RF with 200 Trees



(c) Single CCT

(d) CCF with 200 Trees

Figure 1: Decision surfaces for classification of artificial spirals dataset. The blue, red, and black crosses correspond to data points from three separate classes. The shading of the background shows the class predicted for that area by the respective classification algorithm. (a) Shows the hierarchical partitions and surface for a single axis aligned tree while (b) shows the RF decision surface arising from averaging over 200 axis aligned trees using the RF algorithm. (c) Shows a single canonical correlation tree (CCT) and (d) shows the CCF decision surface arising from averaging over 200 CCTs. Comparing (b) and (d) shows that the CCF has a smoother decision surface which better represents the data than the RF. More precisely the decision surface is piecewise linear for the CCF and piecewise orthogonal for the RF.

2. Background

2.1 Decision Tree Ensembles

A decision tree is a predictive model that imposes sequential divisions of an input space to form a set of partitions known as leafs, each containing a local classification or regression model. Out-of-sample prediction is performed by using the partitioning to assign each new input to a particular leaf and then using the corresponding local predictive model. Typically, the leaf models are taken to be independent of each other and the outputs are assumed to be independent of input features given the leaf assignments.

Classical decision tree learning algorithms work in a greedy top down fashion, exhaustively searching the space of unique axis-aligned splits and choosing the best based on a splitting criterion, such as the Gini gain or information gain used in CART (Breiman et al., 1984) and C4.5 (Quinlan, 2014) respectively. Once a split is chosen, it is used to assigned each data point in the training set to one of the newly generated child nodes. The process then recurses in a self-similar manner for each of the child nodes, continuing until no further split is advantageous or some user-set limit is reached for all the generated nodes. For classification with continuous features this typically only occurs once each leaf is "pure," containing only data points of a single class. When used as individual classifiers or regressors, trees are usually "pruned" after being grown to prevent overfitting. This is a regularization process which involves collapsing sub-branches of the tree to single leaf nodes.

It was established by Ho (1995) that combining individual trees to form a decision tree ensemble, or forest, can simultaneously improve predictive performance and provide regularization against overfitting without the need for pruning. In a forest, each tree is separately trained, with prediction based on averaging predictions from the individual trees, resulting in a predictor that typically outperforms any single constituent tree (Rokach, 2010). As classical decision tree algorithms are deterministic procedures, such combination requires the introduction of probabilistic elements into the generative process to prevent identical trees. One possible approach, the random subspace method (Ho, 1995, 1998), involves only searching splits using a randomly selected subset of the features at each node. Bagging (Breiman, 1996) instead trains each tree on a bootstrap sample of the original dataset. Arguably the most widely used decision tree ensemble approach, random forests (RFs) (Breiman, 2001), combines these randomization approaches, providing improved out-of-sample predictive performance to using either approach in isolation.

2.2 Oblique Decision Trees and Forests

Oblique decision trees (ODTs) extend classical decision trees by splitting using linear combinations of the available features. Some algorithms such as OC1 (Murthy et al., 1994) attempt to directly optimize for the hyperplane representing the best split, while others, such as functional trees (Gama, 2004) and QUEST (Loh and Shih, 1997), carry out a linear discriminant analysis (LDA) (Fisher, 1936; Rao, 1948) to find a projection which optimizes the discriminant criterion and then search over possible splits in this projected space. Although ODTs generally produce better results than axis aligned trees, many existing algorithms suffer from a number of common issues such as a failure to effectively deal with multiple classes, numerical instability or significant increase in computational cost compared with an axis aligned alternative. Many also only apply to classification problems and do not naturally extend to regression or multiple output settings.

A number of approaches for constructing oblique decision forests using random projections have been suggested in the literature. Random rotation ensembles (Blaser and Fryzlewicz, 2016) apply a separate random rotation to the input features for each tree in the ensemble, training the tree in the standard axis-aligned way on this rotated space. As such, each tree uses orthogonal splits, but in a different co-ordinate system. Forest-RC (Breiman, 2001) instead uses random linear combinations of subsets of the input features to generate hyperplane splits at each node, such that each tree is itself an oblique decision tree. Randomer forests (Tomita et al., 2015) also use random linear combinations of features at each node, but instead of sub-sampling fixed numbers of features for each linear combination, they use sparse random projections of all the features. All these methods have been shown to offer improvements in predictive accuracy to RFs on some datasets, but also worse performance on others. In particular, they tend to perform poorly on problems with unordered categorical features.

Rotation forests (Rodriguez et al., 2006) take a more structured approach. Instead of using bagging or random subspacing, they apply a rotation pre-processing step using principle component analysis (PCA) on random groupings of the input features. This significantly reduces sensitivity to correlation between features and delivers significant performance improvements over both RFs and randomly rotated alternatives. However, this improvement comes at a significant computational cost, as the method considers all the features when splitting each node, as opposed to using a small subset of them as done by methods employing random subspacing. Furthermore, as all splits in a particular tree are orthogonal to one another and as PCA does not incorporate class information, the performance of rotation forests is still sensitive to localized or class dependent correlations. Perhaps because of the faster speed of training, axis-aligned RFs are still more prevalently used than rotation forests, despite the typically improved predictive accuracy of the latter.

Lemmond et al. (2008) and Menze et al. (2011) both introduced the idea of creating forests of oblique decision trees for classification using splits based on LDA projections, providing noticeable improvements in predictive accuracy over RFs. However, both methods only apply to binary classification and neither carries out LDA in a manner that is both numerically stable and computationally efficient. The latter paper also introduces the idea of carrying out a ridge regression adjusted LDA where there is regularization towards the principle component directions. However, their results suggest no advantage is gained by this regularization compared to using LDA directly.

In work developed independently to our own, Zhang and Suganthan (2014) consider three more generally applicable approaches: PCA-RF, LDA-RF, and RF-ensemble. PCA-RF uses PCA to project the features at each node, and then calculates splits in this projected space. LDA-RF instead uses a multiclass form of LDA to calculate the projections. RF-ensemble constructs an ensemble where each tree uses either PCA, LDA, or axis-aligned splits depending on which approach provided the best split at the root node for that tree. They show that all three approaches deliver regular improvements over RFs on a selection of (predominantly) UCI datasets (Lichman, 2013), though they do not provide comparisons with rotation forests. Their results also suggest that LDA-RF outperforms PCA-RF and that RF-ensemble provides small further improvements on both. Weaknesses of their methods include not applying to regression or multi-output problems,[2] a combinatorial computational complexity in number of categories for categorical features, and numerical instability in the LDA calculation (De la Torre, 2012).

2. Though it is not suggested in the paper, PCA-RF could in theory be applied to regression and multiple output problems. LDA-RF and RF-ensemble, on the other hand, do not generalize beyond single output classification.

2.3 Canonical Correlation Analysis

Canonical correlation analysis (CCA) (Hotelling, 1936) is a deterministic method for calculating pairs of linear projections that maximise the correlation between two matrices in the co-projected space. It is a co-ordinate free process that is unaffected by rotation, translation or global scaling of the inputs. Consider applying CCA between the arbitrary matrices $W \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{n\times k}$. The first pair of canonical coefficients are given by

$$\{A_1, B_1\} = \underset{a\in\mathbb{R}^d, b\in\mathbb{R}^k}{\operatorname{argmax}} \left(\operatorname{corr}\left(Wa, Vb\right)\right) \tag{1}$$

subject to the conditions $\|a\|_2 = 1$ and $\|b\|_2 = 1$. The corresponding canonical correlation components are given by WA_1 and VB_1. Another $\nu_{\max} - 1$ pairs are created where $\nu_{\max} = \min\left(\operatorname{rank}\left(W\right), \operatorname{rank}\left(V\right)\right)$ by repeating the same optimization with the additional constraints that the new components are uncorrelated with all previous components, e.g.:

$$\left(WA_1\right)^T WA_2 = 0 \quad \text{and} \quad \left(VB_1\right)^T VB_2 = 0. \tag{2}$$

As shown by, for example, Cohen and Ben-Israel (1969); Borga (2001), the solution of (1) has a simple closed form (see Appendix A). However, this solution can be numerically unstable as it requires an inversion of typically degenerate covariance matrices. Björck and Golub (1973) demonstrated that the solution for CCA can also be found in a numerically stable fashion using a combination of QR (Householder, 1958) and SVD (Golub and Reinsch, 1970) decompositions. Details of our numerically stable approach based on this work are given in Appendix A.

A key motivation for using CCA is that it is applies for any two matrices. This means that, by using a 1-of-K class encoding for classification problems, we can use CCA, and therefore CCFs, for regression, classification, multi-output classification, and multivariate regression, all of which will be considered later in the paper. We note that for the case of single-output classification, then the feature projection matrix produced by our CCA approach is analytically equivalent to that produced by multi-class LDA (Hastie et al., 1995; De la Torre, 2012). For a single-output regression, CCA is instead equivalent to projecting onto the hyperplane produced by the output of a linear regression (Sun et al., 2008). Along with unifying these cases and generalizing to multiple outputs, there are also numerical advantages to using CCA. For example, the CCA approach given in Appendix A provides more numerical stability and easier regularization than standard approaches for calculating LDA. This is particularly important given that the analysis is generally done on a bootstrap sample of the original data, such that the problem is highly likely to be degenerate.

3. Canonical Correlation Forests

3.1 Overview

We start by providing a high level overview of CCFs, before introducing formal notation and an in-depth description later in the section. As with RFs and most other decision tree ensemble methods, the trees in a CCF are independently trained in a greedy, self-similar, top down procedure that successively chooses the best split (according to some split criterion) by exhaustively searching over a set of candidates. The algorithm starts with a root node containing all the training data and each time a new split is selected, this creates two new child nodes with each data points passed down to either the left or right child depending on what side of the split the data point falls. The

same procedure for choosing the best split is then applied to each of the child nodes and this process continues recursively until all nodes have met a stopping criterion or contain only a single unique data point, at which point they become leaf nodes.

There are two key differences between the training algorithms for CCFs and RFs. Firstly, for RF training each tree is trained on a bootstrap sample of the data in a process known as bagging, while for CCFs, each tree is trained on the full training dataset. Secondly, CCFs consider a different set of split candidates. In RF training then a random subset of the features are considered at each node and the set of split candidates corresponds to all unique axis aligned partitions of the data using these features. In CCF training a random subset of the features is also taken, but CCA with projection bootstrapping (see Section 3.3.1) is used first to project the features into canonical component space, with the set of split candidates corresponding to the unique partitions in this projected space. The chosen partition then implies a hyperplane split that can be used directly at test time. This training process is summarized in Figure 2, using an example classification problem. The resulting CCT along with an axis aligned alternative is shown in Figure 3.

3.2 Forest Definition and Prediction

Though CCFs are able to deal with categorical inputs, missing data, and multiple outputs (see Sections 3.3.2 and 6), we neglect these cases for now in the interest of clarity. Our aim is to make predictions $v \in \mathbb{V}$ about outputs $y \in \mathbb{Y}$, given corresponding vectors of input features $x \in \mathbb{R}^D$. For regression, the outputs y are single numeric values $y \in \mathbb{R}$. For classification, the outputs are class labels $k \in \{1, \ldots, K\}$, but for notational convenience we represent these using a 1-of-K encoding $y \in \mathbb{I}^K$, where for class k the k^{th} element of y is set to 1 and all other values are set to 0. Though the predictions convey information about the outputs, it need not be the case that $\mathbb{V} = \mathbb{Y}$. For example, for classification we consider predicting relative class probabilities, $v \in [0, 1]^K$, as we are uncertain about the true class. For regression we will only consider predicting a point estimate for the output, i.e. $\mathbb{V} = \mathbb{Y}$, though it is possible to also calculate uncertainty estimates in the same way as for RFs (see for example Criminisi et al. (2011)).

CCFs represent a supervised learning approach and so require training using labelled input pairs. Once trained, prediction can be carried out at arbitrary input points. We consider using a training dataset comprising of N inputs $X = \{x_n\}_{n=1}^N$ and corresponding outputs $Y = \{y_n\}_{n=1}^N$. Here X and Y can be conveniently represented as matrices, with the first index denoting the data point by convention. We use the notation $X_{(u,v)}$ for indexing, where u and v can both be either scalars or vectors of indexes and : indicates all instances in that dimension. Thus, for example, $Y_{(1,:)}$ corresponds to y_1 and $X_{([1,4],[2,3])}$ denotes a matrix containing the second and third features of the first and fourth data points. Vectors are indexed in the same way. For example $x_{([1,3])}$ represents the first and third dimensions (i.e. features) of x.

Let $T = \{t_i\}_{i=1\ldots L}$ denote a CCF, comprising of L individual canonical correlation trees (CCTs) t_i. When clear from the context, we will omit the tree index i to avoid clutter. As with classical decision trees, CCTs define a hierarchical partitioning on the input space. Each individual tree $t_i = \{\Psi, \Theta\}$ is defined by a set of discriminant nodes $\Psi = \{\psi_j\}_{j \in \mathcal{J} \setminus \partial\mathcal{J}}$ and a set of leaf nodes $\Theta = \{\theta_j\}_{j \in \partial\mathcal{J}}$ where $\mathcal{J} \subset \mathbb{Z}^{\geq 0}$ is the set of node indices and $\partial\mathcal{J} \subseteq \mathcal{J}$ is the subset of leaf node indices. Each discriminant node is defined by the tuple $\psi_j = \{j, \delta_j, \phi_j, s_j, \chi_{j,\ell}, \chi_{j,r}\}$ where j is the unique node identifier, $\delta_j \in \{1, \ldots, D\}^\lambda$ is a vector of indexes giving the features used for splitting

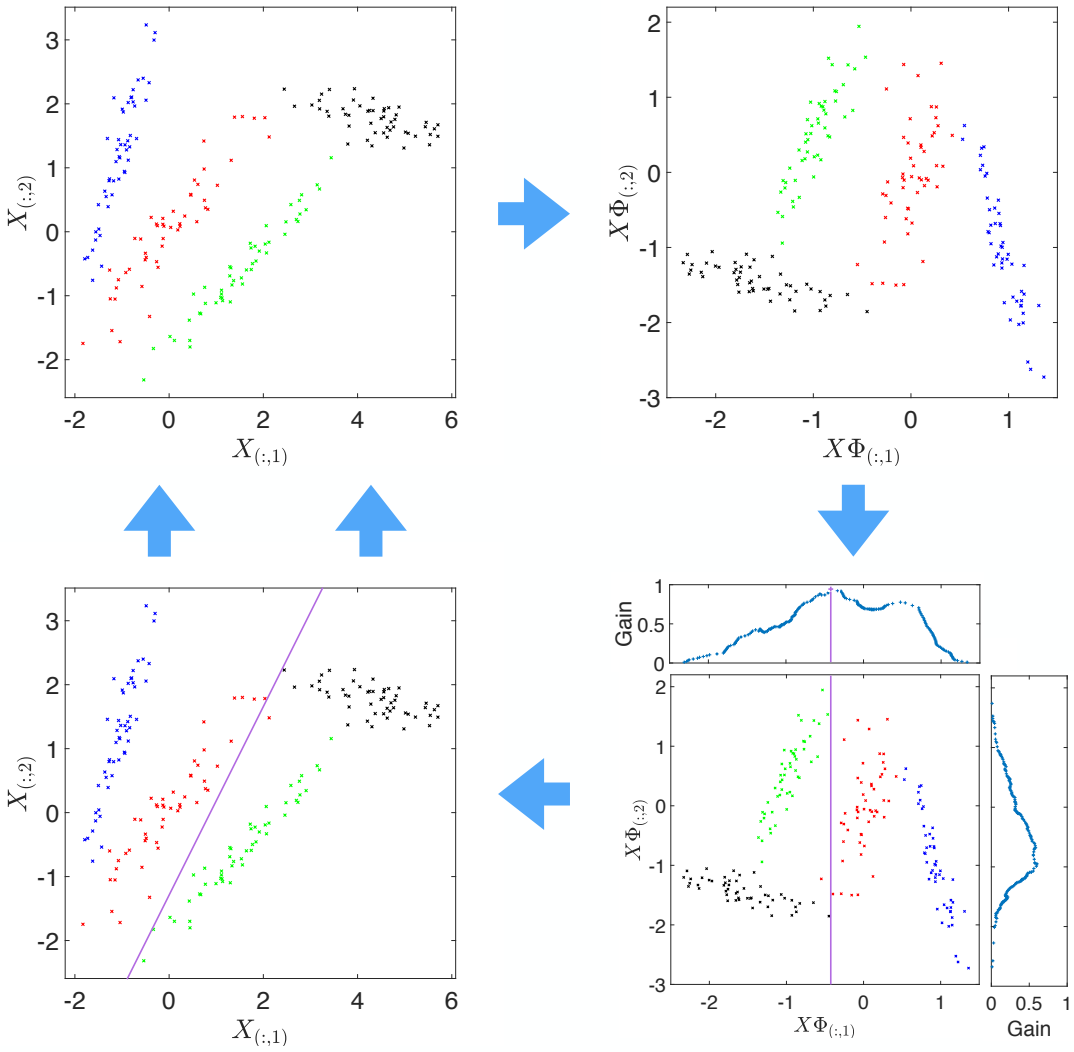


Figure 2: High level demonstration of CCT training process for a classification task. Given the original training data at the node, λ ($= 2$ in this example) features are subsampled for consideration (top left). CCA with projection bootstrapping (see Section 3.3.1) is carried out to project the data into a space that maximally correlates the inputs with the outputs (top right). Note how knowing only one of the features in this projected space is significantly more informative about the class identity than knowing only one of the features in the original space. For example, knowing the horizontal axis value in the projected space is enough to uniquely identify whether a point is in the blue class or not. The next step is to calculate a split criterion (e.g. information gain) for each of the possible unqiue splits (splits are always taken halfway between adjacent points) in this projected space (bottom right). The split with the highest gain is selected for splitting, implying a hyperplane split in the original space (bottom left). The self-similar process is then repeated separately on each of the two newly created partitions, terminating when each partition has no further advantageous split (e.g. because it only contains data points of a particular class). The tree is complete (see Figure 3) when all branches have reached termination.

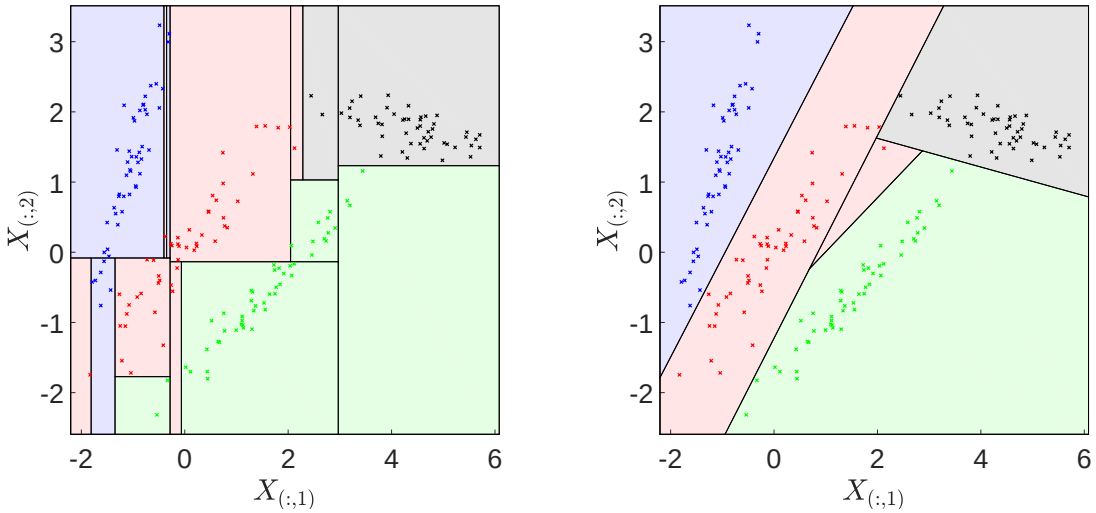


Figure 3: Tree resulting from the axis aligned decision tree algorithm CART (Breiman et al., 1984) (left) and CCT training (right) on the dataset from Figure 2.

at the node, $\phi_j \in \mathbb{R}^\lambda$ is a weight vector used to project these features, $s_j \in \mathbb{R}$ is the point at which the split occurs in the projected space $x^T_{(\delta_j)}\phi_j$, and $\{\chi_{j,\ell}, \chi_{j,r}\} \subseteq \mathcal{J}\backslash j$ are the two child node ids.

Let $B(j,t)$ denote the partition of the input space associated with node j of tree t such that $B(0,t) = \mathbb{R}^D$ and $B(j,t) = B(\chi_{j,\ell},t) \cup B(\chi_{j,r},t)$. The partitioning procedure is then defined such that

$$\begin{aligned} B(\chi_{j,\ell},t) &= B(j,t) \cap \left\{x \in \mathbb{R}^D : x^T_{(\delta_j)}\phi_j \leq s_j\right\} \\ B(\chi_{j,r},t) &= B(j,t) \cap \left\{x \in \mathbb{R}^D : x^T_{(\delta_j)}\phi_j > s_j\right\}. \end{aligned} \tag{3}$$

Thus Ψ defines a hierarchical partitioning procedure that deterministically assigns inputs to leaf nodes. We further introduce the notation ω_j to indicate the indices of the training points which fall into the partition of node j, i.e.

$$\omega_j = \{n \in 1 \ldots N \colon x_n \in B(j,t)\}, \tag{4}$$

and $N_j = \|\omega_j\|_0$ as the corresponding number of training points at the node.

Each leaf node is itself defined by a local prediction model $\theta_j : \mathbb{R}^D \to \mathbb{V}$.[3] Although more complicated leaf models are possible, using for example logistic regression (Gama, 2004) or kernels (Geurts et al., 2006b), in this paper we only consider the case where the leaf model is a deterministic assignment to the average of the outputs reaching that leaf in the training set. Specifically

$$\theta_j(x) = \frac{1}{N_j}\sum_{n \in \omega_j} y_n, \tag{5}$$

3. Technically speaking, the prediction space of individual trees need not be same as that of forest, see for example Criminisi et al. (2011). However, we omit consideration of this scenario for clarity.

such that the tree's prediction is independent of the input, given the leaf assignment. We therefore drop the dependence on x, such that each leaf node is fully defined by its identifier j and prediction $\theta_j \in \mathbb{V}$.

As with RFs, out of sample prediction in CCFs is done using an equally weighted voting scheme of the tree predictions. Slightly abusing notation, let $t_i(x) \in \mathbb{V}$ denote the prediction of t_i for input x. The forest's prediction is then given by

$$T(x) = \frac{1}{L} \sum_{i=1}^{L} t_i(x). \tag{6}$$

We note that more complicated methods of combining the tree predictions could be used instead, see, for example, Robnik-Šikonja (2004) and (Geurts et al., 2006b).

3.3 Forest Training

Algorithms 1 and 2 give a step by step walk-through for training a CCF, outlining the forest training and tree growing processes respectively. In both cases some simplifications have been made for clarity, with more comprehensive algorithm blocks provided in Appendix B, additionally detailing things such as categorical features and missing data. As we can see from Algorithm 1, CCF training requires as inputs the data $\{X, Y\}$, a number number of trees $L \in \mathbb{Z}^+$, a number of features to sub-sample $\lambda \in \{1, \ldots, D\}$, an impurity measure $g : \mathbb{Y}^{\mathbb{Z}^+} \to \mathbb{R}$ (see (12) and (13)), and stopping criteria c (see Section 3.3.3). Other than the data, all of these required inputs have default values, meaning that CCFs can be trained without parameter tuning, as is done in all of our experiments. Setting these options will be discussed in 3.3.3, for now we just note that all these options are shared with RFs and therefore the intuition for how they should be set predominantly transfers. We therefore also refer the reader to previous work that examines the effect these parameters on decision tree ensembles more generally, e.g. Bernard et al. (2009); Criminisi et al. (2011).

As should hopefully be clear by now, each tree in a CCF is trained independently using the full dataset. The tree training process, GROWTREE, is self similar, starting at the root node with the full dataset. At each call it selects an optimal split for a set of generated candidates (lines 1 to 18); decides whether to assign the current node as a leaf or discriminant node (line 19); and then if the node is assigned as a decision node, it recursively calls GROWTREE again (with appropriate partitions of the data) to produce sub-trees for each of the newly generated child nodes (lines 26 to 29). The training process is complete once all generated branches have terminated as leaf nodes.

The process for deciding whether to assign a node as a leaf node or a decision node is straightforward - the node is assigned to be a leaf if no split is beneficial (as per the split gain discussed later) or if a stopping criterion is met (see Section 3.3.3). Otherwise it becomes a decision node. The key part of the algorithm is therefore the process for selecting the split, namely selecting the features used for splitting δ_j, the projection vector ϕ_j, and the split point s_j. This can be further broken down into generating a set of candidate splits and selecting the optimal split from this candidate set.

As in RFs, the process of generating the set of candidate splits for CCFs starts by randomly sampling, δ_j, the subset of the features to consider at that node (lines 1 and 2 of Algorithm 2). This corresponds to sampling λ features without replacement from all of those present. For RFs, the set of candidate splits comprises of all unique axis-aligned splits using these features. For CCFs the candidate splits will be hyperplane splits implied by the possible axis-aligned splits in a projected

Algorithm 1: CCF training algorithm

Inputs: features $X \in \mathbb{R}^{N\times D}$, outputs $Y \in \mathbb{Y}^N$, number of trees $L \in \mathbb{Z}^+$ (default is 500), number of features to sub-sample $\lambda \in \{1,\dots,D\}$ (default is $\lceil \log_2(D)+1 \rceil$), impurity measure $g : \mathbb{Y}^{\mathbb{Z}^+} \to \mathbb{R}$ (default is (12) for classification and (13) for regression), stopping criteria c (see Section 3.3)

Outputs: CCF T

Preprocess X ▷ See Section 3.3.3
for $i = 1 : L$ **do**
  Randomly assign missing values in X ▷ See Section 3.3.3
  $t_i \leftarrow \text{GROWTREE}(0, X, Y, \lambda, g, c)$ ▷ Each tree trained independently
end for
return $T = \{t_i\}_{i=1\dots L}$

Algorithm 2: GROWTREE

Inputs: unique node identifier for root node j, $X^j = X_{(\omega_j,:)} \in \mathbb{R}^{N_j\times D}$, $Y^j = Y_{(\omega_j,:)} \in \mathbb{Y}^{N_j}$, λ, g, c

Outputs: subtree $\{\Psi_j, \Theta_j\}$ where Ψ_j are discriminant nodes and Θ_j leaf nodes

Subsample features ids δ_j by sampling from $\{1,\dots,D\}$ λ times without replacement.
Set $\mathcal{X} \leftarrow X^j_{(:,\delta_j)}$
Construct a bootstrap sample $\{\mathcal{X}', \mathcal{Y}'\}$ by sampling N_j data points with replacement from $\left\{\mathcal{X}, Y^j\right\}$
$\{\Phi, \Omega\} \leftarrow \text{CCA}(\mathcal{X}', \mathcal{Y}')$ ▷ Calculate CCA coefficients using bootstrap sample
$U \leftarrow \mathcal{X}\Phi$ ▷ Project *original features* into canonical component space
$G_{\text{base}} \leftarrow g\left(Y^j\right)$ ▷ Impurity of current node
$\nu_{\max} = \min\left(\text{rank}\left(\mathcal{X}'\right), \text{rank}\left(\mathcal{Y}'\right)\right)$ ▷ Number of projections
for $\nu \in 1 : \nu_{\max}$ **do** ▷ Evaluate splits for each projection
  $u \leftarrow \text{SORT}(U_{(:,\nu)})$
  for $i \in 2 : N_j$ **do** ▷ Exhaustive search on unique splits
    $S_{(i,\nu)} \leftarrow (u_{(i-1)} + u_{(i)})/2$ ▷ Split halfway between consecutive points
    $\tau^\ell \leftarrow \left\{n \in \{1,\dots,N_j\} : U_{(n,\nu)} \le S_{(i,\nu)}\right\}$ ▷ Points that would be assigned to left child
    $N_{\chi_{j,\ell}} \leftarrow$ number of elements in τ^ℓ
    $\tau^r \leftarrow \{1,\dots,N_j\} \setminus \tau^\ell$
    $G_{(i,\nu)} \leftarrow G_{\text{base}} - \frac{N_{\chi_{j,\ell}}}{N_j} g\left(Y_{(\tau^\ell,:)}\right) - \frac{N_j - N_{\chi_{j,\ell}}}{N_j} g\left(Y_{(\tau^r,:)}\right)$ ▷ Split gain
  end for
end for
$\{i^*, \nu^*\} \leftarrow \text{argmax}_{i,\nu}\, G_{(i,\nu)}$ ▷ Choose best split
if $G_{(i^*,\nu^*)} \le 0$ (i.e. no split is beneficial) or any stopping of criteria c satisfied **then** ▷ Node is a leaf
  $\theta_j \leftarrow \frac{1}{N_j}\sum_{n=1}^{N_j} Y^j_{(n,:)}$ ▷ Predictive model is average of points at leaf
  return $\left\{\cdot, \{j, \theta_j\}\right\}$
else ▷ Node is a discriminant node
  Generate unique identifiers for children $\chi_{j,\ell}$ and $\chi_{j,r}$
  $\phi_j \leftarrow \Phi_{(:,\nu^*)}, \quad s_j \leftarrow S_{(i^*,\nu^*)}$ ▷ Chosen projection and split point
  $\psi_j \leftarrow \left\{j, \delta_j, \phi_j, s_j, \chi_{j,\ell}, \chi_{j,r}\right\}$
  $\tau^\ell \leftarrow \left\{n \in \{1,\dots,N_j\} : U_{(n,\nu^*)} \le S_{(i^*,\nu^*)}\right\}$ ▷ Assign data points to children
  $\tau^r \leftarrow \{1,\dots,N_j\} \setminus \tau^\ell$
  $\{\Psi_\ell, \Theta_\ell\} \leftarrow \text{GROWTREE}(\chi_{j,\ell}, X^j_{(\tau^\ell,:)}, Y^j_{(\tau^\ell,:)}, \lambda, g, c)$ ▷ Recurse for left child and right child
  $\{\Psi_r, \Theta_r\} \leftarrow \text{GROWTREE}(\chi_{j,r}, X^j_{(\tau^r,:)}, Y^j_{(\tau^r,:)}, \lambda, g, c)$
  return $\{\psi_j \cup \Psi_\ell \cup \Psi_r, \Theta_\ell \cup \Theta_r\}$
end if

space $X\Phi$. To calculate Φ, we first take a bootstrap sample of the local data at the node $\{\mathcal{X}', \mathcal{Y}'\}$ (line 3 of Algorithm 2). Using this bootstrap sample we then carry out a CCA between the features and the outputs (line 4 of Algorithm 2)

$$[\Phi, \Omega] = \text{CCA}\left(\mathcal{X}', \mathcal{Y}'\right) \tag{7}$$

where Φ and Ω are the canonical coefficients corresponding to $\mathcal{X}'$ and $\mathcal{Y}'$ respectively. We note that Ω is not directly used by the algorithm. Our set of candidate splits is now the unique axis-aligned splits in the projection of the *original* input features into the canonical component space, namely (line 4 of Algorithm 2)

$$U = X_{(\omega_j, \delta_j)}\Phi \tag{8}$$

remembering that ω_j is the index of data points present at node j. Choosing a split in the space of U implies a hyperplane split in the space of X, given by the projection ϕ_j, corresponding to one of the columns of Φ, and the split point s_j in the projected space. Note that CCA is only required during the training phase with the splitting rule (3) used directly for out of sample prediction.

The process of choosing the best split from the set of possible candidates is analogous to the exhaustive search approach used by most decision tree methods (including RFs). The only difference is that the search uses U instead of $X_{(\omega_j, \delta_j)}$. The core idea is to use an impurity criterion $g : \mathbb{Y}^{\mathbb{Z}^+} \to \mathbb{R}$ and to choose the split which most reduces the impurity (lines 15 and 18 of Algorithm 2). Namely we wish to solve

$$\begin{aligned}\{\phi_j, s_j\} &= \underset{\phi\in\Phi, s\in\mathbb{R}}{\text{argmax}}\ G\left(Y_{(\omega_j,:)}, \phi, s\right) \\ &= \underset{\phi\in\Phi, s\in\mathbb{R}}{\text{argmax}}\ \left(g\left(Y_{(\omega_j,:)}\right) - \frac{N_{\chi_{j,\ell}}}{N_j} g\left(Y_{(\omega_{\chi_{j,\ell}},:)}\right) - \frac{N_{\chi_{j,r}}}{N_j} g\left(Y_{(\omega_{\chi_{j,r}},:)}\right)\right)\end{aligned} \tag{9}$$

where $G\left(Y_{(\omega_j,:)}, \phi, s\right)$ is commonly referred to as the gain of a split and using (3), (4), and (8) we have that

$$\omega_{\chi_{j,\ell}}(\phi, s) = \left\{n \in \omega_j \colon X_{(n,\delta_j)}\phi \leq s\right\} \tag{10}$$

$$\omega_{\chi_{j,r}}(\phi, s) = \left\{n \in \omega_j \colon X_{(n,\delta_j)}\phi > s\right\} = \omega_j \backslash \omega_{\chi_{j,\ell}}(\phi, s)\,. \tag{11}$$

We see that the gain of a split depends only on how the data points are assigned to each of the children, such that any $\{\phi, s\}$ that leads to the same partitioning of the training data gives the same gain. By construction we already have that there are only a finite number of ϕ considered and so the set of unique possible splits is now fully defined by the associated values of s that lead to distinct partitions of the training data. We define this by restricting valid splits to be halfway between consecutive points in the respective sorted column of U (see lines 9 and 11 of Algorithm 2). Consequently there are $(N_j - 1)\nu_{\max}$ possible splits where $\nu_{\max}$ is the number of columns of Φ (see Section 2.3).

The choice of impurity criterion depends on whether classification or regression is being performed. The intuitive property we desire for the impurity criterion is that it is high when the uncertainty in the output is high or equivalently when homogeneity in the outputs for points at the node

is low. For classification we take as a default the entropy of the node defined as

$$g\left(Y_{(\omega_j,:)}\right) = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{12}$$

where p_k is the empirical probability of class k at the node, namely $p_k = \frac{1}{N_j}\sum_{n\in\omega_j} Y_{(n,k)}$. The entropy impurity criterion is thus the entropy of the predictive distribution that would be generated if the node were to become a leaf. Using the entropy as the impurity metric corresponds to using information gain as the split criterion, as introduced by Quinlan (1986). A common alternative for classification is the Gini impurity $g\left(Y_{(\omega_j,:)}\right) = 1 - \sum_{k=1}^{K} p_k^2$ (Breiman et al., 1984). Entropy is taken as a default due to slightly superior performance in our experiments, but we note that performance variations were small and the overall improvement was certainly not definitive.

For regression we use the empirical variance,

$$g\left(Y_{(\omega_j)}\right) = \frac{1}{N_j}\sum_{n\in\omega_j} Y_{(n)}^2 - \left(\frac{1}{N_j}\sum_{n\in\omega_j} Y_{(n)}\right)^2, \tag{13}$$

as an impurity measure (Breiman et al., 1984). Using this impurity measure is often referred to as the mean squared error split criterion as it corresponds to the mean square error that would result from using the mean of the samples (which is the predictive model taken if the node is a leaf) as the prediction for the training data at that node. Alternative split criteria, such those based on curvature (Loh, 2002), could also be used, but we will not consider these further here.

Given a split and the assignment of the node to be a decision node, all that remains is to recurse to the children. This simply involves assigning all datapoints to either the left or right child using (3) (lines 26 and 27 in Algorithm 2) and calling GROWTREE for each child using the corresponding partition of the training data.

3.3.1 PROJECTION BOOTSTRAPPING

We refer to calculating the projection matrix Φ using a local bootstrap sampling of the data but then using the original dataset for choosing the split in the projected space as *projection bootstrapping*. The motivation for projection bootstrapping, instead of say using bagging, is that it retains all the information from the dataset when choosing $\{\phi_j, s_j\}$ from the candidates, thereby improving the predictive accuracy of individual trees. This is beneficial because the use of hyperplane splits increases the diversity of the ensemble compared with axis-aligned forests. Therefore less artificial randomness needs to be added to the tree training process to sufficiently decorrelate the tree predictions. Consequently, as shown in Section 4, projection bootstrapping leads to improved ensemble predictive performance compared to bagging. We found, on the other hand, that omitting both bagging and projection bootstrapping, and thus relying on random subspacing alone to decorrelate tree predictions, reduced the randomness in the tree training process too far, leading to degraded performance.

An important exception to using projection bootstrapping is in the case where $\lambda = D$, i.e. when we consider all of the features at each node, for which we use bagging instead. The reason for this is that no feature subspacing occurs in this scenario and so additional randomness needs to be added into the training process to prevent overfitting. Using the default settings for λ, this scenario occurs only when $D \leq 2$.

3.3.2 DATA PREPROCESSING

The format of our forest definition given in Section 3.2 requires X to be in numerical form. This is not a problem for ordered categorical features which can simply be treated as numerical features using the class index. For unordered categorical features $x^c \in \mathcal{S}$, where $\mathcal{S}$ represents the space of arbitrary qualitative attributes, we use a 1-of-K encoding such that the feature is converted into K binary features. To ensure equal probability of selecting categorical and numerical features, the expanded binary array of each categorical feature is still treated as a single feature during the random subspacing (i.e. sampling δ_j, see Algorithm 5). In the rest of the paper we refer to numeric, binary, and ordered categorical features (i.e. those that do not require this expansion) as ordinal and non-ordered categorical features as non-ordinal.

A second preprocessing step is to centre and rescale the input data so that it has zero mean and unit variance for each dimension (ignoring any missing data). As CCA and the split criteria are invariant under affine transformations of the full dataset, this does not have any direct bearing on the tree training. However it ensures equal influence of the input features in the rank reduction used to ensure stability of the CCA calculation as described in Section A.

The final preprocessing step is to deal with missing data. Here we take the simple approach of randomly assigning each missing value in X to a random draw from a unit Gaussian (noting that we have previously ensure that each feature has zero mean and unit variance). This is done independently for each tree so that each tree is affected in a different way. The same approach is used for both training and testing. Though we found this simple approach worked well in practise, one could also envisage using more complicated schemes, such as sampling from the empirical distribution of the feature over the training data instead.

3.3.3 STOPPING CRITERIA AND OTHER PARAMETER SETTINGS

As previously stated, the CCF training algorithm has four parameters of note: the number of trees L, the number of features to sub-sample at each node λ, the impurity measure g, and the stopping criteria c. We reiterate that all of these parameters are common to RFs.

Setting g has already been discussed in Section 3.3, using entropy and variance as defaults for classification and regression respectively. The setting of L need only be based on computational budget as it is commonly accepted (Criminisi et al., 2011), and there are theoretical results suggesting (Breiman, 2001; Biau, 2012), that more trees is always preferable for common decision ensemble methods (with the exception of boosting which can overfit if L is too large). We take $L = 500$ as a default value as compromise between speed and accuracy, noting that, as shown in Section 7.4, the misclassification rate has usually stabilized by this value of L. It may often, however, be beneficial to use significantly smaller values for L when speed is paramount none-the-less. From this perspective CCFs offer the potential for significant speed improvements over RFs, as we found that on average only 15 CCTs are required to match the accuracy of 500 RF trees (see Section 7.4). When the number of features is very large, it may also be beneficial to use more than 500 trees.

The default number of features to sample at each step is taken to be $\lambda = \lceil \log_2(D) + 1 \rceil$ where $\lceil \cdot \rceil$ represents the ceiling function, with the exception that we set $\lambda = 2$ when $D = 3$ so that random subspacing and CCA can both be employed. This is the default setting taken by, for example, the WEKA RF implementation (Hall et al., 2009). It has the advantage of being very fast, as the training cost subsequently scales logarithmically with the number of features (see Section 7.2). We also

found that it was also an effective choice in terms of empirical predictive accuracy, outperforming the other common choice of $\lambda = \left\lceil \sqrt{D} \right\rceil$. One could of course also tune λ at the expense of training speed, but we note that results in the recent survey of Fernández-Delgado et al. (2014), along we our own comparisons to their results, suggest that there is no noticeable difference between the performance of RF packages that tune hyperparameters and those that do not. We further refer the reader to the study for the effect of λ on RF performance by Bernard et al. (2009).

Though a number of possible stopping criteria exist (Criminisi et al., 2011), the main two used in practice are a maximum tree depth, beyond which all nodes are assigned as leaves (our default is ∞), and a minimum number of contained datapoints for which a node is allowed to split (our default is 2 for classification and 6 for regression). Though not technically a stopping criterion, a common associated parameter (omitted from the algorithm blocks) is a minimum number of datapoints allowed for a generated leaf node (our default is 1 for classification and 3 for regression). As such, only splits that assign at least this many points to a leaf are considered. The rational for these stopping criteria is to try and guard against potential overfitting, particularly when the L is small. Our default settings are again in line with the defaults of the WEKA RF implementation (Hall et al., 2009). We note that the default settings for classification correspond to using no stopping criteria (other than assigning the node to be a leaf when no split is beneficial). The default settings for regression are slightly more conservative, reflecting a greater apparent tendency for decision tree ensembles to overfit for regression problems then classification problems. However, it may that this apparent tendency is due more to the error metrics that are typically used for assessing the two (i.e. mean squared error compared to misclassification rate), rather than any inherent tendencies of the algorithms themselves.

4. Classification Experiments

4.1 Comparison to State-of-the-art Tree Ensembles

To investigate the predictive performance of CCFs, we ran comparison tests against RFs and rotation forest (Rot-For) over a broad variety of datasets. The results show that CCFs significantly outperformed both methods and create a new benchmark in classification accuracy for out-of-the-box decision tree ensembles, despite being a considerably more computationally efficient than rotation forests as discussed in Section 7.2.

To better understand the key factors behind these improvements, we also considered a number of variations of the CCF algorithm. Firstly, we considered using bagging, as per RF, instead of projection bootstrapping. We refer to this algorithm as CCF-Bag.

Secondly, we considered the effect of uniformly sampling the projection matrix, Φ, from the space of valid rotation matrices, instead of using CCA. Here our notion of a uniform distribution over rotations is based on the Haar measure as discussed by Diaconis and Shahshahani (1987) and Blaser and Fryzlewicz (2016). We refer to this approach as random rotation forests (RRFs). To carry out the required rotation matrix sampling, RRFs use the procedure laid out in Blaser and Fryzlewicz (2016) based on a QR decomposition (Householder, 1958) of independent draws from a unit Gaussian. Note that, unlike Blaser and Fryzlewicz (2016), the random rotation in RRFs is carried out at separately each node, as opposed to using a single rotation for the whole tree.

Finally we considered using RRFs with the addition of bagging, referring to this as algorithm as RRF-Bag. RRF-Bag shares similarity with the Forest-RC algorithm of Breiman (2001). A key

difference is that Forest-RC independently samples elements of the projection matrix uniformly in the range $\{-1, +1\}$, biasing the projected features, $U = \mathcal{X}\Phi$, away from the original axis compared with uniformly sampling rotations. Forest-RC also allows sparsity by calculating n_p separate projections, each with a maximum of n_v non-zero values, such that Φ is constructed as a $D \times n_p$ matrix where there are only n_v values are nonzero in each column. We do not consider Forest-RC directly here, but note that previous results suggest that its average performance is similar to RFs.

MATLAB's `TreeBagger` function was used for the RF implementation, while our own MATLAB implementation[4] was used for CCFs, CCF-Bag, RRFs, and RRF-Bag. As per the default parameters given in 3.3.3, each ensemble was built using $L = 500$ trees, $\lambda = \lceil \log_2(D) + 1 \rceil$ (with the exception $\lambda = 2$ when $D = 3$), the entropy impurity criterion, and no stopping criteria other than stopping when no split is beneficial. The choice of the entropy impurity criterion over the Gini impurity criterion and setting $\lambda = \lceil \log_2(D) + 1 \rceil$ rather than the common alternative $\lambda = \left\lceil \sqrt{D} \right\rceil$ was made on the basis that both choices gave improved predictive performance for RFs on average compared with these alternatives. For CCFs, CCF-Bag, RRFs and RRF-Bag, missing values were sampled randomly for each tree as explain in 3.3.2 while for RFs they were dealt with internally by the `TreeBagger` function.

Rotation Forests were implemented in WEKA (Hall et al., 2009) with 500 trees and the default tree options except that we used binary, unpruned, trees and set the minimum number of instances per leaf to 1. In addition to keeping the implementation of rotation forest as consistent as possible with the other algorithms, these settings dominated rotation forests of the same size with the default options over a single cross validation. As recommended by Rodriguez et al (Rodriguez et al., 2006), 1-of-K encoding was used for non-ordinal features for rotation forests (note rotation forests do not then treat them differently to ordinal variables), while missing values were replaced by the mean.

The majority of the 37 datasets considered were taken from the UCI machine learning database[5] (Lichman, 2013) with the exceptions of the *ORL* face recognition dataset (Samaria and Harter, 1994), the *Polyadenylation Signal Prediction* (*polya*) dataset (Li and Liu, 2002), and the artificial spiral dataset from Figure 1. Summaries of the datasets are given in Table 1. Note for the *vowel-c* dataset the sex and identifier for the speaker are included, whereas these are omitted for the *vowel-n* dataset which is otherwise identical. The *wholesale-c* and *wholesale-r* datasets correspond to predicting the *channel* and *region* attributes respectively.

For each dataset, 15 different 10-fold cross-validation tests were performed, results from which are given in Table 2. Table 3 shows a summary of results over all the datasets, giving the number of datasets for which the performance of one dataset was significantly better than another at the 1% level of a Wilcoxon signed rank test. Table 4 shows the average performance ranks of the algorithms. These results show that CCFs performed excellently, comfortably outperforming all the other approaches. The improvement compared to RFs was particularly large, with RFs having on average 1.74 times as many misclassifications as CCFs (ignoring the two datasets where CCFs had perfect accuracy). To give another perspective, if one were using RFs and decided to switch to CCFs, then the number of misclassifications would be reduced by a factor of 28.7% on average for the tested datasets. We emphasise that these are improvements over a generic selection of datasets, rather than being carefully chosen problems. Therefore the magnitude and regularity (33 of 37 datasets had lower error for CCFs than RFs) of the improvements represents a substantial ad-

4. `https://github.com/twgr/ccfs/`
5. `https://archive.ics.uci.edu/ml/datasets.html`

Table 1: Dataset summaries for classification experiments. K = number of classes, N = number of data points, D_c = number of non-ordinal features, D_r = number of ordinal features, and % Miss = percentage of feature values which are missing.

Dataset	K	N	D_c	D_r	% Miss
Balance scale	3	625	0	4	0
Banknote	2	1372	0	4	0
Breast tissue	6	106	0	9	0
Climate crashes	2	360	0	18	0
Fertility	2	100	0	9	0
Heart-SPECT	2	267	0	22	0
Heart-SPECTF	2	267	0	44	0
Hill valley	2	1212	0	100	0
Hill valley noisy	2	1212	0	100	0
ILPD	2	640	0	10	0
Ionosphere	2	351	0	33	0
Iris	3	150	0	4	0
Landsat satellite	2	6435	0	36	0
Letter	26	20000	0	16	0
Libras	15	360	0	90	0
MAGIC	2	19020	0	10	0
Nursery	5	12960	0	8	0
ORL	40	400	0	10304	0
Optical digits	10	5620	0	64	0

Dataset	K	N	D_c	D_r	% Miss
Parkinsons	2	195	0	22	0
Pen digits	10	10992	0	16	0
Polya	2	9255	0	169	0
Seeds	3	210	0	7	0
Skin seg	2	245057	0	3	0
Soybean	19	683	13	22	3.29
Spirals	3	10000	0	2	0
Splice	3	3190	60	0	0
Vehicle	4	846	0	18	0
Vowel-c	11	990	2	10	0
Vowel-n	11	990	0	10	0
Waveform (1)	3	5000	0	21	0
Waveform (2)	3	5000	0	40	0
Wholesale-c	2	440	1	7	0
Wholesale-r	3	440	0	7	0
Wisconsin	2	699	0	9	0.25
Yeast	10	1484	0	8	0
Zoo	7	101	0	16	0

vancement. In Section 7.1 we will show that there are certain circumstances, namely where there is high correlation between the input features, for which the improvements are typically significantly larger than this. For example, on the highly correlated *hill-valley* dataset the error was reduced from 39.16% for RF to 0% for CCFs. However, the results also show that the improvements do not come from better dealing with input correlations alone, with a number of datsets whose inputs are not correlated showing significant improvements. For example, the error for the *balance scale* dataset was roughly halved, despite there being zero correlation between its inputs features.

Elsewhere, the domination of CCFs over CCF-Bag highlights the improvement from using projection bootstrapping instead of bagging, while the good performance on a large variety of datasets demonstrates the robustness and wide ranging applicability of CCFs. The more computationally intensive rotation forest algorithm performed reasonably well, but was still comfortably outperformed by CCFs on average.

Another interesting result was that RRFs delivered significantly better predictive performance than RRF-Bag. Previous approaches using per-node random rotations, such as Forest-RC (Breiman, 2001) and randomer forests (Tomita et al., 2015), have employed bagging in addition to random rotations. These results suggest that this is actually detrimental to the ensemble performance and supports the motivation for projection bootstrapping provided in Section 3.3.1.

Table 2: Mean and standard deviations of percentage of test cases misclassified. Method with best accuracy is shown in bold. • and ○ indicate that CCFs were significantly better and worse respectively at the 1% level of a Wilcoxon signed rank test. The *ORL* and *Polya* datasets could not be run in reasonable time for rotation forest.

Dataset	CCF	RF	Rot-For	CCF-Bag	RRF	RRF-Bag
Balance scale	9.60 ± 3.73	18.66 ± 4.56 •	**7.25 ± 3.38** ○	8.80 ± 3.82 ○	13.53 ± 3.85 •	10.85 ± 3.89 •
Banknote	**0.00 ± 0.00**	0.58 ± 0.64 •	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	0.02 ± 0.12
Breast tissue	28.18 ± 12.00	31.09 ± 12.38 •	28.48 ± 12.36	28.36 ± 11.82	**26.06 ± 11.85** ○	26.24 ± 12.08 ○
Climate crashes	**5.81 ± 4.10**	6.46 ± 4.10 •	6.11 ± 3.94 •	6.56 ± 4.16 •	7.06 ± 4.15 •	7.22 ± 4.15 •
Fertility	13.33 ± 9.67	14.53 ± 9.45 •	**12.40 ± 8.87**	13.53 ± 9.70	12.67 ± 9.10 ○	12.87 ± 9.22
Heart-SPECT	18.59 ± 6.97	18.86 ± 7.25	**17.56 ± 7.29** ○	18.12 ± 7.12	18.17 ± 6.88	18.37 ± 7.06
Heart-SPECTF	18.64 ± 7.36	18.91 ± 7.09	18.57 ± 7.06	**18.25 ± 7.10**	19.26 ± 7.63	19.56 ± 7.73 •
Hill valley	**0.00 ± 0.00**	39.16 ± 4.05 •	6.32 ± 2.65 •	**0.00 ± 0.00**	4.17 ± 1.86 •	6.65 ± 2.28 •
Hill valley noisy	**4.73 ± 1.90**	42.08 ± 4.57 •	11.01 ± 2.80 •	5.24 ± 1.84 •	15.78 ± 3.44 •	18.64 ± 3.88 •
ILPD	27.56 ± 5.08	29.92 ± 5.17 •	29.02 ± 5.25 •	28.20 ± 5.25 •	27.55 ± 4.94	**27.29 ± 4.98**
Ionosphere	4.82 ± 3.44	6.53 ± 3.95 •	5.79 ± 3.56 •	5.50 ± 3.66 •	**4.69 ± 3.43**	5.01 ± 3.55
Iris	2.76 ± 3.95	5.07 ± 5.22 •	4.31 ± 5.19 •	**2.13 ± 3.65** ○	4.53 ± 5.31 •	4.49 ± 5.21 •
Landsat satellite	8.18 ± 1.06	8.03 ± 1.00 ○	**7.75 ± 1.02** ○	8.59 ± 1.05 •	8.17 ± 1.02	8.58 ± 1.02 •
Letter	**2.10 ± 0.33**	3.36 ± 0.39 •	2.43 ± 0.32 •	2.43 ± 0.35 •	2.82 ± 0.37 •	3.29 ± 0.40 •
Libras	9.80 ± 4.68	18.54 ± 5.93 •	**9.48 ± 4.47**	11.26 ± 4.99 •	12.06 ± 4.95 •	13.59 ± 5.10 •
MAGIC	**11.54 ± 0.68**	11.90 ± 0.72 •	12.67 ± 0.71 •	11.69 ± 0.72 •	12.46 ± 0.70 •	12.59 ± 0.74 •
Nursery	0.04 ± 0.07	0.19 ± 0.13 •	**0.03 ± 0.06** ○	0.09 ± 0.10 •	0.10 ± 0.11 •	0.22 ± 0.15 •
ORL	**1.58 ± 2.08**	1.72 ± 2.03	-	2.05 ± 2.17 •	2.62 ± 2.58 •	3.55 ± 3.10 •
Optical digits	**1.26 ± 0.45**	1.59 ± 0.49 •	1.29 ± 0.41	1.37 ± 0.46 •	1.46 ± 0.46 •	1.62 ± 0.49 •
Parkinsons	**5.87 ± 5.17**	9.43 ± 6.17 •	7.19 ± 5.33 •	7.47 ± 6.04 •	6.70 ± 5.27 •	7.43 ± 5.64 •
Pen digits	**0.41 ± 0.19**	0.82 ± 0.28 •	0.48 ± 0.22 •	0.45 ± 0.20 •	0.53 ± 0.23 •	0.63 ± 0.25 •
Polya	**21.02 ± 1.30**	21.20 ± 1.27 •	-	21.15 ± 1.25 •	21.37 ± 1.35 •	21.89 ± 1.29 •
Seeds	**4.60 ± 4.56**	5.78 ± 5.01 •	4.79 ± 4.60	5.46 ± 5.30 •	5.84 ± 4.91 •	6.41 ± 5.03 •
Skin seg	**0.03 ± 0.01**	0.05 ± 0.01 •	0.04 ± 0.01 •	0.03 ± 0.01 •	0.03 ± 0.01 •	0.03 ± 0.01 •
Soybean	**5.22 ± 2.73**	5.50 ± 3.02	5.67 ± 2.91	5.54 ± 2.94 •	5.58 ± 2.75 •	5.71 ± 2.90 •
Spirals	0.27 ± 0.16	1.21 ± 0.33 •	1.01 ± 0.32 •	0.27 ± 0.16	0.26 ± 0.16	**0.26 ± 0.15**
Splice	**3.04 ± 0.88**	3.05 ± 0.93	4.21 ± 1.16 •	3.10 ± 0.87	10.06 ± 1.86 •	11.32 ± 2.03 •
Vehicle	**17.34 ± 4.13**	25.22 ± 4.57 •	21.12 ± 4.27 •	17.36 ± 4.06	22.63 ± 4.19 •	22.95 ± 4.08 •
Vowel-c	0.85 ± 0.89	2.56 ± 1.66 •	0.95 ± 0.94	1.21 ± 1.03 •	**0.69 ± 0.90**	1.06 ± 1.12 •
Vowel-n	1.84 ± 1.30	3.93 ± 1.98 •	1.43 ± 1.19 ○	2.51 ± 1.55 •	**1.10 ± 1.06** ○	1.41 ± 1.10 ○
Waveform (1)	13.52 ± 1.56	15.06 ± 1.69 •	13.53 ± 1.50	13.46 ± 1.58	13.56 ± 1.56	**13.37 ± 1.48** ○
Waveform (2)	13.30 ± 1.68	14.61 ± 1.64 •	13.31 ± 1.64	**13.28 ± 1.61**	13.65 ± 1.72 •	13.81 ± 1.68 •
Wholesale-c	8.58 ± 4.06	**8.15 ± 4.01**	8.44 ± 3.99	8.47 ± 4.03	8.65 ± 4.17	8.45 ± 4.01
Wholesale-r	30.95 ± 5.84	28.97 ± 6.10 ○	**28.18 ± 6.19** ○	28.80 ± 5.89 ○	32.29 ± 5.91 •	29.62 ± 5.89 ○
Wisconsin cancer	3.23 ± 2.02	3.54 ± 2.17 •	**2.87 ± 1.82** ○	3.02 ± 1.90 ○	3.03 ± 1.97 ○	2.91 ± 1.83 ○
Yeast	38.47 ± 4.04	37.89 ± 4.22 ○	**37.21 ± 4.23** ○	37.27 ± 3.94 ○	38.41 ± 4.15	37.72 ± 4.10 ○
Zoo	**3.33 ± 5.75**	5.13 ± 6.73 •	5.47 ± 6.71 •	3.60 ± 5.83	4.67 ± 6.72 •	4.67 ± 6.72 •

Table 3: Number of victories column vs row at 1% significance level of Wilcoxon signed rank test for classification experiments (37 total). CCF victories and losses are given in blue and red respectively.

	CCF	RF	Rotation Forest	CCF-Bag	RRF	RRF-Bag
CCF	-	**3**	**8**	**5**	**4**	**6**
RF	**28**	-	27	26	24	21
Rotation Forest	**15**	2	-	10	10	8
CCF-Bag	**19**	1	11	-	5	5
RRF	**22**	8	15	18	-	4
RRF-Bag	**24**	9	17	22	20	-

Table 4: Mean rank across datasets of mean test misclassification rate. Lower rank corresponds to better performance (average rank $= 3.5$). *ORL* and *Polya* datasets are omitted from comparison to avoid biasing against rotation forests.

Algorithm	Mean Rank
CCF	2.43
CCF-Bag	2.91
Rotation Forest	2.96
RRF	3.60
RRF-Bag	4.13
RF	4.97

Table 5: Mean and standard deviations of percentage of test cases misclassified for CCF and algorithms from Zhang and Suganthan (2014). All results are from an ensemble of 100 trees. Method with best accuracy is shown in bold. • and ○ indicate that CCFs were significantly better and worse respectively at the 1% level of a t-test.

Dataset	CCF	PCA-RF	LDA-RF	RF-ensemble
Balance scale	**9.70 ± 3.57**	12.41 ± 3.53 •	11.46 ± 3.44 •	11.91 ± 3.41 •
Breast tissue	27.82 ± 11.79	**26.80 ± 13.39**	27.64 ± 13.01	28.00 ± 13.93
Climate crashes	5.78 ± 3.83	8.81 ± 3.38 •	7.70 ± 3.50 •	**2.72 ± 3.42** ○
Iris	**2.58 ± 3.92**	4.53 ± 4.84 •	4.63 ± 5.06 •	4.61 ± 5.21 •
Libras	**9.72 ± 4.75**	12.24 ± 5.57 •	15.13 ± 5.63 •	14.10 ± 5.69 •
Nursery	**0.05 ± 0.07**	0.29 ± 0.15 •	0.19 ± 0.13 •	0.26 ± 0.18 •
Parkinsons	**6.57 ± 5.53**	8.23 ± 6.15 •	9.89 ± 6.93 •	8.77 ± 6.38 •
Seeds	**4.89 ± 4.55**	6.13 ± 5.21	6.04 ± 4.50	6.06 ± 4.80
Splice	**3.30 ± 0.97**	9.04 ± 1.62 •	7.90 ± 1.79 •	5.78 ± 1.36 •
Vehicle	**17.44 ± 4.19**	21.33 ± 4.47 •	20.66 ± 4.60 •	21.77 ± 4.45 •
Waveform (1)	**13.53 ± 1.57**	13.81 ± 1.53	14.12 ± 1.50 •	13.78 ± 1.51
Waveform (2)	13.76 ± 1.65	**13.72 ± 1.62**	14.12 ± 1.54	13.79 ± 1.60
Wisconsin	3.31 ± 2.11	2.92 ± 2.01	3.02 ± 1.91	**2.84 ± 1.91**
Yeast	38.79 ± 4.13	37.82 ± 3.65	**37.77 ± 3.69**	37.92 ± 3.65

4.1.1 COMPARISON TO RANDOM FORESTS WITH ENSEMBLE OF FEATURE SPACES

We next make comparisons to the PCA-RF, LDA-RF, and RF-ensemble methods introduced by Zhang and Suganthan (2014). As Zhang and Suganthan (2014) performed the same cross validation tests on 14 of the common datasets, we make direct comparisons to their quoted results. Taking the first 100 trees from our experiments to match the ensemble sizes gives the comparisons provided in Table 5. We see that CCFs consistently outperform all three methods, giving win/draw/loss ratios at the 1% significance level of a t-test for CCFs of 8/6/0, 9/5/0, and 7/6/1 compared to PCA-RF, LDA-RF and RF-ensemble respectively. On average PCA-RF, LDA-RF, and RF-ensemble had 1.66, 1.51 and 1.49 times as many misclassifications as CCFs respectively. We reiterate that, other than PCA-RF, these methods do not extend to the regression and multiple output cases like CCFs and the other approaches we consider do.

4.2 Comparison to Other Classifiers

In order to provide comparison to a wider array of classifiers, we also tested CCFs using the experimental setup of Fernández-Delgado et al. (2014) from their recent survey of 179 classifiers applied to 121 datasets. We used the same partitions which were a mix of 4-fold cross validations and predefined train / test splits.[6] We omitted datasets containing non-ordinal features on the basis that they pre-processed such features by treating the category indexes as numeric features, which is substantially different to our recommended procedure (see Section 3.3.2). For consistency with their approach, we fixed the missing features values (to the feature mean), rather than using the procedure laid out in Section 3.3.2. As a check that tests were run correctly, we also ran MATLAB's TreeBagger algorithm, with 500 trees and $\lambda = \lceil \log_2(D) + 1 \rceil$, which as expected gave performance on par with the very similar rforest_R algorithm. An exception to this was the *image-segmentation* dataset for which we were unable to replicate the rforest_R results to within any plausible level of variability. We therefore omitted this dataset from the results. In total 82 datasets were compared, for which summaries along with detailed results are available in Appendix E, while a summary of the results for the top 20 performing classifiers (based on average accuracy rank) is given in Table 6. We note that our quoted results for the competing methods differs from those of the original paper due to using only a subset of the original datasets (averages were reprocessed using the individual dataset scores provided by Fernández-Delgado et al. (2014)). We also note that the relatively poor performance of rotation forests that is quoted most likely stems from the fact that they set $L = 10$ for rotation forests and $L = 500$ for most of the RF implementations, constituting an arguably unfair comparison.

Whereas Fernández-Delgado et al. (2014) included an, often intensive, parameter tuning step in their tests, we used CCFs in an out-of-the-box fashion, taking the same parameters as in Section 3.3.3. Despite this, CCFs outperformed all other tested classifiers based on every calculated performance metric. This superior performance is particularly impressive in light of the observation by Wainberg et al. (2016) that the method Fernández-Delgado et al. (2014) use for the parameter tuning positively biases the estimated predictive accuracy and κ score (Carletta, 1996) (an thus negatively biases the error). Namely, their parameter tuning step makes use of the full dataset (train

6. Partitions and results from Fernández-Delgado et al. (2014) are available at http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/

7. Note that these are *not* the number significant victories / losses as quoted elsewhere due to the small number of folds considered.

Table 6: Comparison of top 20 performing classifiers on 82 UCI datasets. R is the mean rank over all 181 classifiers according to error rate; E is the mean error rate (%); κ is the mean Cohen's κ (unlike others, higher is better); R_{CCF}, E_{CCF} and κ_{CCF} are the respective values for CCFs on the datasets where the competing classifier successfully ran (note CCFs successfully ran on all datasets); N_v and N_l are the number of datasets where the CCFs κ was higher and lower than the classifier respectively;[7]and p is the p-value for whether the CCFs κ mean is higher using a Wilcoxon signed rank test. Classifier types: SVM = support vector machine, NNET = neural net, RF = random forest, Bag = bagging, and BST = boosting. Results for CCF, rForest_R, TreeBagger, and RotationForest are all statistically valid, while results for the other datasets are potentially inflated, such that these classifiers may actually perform worse than quoted, see main text for details. For further details on classifiers see Fernández-Delgado et al. (2014).

Classifier	R	R_{CCF}	E	E_{CCF}	κ	κ_{CCF}	N_v	N_l	p
CCF	**28.87**	-	**14.08**	-	**70.67**	-	-	-	-
svmPoly (SVM)	31.53	30.57	15.73	14.27	65.10	69.61	54	25	2.3e-4
svmRadialCost (SVM)	31.84	30.57	15.33	14.27	66.55	69.61	43	36	0.11
svm_C (SVM)	32	29.21	15.67	14.18	67.65	70.49	46	32	0.18
elm_kernel (NNET)	32.19	30.14	15.20	14.54	69.01	69.75	42	36	0.16
parRF (RF)	33.03	28.87	15.54	14.08	67.73	70.67	52	27	0.014
svmRadial (SVM)	33.77	30.57	15.68	14.27	65.88	69.61	50	28	1.6e-3
rf_caret (RF)	34.48	29.21	15.56	14.18	67.67	70.49	54	23	6.3e-4
rforest_R (RF)	40.70	29.21	15.82	14.18	66.67	70.49	57	20	2e-5
TreeBagger (RF)	40.91	28.87	15.75	14.08	67.51	70.67	55	23	3.5e-5
Bag_LibSVM (Bag)	42.28	29.48	16.65	14.25	58.13	70.27	70	12	3.2e-12
C50_t (BST)	42.61	28.87	16.85	14.08	66.11	70.67	57	22	4.0e-4
nnet_t (NNET)	42.87	28.87	18.74	14.08	64.72	70.67	54	26	1.5e-3
avNNet_t (NNET)	43.26	28.87	18.77	14.08	64.88	70.67	50	29	1.0e-3
RotationForest	44.62	28.87	16.64	14.08	65.34	70.67	64	15	7.1e-9
pcaNNet_t (NNET)	45.86	28.87	19.28	14.08	63.83	70.67	54	25	1.5e-4
mlp_t (NNET)	46.06	28.87	17.38	14.08	66.75	70.67	54	26	1.6e-3
LibSVM (SVM)	46.50	28.87	16.65	14.08	63.80	70.67	57	21	2.9e-6
MB_LibSVM (BST)	46.90	28.87	16.82	14.25	64.47	70.27	61	17	1.8e-6
Regularized RF_t (RF)	49.56	28.87	16.71	14.18	66.30	70.49	59	20	1.1e-5

and test), such that the quoted performance on the test set is inflated. Crucially, this does not affect methods that do not use a parameter tuning step - specifically CCFs, rForest_R, TreeBagger, and rotation forests from those in shown in Table 6. As the results for others are all biased towards better performance, improvements of CCFs over these methods should always be at least as large as those quoted and never less. Similarly for cases were the improvement for CCFs is shown to be statistically significant, then this significance result still holds.

As also noted by Wainberg et al. (2016), the procedure of Fernández-Delgado et al. (2014) was further biased in its consideration of failed runs as the quoted ranks, average accuracies, and κ scores omit failed runs in the quoted average for each classifier. This causes bias because, for example, a classifier that failed to run a dataset where the best attainable error rate was relatively high will have its average error negatively biased. We note that experiments for the CCFs and TreeBagger runs had no such failed runs. This bias does not effect the significance test comparisons or number of victories / losses compared with CCFs as these comparisons are based only on the datasets were the competing classifier successfully ran on all folds. Note also that none of the top

20 classifiers had more than 4 failures so the effects of failed runs was relatively small. However, in the interest of correctly comparing the mean rank, mean error, and mean κ between CCFs and the alternatives, we also provide the average metrics for CCFs on the subset of datasets where each classifier successfully run (see Table 6). These show that CCFs outperformed all of the other classifiers on all three metrics. Similarly, we found that for each classifier, the number of times CCFs outperformed that classifier was larger than the number of times it performed worse.

5. Regression Experiments

We now investigate the predictive performance of CCFs for regression. We start with an illustrative example to show the benefits of CCFs over axis aligned approaches. Namely, we consider the six hump camel function (Molga and Smutnicki, 2005)

$$f(x_1, x_2) = \left(4 - 2.1x_2^2 + \frac{x_2^4}{3}\right) x_2^2 + x_1 x_2 + (-4 + 4x_1^2)x_1^2 \tag{14}$$

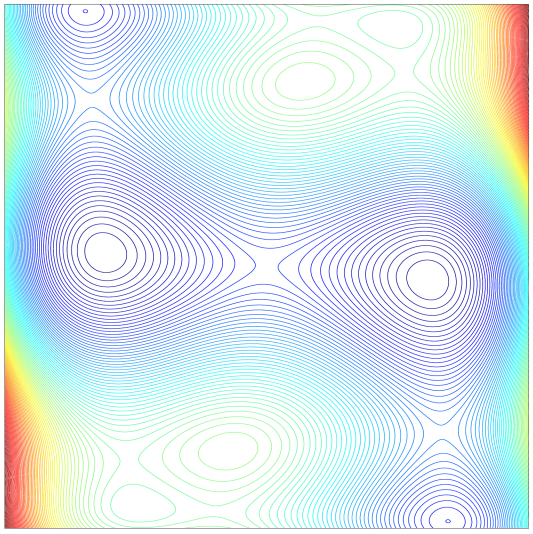
in the range $x_1 \in [-1.15, 1.15]$, $x_2 \in [-1.75, 1.75]$. Figure 4 shows the contour plots resulting from training a CCF and RF on a random sample of points in this range. These show that the RF predictions fail to catch the true contours of the function, with the RF contours noticeably aligned with the axis to the detriment of the regressor. The CCF on the other hand is able to accurately capture the regression surface.

We next consider a more rigorous comparison of CCFs to RFs, rotation forests, CCF-Bag, RRFs, and RRF-bag. Parameters were again set as per Section 3.3.3, meaning that we used the mean squared error criterion and set the minimum number of points at a leaf node to 3 for all methods. All other parameters and the test procedure (i.e. 15 separate $10-$fold cross-validation tests) were as per the classification experiments in Section 4.1. As in Pardo et al. (2013), we used the 61 regression datasets from the WEKA dataset collection[8] (Hall et al., 2009), 30 of which were assembled by Luis Torgo.[9] Summary information about each dataset is provided in Table 7, the full results are given in Table 8, summaries for the significance test results are shown in Table 9, and the mean ranks are given in Table 10.

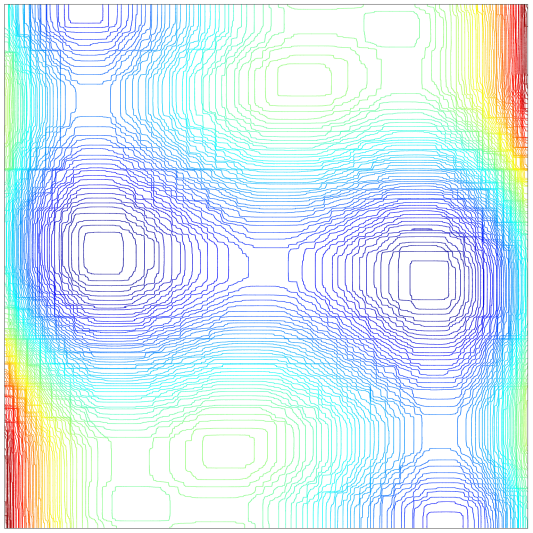
These results show that, as with the classification experiments, CCFs outperformed all the other methods. Interestingly, the performance of RFs was noticeably improved relative to the other approaches. Though still comfortably outperformed by CCFs and CCF-Bag, RFs performed similarly to rotation forests and substantially better than the random rotation approaches for these problems. These results were confirmed by repeating the RF tests using the same code base as for CCFs, producing very similar results (this check was also performed for the classification experiments). One possible reason for this difference is that it stems from differences in the nature of the error criteria used for the two methods - misclassification rate and mean squared error respectively - rather than inherent differences between classification and regression. For example, the latter error criteria is more influenced by rare large errors.

8. http://www.cs.waikato.ac.nz/ml/weka/datasets.html
9. http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

(a) Ground Truth



(b) RF prediction



(c) CCF prediction

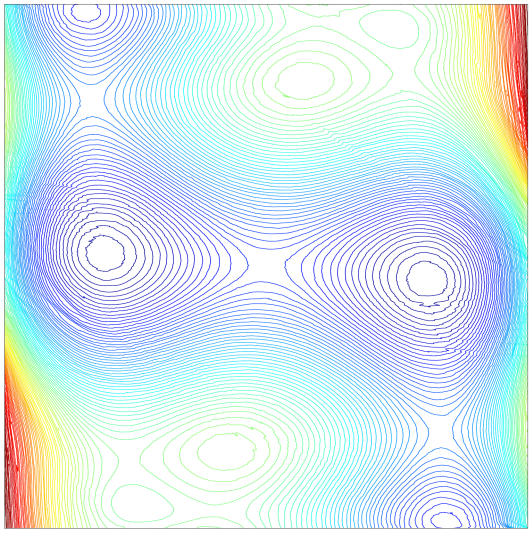


Figure 4: Contour plots for regression of the six hump camel function (14) in the range $x_1 \in [-1.15, 1.15]$, $x_2 \in [-1.75, 1.75]$. a) shows the ground truth, corresponding to a contour plot for a 1000×1000 grid. The training dataset was formed of 20000 random samples without replacement from these 10^6 points. The trained regressor was then used to predict all 10^6 points in the grid to construct the contour plots shown in b) and c). Mean squared errors for the 9.8×10^5 points not used in training were 8.45×10^{-4} and 3.40×10^{-4} for the RF and CCF respectively.

Table 7: Dataset summaries for regression experiments. N = number of data points, D_c = number of non-ordinal features, D_r = number of ordinal features, and % Miss = percentage of feature values which are missing.

Dataset	N	D_c	D_r	% Miss
2D planes	40768	0	10	0
Abalone	4177	1	7	0
Ailerons	13750	0	40	0
Auto 93	93	3	19	0.68
Auto horsepower	203	8	17	1.04
Auto MPG	398	1	6	0.22
Auto price	159	0	15	0
Bank 32NH	8192	0	32	0
Bank 8FM	8192	0	8	0
Basketball	96	0	4	0
Body fat	252	0	14	0
Bolts	40	0	7	0
Breast tumor	286	6	3	0
CA housing	20640	0	8	0
Cholesterol	303	3	10	0.10
Cleveland	303	3	10	0.10
Cloud	108	2	4	0
CPU	209	1	6	0
CPU act	8192	0	21	0
CPU small	8192	0	12	0
Delta ailerons	7129	0	5	0
Delta elevators	9517	0	6	0
Detroit	13	0	13	0
Diabetes	43	0	2	0
Echocardiogram	130	0	9	8.29
Elevators	16599	0	18	0
Electricity usage	55	0	2	0
Fish catch	158	1	6	7.87
Friedman	40768	0	10	0
Fruit fly	125	1	3	0
Gas consumption	27	0	4	0

Dataset	N	D_c	D_r	% Miss
House 16H	22784	0	16	0
House 8L	22784	0	8	0
Housing	506	0	13	0
Hungarian	294	3	10	13.47
Kinematics	8192	0	8	0
Longley	16	0	6	0
Low birth weight	189	1	8	0
Machine CPU	209	0	6	0
MBA grade	61	0	2	0
Meta	528	2	19	4.55
MV	40768	3	7	0
PBC	418	1	17	16.47
Pharynx	195	2	10	0.09
Pole	15000	0	48	0
Pollution	60	0	15	0
Puma 32H	8192	0	32	0
Puma 8NH	8192	0	8	0
PW Linear	200	0	10	0
Pyrimidines	74	0	27	0
Quake	2178	0	3	0
SCHL vote	37	0	5	0
Sensory	576	0	11	0
Servo	167	4	0	0
Sleep	58	0	7	1.97
Stock	950	0	9	0
Strike	625	1	5	0
Triazines	186	0	60	0
Veteran	137	1	6	0
Vineyard	52	0	3	0
Wisconsin	194	0	32	0

Table 8: Mean squared error as a percentage of the data variance. Conventions as per Table 2.

Dataset	CCF	RF	Rot For	CCF-Bag	RRF	RRF-Bag
2D planes	6.37 ± 0.12	**5.57 ± 0.10** ○	5.87 ± 0.11 ○	6.09 ± 0.11 ○	6.57 ± 0.12 ●	6.08 ± 0.11 ○
Abalone	40.83 ± 4.91	43.88 ± 5.39 ●	42.08 ± 5.11 ●	**40.72 ± 4.86** ○	43.62 ± 5.18 ●	43.24 ± 5.16 ●
Ailerons	17.46 ± 1.15	18.60 ± 1.24 ●	**17.23 ± 1.10** ○	17.48 ± 1.14	19.03 ± 1.30 ●	19.74 ± 1.35 ●
Auto 93	**27.31 ± 32.94**	34.42 ± 38.42 ●	34.92 ± 34.50 ●	28.15 ± 34.09 ●	30.46 ± 35.70 ●	34.41 ± 39.39 ●
Auto horsepower	**10.89 ± 14.89**	15.37 ± 18.44 ●	12.11 ± 15.11 ●	11.23 ± 15.10 ●	13.21 ± 16.42 ●	15.00 ± 17.55 ●
Auto MPG	11.53 ± 4.56	12.61 ± 5.44 ●	11.85 ± 4.91 ●	11.68 ± 4.53 ●	**11.44 ± 4.73**	11.89 ± 4.82 ●
Auto price	**10.91 ± 7.57**	12.20 ± 9.60 ●	11.83 ± 9.62 ●	11.95 ± 8.57 ●	12.75 ± 11.81 ●	14.59 ± 13.41 ●
Bank 32NH	48.67 ± 3.83	50.35 ± 3.94 ●	**48.61 ± 3.88**	48.89 ± 3.82 ●	52.50 ± 4.02 ●	53.59 ± 4.05 ●
Bank 8FM	**3.70 ± 0.25**	4.19 ± 0.29 ●	4.38 ± 0.27 ●	3.74 ± 0.25 ●	4.58 ± 0.28 ●	4.86 ± 0.30 ●
Basketball	72.77 ± 31.73	69.98 ± 34.67 ○	**66.38 ± 30.50** ○	69.77 ± 31.60 ○	73.88 ± 31.73 ●	69.79 ± 31.75 ○
Body fat	**4.73 ± 3.26**	5.41 ± 3.47 ●	4.95 ± 3.51	5.36 ± 3.34 ●	10.06 ± 4.31 ●	11.55 ± 4.75 ●
Bolts	**8.62 ± 8.05**	10.57 ± 7.58 ●	17.45 ± 13.24 ●	10.61 ± 9.66 ●	22.32 ± 17.85 ●	24.61 ± 19.00 ●
Breast tumor	108.1 ± 25.03	99.96 ± 23.73 ○	**98.19 ± 23.15** ○	104.8 ± 24.13 ○	115.0 ± 26.53 ●	105.3 ± 24.86 ○
CA housing	19.46 ± 1.13	**17.55 ± 1.08** ○	20.29 ± 1.17 ●	19.65 ± 1.13 ●	21.33 ± 1.19 ●	21.90 ± 1.20 ●
Cholesterol	99.71 ± 43.48	99.56 ± 43.08	**97.62 ± 41.95** ○	99.25 ± 43.31	101.1 ± 44.32 ●	98.53 ± 43.85 ○
Cleveland	48.15 ± 14.20	49.64 ± 14.68 ●	48.31 ± 15.15	**47.80 ± 13.85** ○	48.95 ± 14.15 ●	48.71 ± 13.89 ●
Cloud	22.09 ± 22.59	**20.65 ± 23.51** ○	26.19 ± 28.85 ●	21.65 ± 23.65 ○	24.08 ± 27.58 ●	25.98 ± 29.25 ●
CPU	**8.07 ± 20.08**	13.93 ± 28.85 ●	12.95 ± 42.08	10.28 ± 22.11 ●	12.06 ± 26.98 ●	17.20 ± 32.16 ●
CPU act	**1.66 ± 0.14**	1.75 ± 0.19 ●	1.77 ± 0.23 ●	1.71 ± 0.15 ●	2.63 ± 0.43 ●	2.97 ± 0.52 ●
CPU small	**2.25 ± 0.19**	2.30 ± 0.24 ●	2.45 ± 0.26 ●	2.29 ± 0.19 ●	2.45 ± 0.22 ●	2.59 ± 0.24 ●
Delta ailerons	28.70 ± 2.44	**28.30 ± 2.64** ○	30.31 ± 3.00 ●	28.53 ± 2.45 ○	28.99 ± 2.64 ●	28.76 ± 2.67
Delta elevators	36.40 ± 2.20	36.74 ± 2.28 ●	**35.41 ± 2.24** ○	36.04 ± 2.19 ○	36.75 ± 2.25 ●	36.01 ± 2.23 ○
Detroit	39.20 ± 78.01	43.25 ± 75.49 ●	102.5 ± 125.6 ●	44.25 ± 90.56 ●	**35.72 ± 68.25** ○	41.68 ± 82.03 ●
Diabetes	70.84 ± 45.65	**64.93 ± 39.75** ○	87.56 ± 54.48 ●	71.01 ± 45.74	67.54 ± 43.10 ○	67.77 ± 43.06 ○
Echocardiogram	56.94 ± 19.42	54.11 ± 18.47 ○	**52.09 ± 17.91** ○	56.53 ± 18.79	58.00 ± 19.32 ●	56.82 ± 18.33
Elevators	**10.34 ± 1.01**	18.39 ± 1.76 ●	14.81 ± 1.45 ●	10.46 ± 1.03 ●	20.03 ± 2.06 ●	21.57 ± 2.22 ●
Electricity usage	21.29 ± 13.25	22.25 ± 14.09	28.82 ± 22.79 ●	**21.23 ± 13.34**	23.54 ± 16.33	23.55 ± 16.40
Fish catch	**2.17 ± 2.35**	6.15 ± 7.20 ●	5.40 ± 10.14 ●	2.55 ± 2.66 ●	3.24 ± 3.33 ●	4.59 ± 4.83 ●
Friedman	9.15 ± 0.19	**6.60 ± 0.17** ○	7.36 ± 0.21 ○	9.28 ± 0.20 ●	8.66 ± 0.22 ○	9.40 ± 0.24 ●
Fruit fly	128.6 ± 66.66	124.0 ± 67.16 ○	**108.0 ± 65.43** ○	123.4 ± 65.91 ○	136.6 ± 68.94 ●	126.2 ± 66.98 ○
Gas consumption	**4.02 ± 4.68**	6.71 ± 7.05 ●	10.08 ± 6.80 ●	4.69 ± 6.90	5.83 ± 5.37 ●	5.81 ± 7.25 ●
House 16H	36.57 ± 4.47	**35.74 ± 4.49** ○	40.51 ± 4.88 ●	37.04 ± 4.42 ●	40.30 ± 4.68 ●	41.98 ± 4.83 ●
House 8L	**29.96 ± 3.11**	30.53 ± 3.13 ●	32.94 ± 3.41 ●	30.08 ± 3.10 ●	31.89 ± 3.26 ●	32.28 ± 3.29 ●
Housing	**12.89 ± 8.47**	13.51 ± 9.04 ●	14.86 ± 9.56 ●	13.49 ± 8.68 ●	13.85 ± 8.93 ●	15.42 ± 9.63 ●
Hungarian	50.94 ± 14.55	55.42 ± 15.34 ●	53.94 ± 16.01 ●	**50.22 ± 14.54** ○	51.04 ± 14.30	51.06 ± 13.76
Kinematics	20.33 ± 1.05	27.97 ± 1.46 ●	21.33 ± 1.16 ●	20.92 ± 1.09 ●	**18.81 ± 1.00** ○	20.98 ± 1.10 ●
Longley	**6.61 ± 7.56**	14.75 ± 12.00 ●	16.67 ± 17.19 ●	8.77 ± 9.75 ●	9.57 ± 8.02 ●	14.32 ± 10.72 ●
Low birth weight	40.09 ± 13.68	39.63 ± 12.92	**37.97 ± 13.03** ○	38.89 ± 13.27 ○	45.71 ± 16.38 ●	43.58 ± 15.95 ●
Machine CPU	**12.80 ± 23.68**	16.69 ± 30.55 ●	23.39 ± 37.67 ●	14.79 ± 24.89 ●	13.10 ± 26.48	17.53 ± 30.16 ●
MBA grade	106.1 ± 69.36	106.0 ± 68.30	**91.00 ± 66.94** ○	107.0 ± 69.62 ●	106.5 ± 68.01	107.3 ± 68.51
Meta	**93.29 ± 159.3**	95.34 ± 165.4 ●	100.3 ± 185.3	95.28 ± 165.6	97.12 ± 175.6	99.64 ± 184.7
MV	**0.00 ± 0.00**	0.02 ± 0.01 ●	0.02 ± 0.00 ●	0.00 ± 0.00 ●	0.29 ± 0.01 ●	0.36 ± 0.02 ●
PBC	62.81 ± 12.95	65.11 ± 13.09 ●	**60.50 ± 13.39** ○	62.45 ± 12.79 ○	64.24 ± 13.04 ●	65.00 ± 13.14 ●
Pharynx	36.35 ± 16.41	37.24 ± 17.90 ●	**32.47 ± 17.89** ○	37.83 ± 16.34 ●	46.21 ± 19.35 ●	49.02 ± 19.39 ●
Pole	2.43 ± 0.21	2.78 ± 0.22 ●	**1.27 ± 0.12** ○	2.65 ± 0.23 ●	2.25 ± 0.21 ○	2.75 ± 0.23 ●
Pollution	**43.35 ± 30.47**	48.87 ± 34.14 ●	47.37 ± 31.65 ●	43.75 ± 30.44	47.81 ± 32.00 ●	50.93 ± 34.02 ●
Puma 32H	65.10 ± 3.48	26.07 ± 1.76 ○	**14.59 ± 0.93** ○	67.43 ± 3.53 ●	66.53 ± 3.19 ●	69.08 ± 3.30 ●
Puma 8NH	32.68 ± 1.77	32.64 ± 1.81	33.18 ± 1.78 ●	**32.60 ± 1.76** ○	34.93 ± 1.86 ●	35.59 ± 1.83 ●
PW Linear	15.33 ± 5.61	16.66 ± 6.09 ●	**15.19 ± 5.55**	15.86 ± 5.83 ●	21.26 ± 8.34 ●	24.02 ± 9.18 ●
Pyrimidines	58.21 ± 113.2	63.96 ± 125.5 ●	69.33 ± 115.6 ●	59.33 ± 115.8 ●	**57.73 ± 113.2**	60.47 ± 112.8 ●
Quake	106.1 ± 11.97	105.2 ± 11.57 ○	**99.75 ± 11.67** ○	103.0 ± 11.51 ○	109.6 ± 12.23 ●	104.5 ± 11.60 ○
SCHL vote	78.31 ± 141.5	**58.17 ± 125.3** ○	93.58 ± 179.9	76.87 ± 140.8 ○	75.34 ± 137.8 ○	79.09 ± 147.0
Sensory	77.27 ± 12.74	**70.44 ± 11.72** ○	71.99 ± 11.75 ○	77.80 ± 12.83 ●	77.44 ± 12.77	78.74 ± 13.20 ●
Servo	**10.93 ± 14.87**	13.85 ± 18.54 ●	14.75 ± 20.91 ●	12.34 ± 14.96 ●	18.54 ± 19.51 ●	23.10 ± 20.77 ●
Sleep	48.24 ± 23.37	56.58 ± 24.68 ●	53.74 ± 25.09 ●	**48.00 ± 22.71**	48.21 ± 24.02	48.82 ± 22.74
Stock	1.11 ± 0.22	1.35 ± 0.34 ●	1.27 ± 0.26 ●	1.25 ± 0.25 ●	**1.06 ± 0.23** ○	1.27 ± 0.26 ●
Strike	87.35 ± 100.0	**84.22 ± 99.55** ○	90.00 ± 102.2 ●	87.84 ± 99.23	87.19 ± 101.0	85.03 ± 100.8 ○
Triazines	71.99 ± 36.69	71.01 ± 35.37	**70.25 ± 34.49**	71.56 ± 36.57	72.44 ± 36.78	72.87 ± 37.27 ●
Veteran	80.39 ± 77.45	88.10 ± 88.31 ●	84.10 ± 89.71	78.38 ± 73.80 ○	79.03 ± 78.11 ○	**78.35 ± 77.85** ○
Vineyard	34.86 ± 27.98	36.99 ± 33.40	46.64 ± 32.90 ●	35.52 ± 28.49 ●	**34.77 ± 28.92**	35.50 ± 31.08
Wisconsin	87.82 ± 24.18	92.91 ± 24.91 ●	90.38 ± 24.95 ●	**87.33 ± 23.96** ○	89.87 ± 25.33 ●	88.52 ± 24.80

Table 9: Number of victories column vs row at 1% significance level of Wilcoxon signed rank test for regression experiments (61 total). CCF victories and losses are given in blue and red respectively.

	CCF	RF	Rotation Forest	CCF-Bag	RRF	RRF-Bag
CCF	-	**16**	**17**	**17**	**8**	**10**
RF	**38**	-	28	31	17	13
Rotation Forest	**35**	24	-	31	22	16
CCF-Bag	**33**	13	18	-	11	3
RRF	**41**	31	32	40	-	13
RRF-Bag	**41**	33	36	41	38	-

Table 10: Mean rank across datasets of test mean squared error. Lower rank corresponds to better performance.

Algorithm	Mean Rank
CCF	2.43
CCF-Bag	2.67
Rotation Forest	3.56
RF	3.64
RRF	4.02
RRF-Bag	4.69

6. Multiple Outputs

We now demonstrate how CCFs can be used for multiple output problems such as multivariate regression, where we wish to regress multiple outputs simultaneously, and multi-output classification, where we wish to learn a single estimator to predict multiple classification tasks. We note that multi-label classification, where we wish to predict a number of classes which are not mutually exclusive, can be thought of as a special case of multi-output classification were all of the classification tasks are binary. In principle, CCFs can also handle problems with arbitrary combinations of classification and regression outputs in the same way as RFs (see e.g. Glocker et al. (2012); Linusson (2013)), but we will not actively consider this case further here.

Various multiple-output axis-aligned decision trees and forests have previously been developed in the literature (Zhang, 1998; De'Ath, 2002; Geurts et al., 2006b; Segal and Xiao, 2011; Glocker et al., 2012; Faddoul et al., 2012; Linusson, 2013). There are three key ways these approaches tend to vary from single axis aligned approaches: the calculation of split gains which must now assess the utility of the split to multiple different tasks simultaneously, the leaf models which must now predict multiple outputs, and the manner in which the individual tree predictions are combined.

In the simplest case, the leafs have independent models for each of the outputs and the ensemble prediction for each output is the average over trees of the corresponding leaf predictions in the same manner as the single output case. Improvements on this approach have been suggested. For example, Geurts et al. (2006b) introduce the idea of kernelizing the decision tree outputs to ensure the ensemble prediction respects the structure of the output data. As these ideas apply equally

Table 11: Errors for multiple output tasks averaged over outputs (misclassification rate for classification, mean squared error for regression). Method with best accuracy is shown in bold. • and ◦ indicate that CCFs were significantly better and worse respectively at the 1% level of a Wilcoxon signed rank test. Type $\in$ {MVR,MOC} gives the problem type corresponding to multivariate regression and multi-output classification respectively; N_{out} = number of targets; N = number of data points; D_c = number of non-ordinal features; and D_r = number of ordinal features. Note there was no missing data for any of these datasets

Dataset	Type	N_{out}	N	D_c	D_r	CCF	RF		RRF	
Andro	MVR	6	49	0	30	35.55 ± 16.07	**23.08 ± 11.51**	◦	29.93 ± 14.22	◦
EDM	MVR	2	154	0	16	**42.32 ± 13.67**	42.52 ± 13.56		42.95 ± 13.91	•
ENB	MVR	2	768	0	8	**1.70 ± 0.42**	2.19 ± 0.46	•	1.86 ± 0.42	•
Jura	MVR	3	359	0	15	**33.97 ± 12.18**	35.69 ± 12.39	•	35.36 ± 13.23	•
Slump	MVR	3	103	0	7	**40.00 ± 16.50**	49.53 ± 20.94	•	46.43 ± 19.01	•
Emotions	MOC	6	593	0	72	**17.85 ± 1.93**	18.15 ± 1.99	•	17.99 ± 1.98	
Flags	MOC	4	194	3	20	60.41 ± 6.99	**58.12 ± 7.30**	◦	60.61 ± 6.93	
Solar Flare	MOC	3	1066	3	7	8.25 ± 1.59	8.27 ± 1.55		**8.21 ± 1.56**	

well to RFs and CCFs, we leave such extensions to future work and focus on the aforementioned independent leaf model case in the interest of simplicity. Using this approach, we note that if we define each y_n in the multi-output case to be the concatenation of the different outputs (i.e. such that we concatenate columns of Y), then the definition for the leaf model given in (5) and the combination of tree predictions given in (6) are unchanged. Furthermore, as the partitioning procedure given in (3) does not change, the CCF prediction process is unchanged from the single output case and the training process varies only in the split selection procedure.

Choosing a split for a multi-output decision tree requires a balancing of the utility of the split to each of the different output tasks. Two common approaches are to use the same split criteria for individual outputs and then either average or maximum gain across each of the outputs (Faddoul et al., 2012). We consider the former and note that this is equivalent to taking the joint information gain and reduction in multivariate mean squared error for our default choices of impurity criterion in classification and regression cases respectively. Specifically we have that the impurity criterion for multi-output classification is the same as (12) (except that the p_k now sum to the number of classification tasks be considered) and the impurity criterion for mutlivariate regression is

$$g\left(Y_{(\omega_j,:)}\right) = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{\sigma_m}\left(\frac{1}{N_j}\sum_{n\in\omega_j}Y_{(n,m)}^2 - \left(\frac{1}{N_j}\sum_{n\in\omega_j}Y_{(n,m)}\right)^2\right) \tag{15}$$

where M is the number of outputs and σ_m is the standard deviation of $Y_{(:,m)}$. The rational for the $\frac{1}{\sigma_m}$ term is to ensure invariance to the scaling of the outputs. In practice this is done by scaling each column of Y to have unit variance as a preprocessing step such that each $\sigma_m = 1$. If one wishes to give preference to a particular output in the estimation (e.g. because prediction of the parameter is more important or more difficult), then one can also use a weighted average across the split gains, i.e. we can replace the $\frac{1}{\sigma_m}$ term in (15) with $\frac{w_m}{\sigma_m}$ where $\sum_{m=1:M} w_m = 1$. This can similarly done for classification by weighting the groups of p_k corresponding to each output.

The final piece of the puzzle for using CCFs for multi-output problems is to note that when Y is the concatenation of all the outputs, then the CCA procedure given in (7), and thus the process for

calculating Φ, is unchanged. This seamless transition is an advantage to our choice of using CCA projections. In theory, one could also make use the canonical components of the outputs for the multi-output case. However, preliminary investigation suggests that a naïve approach, where these are used for the split criterion rather than the original outputs, is not beneficial.

To assess the performance of CCFs for multi-output problems we considered a selection of datasets taken from the Mulan[10] (Tsoumakas et al., 2010) and UCI (Lichman, 2013) databases. The experimental procedures was as for the single output experiments (using the same respective default parameters for regression and classification) and we make comparisons to RF and RRF. As the RF package used previously does not support multiple outputs, an adaptation of our CCF package was used instead. Dataset summaries and results are shown in Table 11 with win/draw/loss ratios for CCFs of $4/2/2$ and $4/3/1$ compared to RFs and RRFs respectively. More extensive investigation is necessary to fully assess the relative merit of CCFs compared to alternatives for multi-output problems, but these results are promising and suggest gains transfer from the single output case.

7. Discussion

7.1 Effect of Correlation between Input Features

As shown by Menze et al. (2011), RFs often struggle on data with highly correlated features. CCA naturally incorporates information about feature correlations, therefore CCFs do not suffer the same issues. This is demonstrated, for example, by their superior performance on the highly correlated *hill valley* datasets.

To more rigorously investigate the effect of correlation, we used the following method of artificially correlating the data (after carrying out the preprocessing described in Section 3.3.2):

1. Add a new random feature

$$X_{(D+1,n)} \sim \mathcal{N}(0, \kappa), \quad \forall n = 1, \ldots, N$$

   where κ is a parameter which will control the degree of correlation to be added.

2. To each of the existing features, randomly either add or subtract the new feature, i.e.

$$X_{(:,d)} \leftarrow X_{(:,d)} + \zeta_d X_{(:,D+1)}, \ \forall d = 1, \ldots, D$$

   where each $\zeta_d \overset{\text{i.i.d}}{\sim} \text{UNIFORM-DISCRETE}\{-1, 1\}$.

As shown in Figure 5, this transformation has no effect on the accuracy of CCFs, whereas the accuracy of RF decreases with increasing κ, eventually giving the same accuracy as random prediction. We also see that RRF fare only marginally better than RF under this artificial correlation. The use of PCA means rotation forests, on the other hand, exhibit similar robustness to global correlations as CCFs.

We postulate that CCFs are better than rotation forests at incorporating class dependent and localized correlations. This is because the rotation step of rotation forests does not incorporate any class information other than in the random elimination of classes. Further, individual trees in a rotation forest are orthogonal and therefore cannot incorporate spatial variation in correlation. The self similar nature of the growth algorithm for CCTs, on the other hand, means that the local correlations of a partition can be incorporated as naturally as the global correlations.

10. `http://mulan.sourceforge.net/datasets-mtr.html`

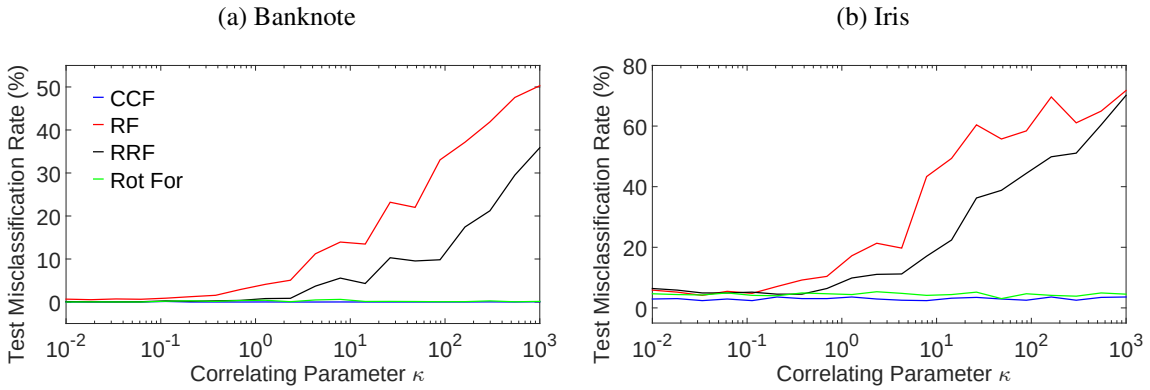


Figure 5: Misclassification rate on artificially correlated data. For each method, 5 separate 10-fold cross validations were run and the mean test accuracy across the 50 tests reported.

To investigate these suggestions formally, we tested performance on compound datasets in which localized and class dependent correlations had been artificially added. To do this, a replica of the dataset was created where a datapoint with class k in the original is assigned to class $k + K$ in the replica. The features in the original and the replica were independently correlated and a constant added to replica to separate it in the input space from the original. A new dataset was then created composing of the datapoints from the original dataset and from the replica. Specifically we construct the new dataset as

$$X \leftarrow \begin{bmatrix} \textsc{Corr}(X) \\ \textsc{Corr}(X) + \beta \mathbb{1}^{N\times(D+1)} \end{bmatrix}, \quad Y \leftarrow \begin{bmatrix} Y & \mathbf{0} \\ \mathbf{0} & Y \end{bmatrix} \tag{16}$$

where CORR denotes the correlation process described earlier in the section and β is a scalar dictating the degree of separation. Note that the CORR process is applied separately for the points in the original dataset and the replica, such that these correlation are class dependent.

We performed a single crossfold validation on a selection of the datasets from Table 2, taking $\beta = 2000$ and $\kappa = 100$. The results, given in Table 12, show that CCFs demonstrated either no change or a small loss in accuracy on all datasets, whereas there was a large loss of accuracy in all cases for RFs and RRFs, and on most of the datasets for rotation forests. These results supports our hypothesis that CCFs are better than rotation forests at dealing with localized and class dependent correlations. This is an important advancement as such correlations cannot be easily accounted for using a data preprocessing step such as PCA. We note that the only cases that did not exhibit a large drop in accuracy for rotation forests had $K = 2$, suggesting the use of class elimination in the calculation of the rotations (Rodriguez et al., 2006) may offer some robustness to class dependent correlations when the number of classes is small.

7.2 Computational Complexity

We now consider the computational complexity of the CCF training algorithm. We focus on the classification case, but note that the results similarly apply to regression by setting $K = 1$. For multiple outputs then the computational complexity is a little more complex, but the quoted results roughly hold for multivariate regression by setting K to the number of outputs and to multiple outputs by replacing K with the total number of classes across all outputs.

Table 12: Mean and standard deviations of percentage of test cases misclassified for compound datasets. Method with best accuracy is shown in bold. • and ○ indicate that CCFs were significantly better and worse respectively at the 1% level of a Wilcoxon signed rank test.

Dataset	CCF	RF	Rot For	RRF
Balance Scale	**7.36 ± 2.09**	46.72 ± 5.92 •	20.24 ± 4.46 •	40.48 ± 5.42 •
Banknote	**0.00 ± 0.00**	34.64 ± 2.94 •	2.85 ± 0.62 •	23.54 ± 1.98 •
Iris	**3.00 ± 3.31**	69.33 ± 7.83 •	19.67 ± 9.87 •	65.33 ± 8.78 •
Landsat satellite	**8.69 ± 0.97**	32.75 ± 1.44 •	11.56 ± 0.82 •	19.04 ± 0.89 •
Parkinsons	**9.74 ± 3.78**	33.33 ± 8.11 •	12.56 ± 6.22	27.18 ± 9.06 •
Vowel-n	**3.13 ± 0.46**	85.91 ± 2.82 •	20.20 ± 2.79 •	78.69 ± 2.87 •
Wisconsin cancer	**2.79 ± 1.76**	30.00 ± 2.65 •	2.93 ± 1.66	8.79 ± 2.88 •
Yeast	**38.72 ± 1.89**	70.13 ± 3.15 •	56.09 ± 1.85 •	63.13 ± 3.36 •

The two limiting factors for the complexity of CCFs are the search over possible splits and the CCA calculations. We first consider the cost of this calculation at a single node compared to a RF. The theoretically achievable computational complexity[11] for each CCA calculation is $\mathrm{O}(N_j\lambda^2)$ (Golub and Van Loan, 2012). Assuming all features are ordinal,[12] the cost of the split search a single node for CCFs given Φ is $\mathrm{O}(N_j\nu_{\max}\log N_j)$ where $\nu_{\max} \leq \min(\lambda, K-1)$ is the number dimensions that need to be searched over. This compares with $\mathrm{O}(N_j\lambda\log N_j)$ for RFs, with the dominant cost for both coming from sorting U (or equivalently $X_{(\omega_j,\delta_j)}$ for RFs). As CCA and the split search are the two dominant costs, we can say that the per node cost of CCFs is $\mathrm{O}(N_j\lambda^2 + N_j\nu_{\max}\log N_j)$ and of RFs is $\mathrm{O}(N_j\lambda\log N_j)$. Louppe (2014) showed that this per node cost for RFs leads to an overall complexity of $\mathrm{O}\left(NL\lambda\left(\log N\right)^2\right)$. Presuming that CCFs have the same structural properties as RFs, using their as Louppe (2014) gives a complexity of

$$\mathrm{O}\left(NL\lambda^2\log N + NL\nu_{\max}\left(\log N\right)^2\right)$$

for CCFs. Clearly the second term is bounded by the complexity of RFs (as $\nu_{\max} \leq \lambda$) and may be substantially smaller when K is small, e.g. for regression problems. The first term, and thus the overall CCF complexity, will also be bounded by the complexity of RF if $\lambda < \log N$. Using the recommended value of $\lambda = \lceil \log_2(D) + 1 \rceil$ this means the complexity of CCF is upper bounded by that of RF whenever $N > D$ and is at worst a factor of $\log(D)/\log(N)$ larger.

If Φ is instead calculated using random rotations then this can be done in $O(\lambda^2)$ time (Blaser and Fryzlewicz, 2016). Consequently it immediately follows that the computational complexity for random rotation approaches is $\mathrm{O}\left(NL\lambda\left(\log N\right)^2 + L\lambda^2\log N\right)$ which is the same as RFs whenever $\lambda < N\log N$.

11. This is based on exploiting the LDA equivalence. The numerically stable method outlined in Appendix A that was used for our experiments is actually $\mathrm{O}\left(N_j(\lambda+K)^2 + \lambda K\min(\lambda,K)\right)$, with the additional terms occuring due to the QR decomposition of the centred Y, the multiplication of the Q matrices, and the SVD decomposition. In practise we found the constant factor for these terms to be small and therefore that this was not a problem. However, for datasets with a very large K, an alternative method for CCA might be preferable, for example by exploiting the LDA equivalence, if numerical stability can be ensured by other means. This is also why the computational complexity for the multi-output cases is more complicated.
12. In the presence of non-ordinal features, λ is replaced in the CCF complexity with the number of features used after the 1-of-K encoding is applied.

Rotation forest training requires each tree to search the full set of features and therefore has a training complexity of $\mathrm{O}\left(NLD\left(\log N\right)^2\right)$. This is exponentially more expensive in the number of features than RFs and CCFs when λ is set to a logarithmic factor of D. This makes their implementation impractical for datasets with more than a modest number of features, as demonstrated by the *ORL* and *polya* datasets for which we were unable to carry out most tests with rotation forests, experiencing both memory issues and an impractically long training time.

Should one wish to use CCFs with a λ that is significantly larger than the suggested default, speeds gains should be achievable by separately generating M projection matrices $\{\Phi_m\}_{m=1:M}$, each using λ/M features, and searching for the optimal split over all generated projections. When K is small, such an approach could also be used to increase the size of the set of candidate splits generated and might therefore also be beneficial to performance in some scenarios. We leave assessing the effect of this change on predictive performance to future work.

7.3 Empirical Run Times

To more explicitly evaluate the practical speed of CCFs compared to competitors, we now consider empirical run times. To ensure fair comparison, we used our code base for the CCF, CCF-Bag, RRF, and RRF-Bag implementations for calculating the RFs and rotation forests timings. All methods thus shared the same code except in the calculation of Φ (which is just set to an identity matrix for RFs and rotation forests), whether bagging is used, and the tree projection calculation and different value of λ required for rotation forests. Runs were carried out on a single machine with 64 AMD Opteron 1.4GHz processors and using 2 threads per core.

Table 13 shows the absolute run times for CCFs and the run times of the alternatives as a ratio of the corresponding CCF run time. To characterise whether changes originate from differences in the size of the learnt trees, the average number of nodes for each tree are also shown. As expected, the run times for rotation forests were significantly larger that the other algorithms for any datasets with more than a few input features. CCF-Bag was (marginally) faster than CCFs and produced smaller trees, as would also be expected because bagging reduces the effective size of the dataset each tree is trained on. Perhaps surprisingly though, RFs, RRFs, and RRF-Bag were all substantially slower than CCFs, each having longer run times on every dataset except *spirals* and respectively taking on average 1.92, 2.03, and 1.83 times longer to run than CCFs. These numbers rise further to 2.57, 2.62, and 2.35 respectively if one omits time spent on parallelization overheads. Although the relative speeds will of course vary between implementations, these results suggest not only do CCFs offer better predictive accuracy than the alternatives, they are faster to train as well.

Though occasionally producing substantially larger trees, RFs and RRF-Bag more often than not produced smaller trees than CCFs, again presumably because of the reduction in effective training dataset size caused by bagging. The speed differences for these algorithms cannot thus originate predominantly from changes in the tree size. CCFs are in fact faster because they consider, on average, a smaller number of candidate splits than the other approaches, with the cost of searching of these splits typically being larger than the cost of carrying out CCA. This is easy to see in the case were $K < \lambda$ as CCFs search over $\nu_{\max} \leq \max(K, \lambda)$ projections, compared to λ for the alternatives. Somewhat more subtly, then this effect still occurs for the tree as a whole even when $K > \lambda$. Although it is still necessary to search over λ splits at the root of the tree when $K > \lambda$, the partitioning procedure attempts to create the "purest" possible children and so the average number of classes present at a node decreases the deeper one goes down the tree. As $\nu_{\max}$ for a particular

Table 13: Combined time for training and testing (using a $90\%/10\%$ train/test split, 500 trees, and the default parameters) and average number of nodes in each tree for CCFs and the other algorithms. CCF results are quoted in absolute values, while others are given as ratios of the equivalent value for CCFs. Results that correspond to faster run times or smaller trees than the corresponding CCF are shown in bold. Results are averaged across 4 separate train / test splits. Variations in run time and average tree size between independent runs were small (typically in the range $0.5\% - 5\%$).

Dataset	CCF	Run Time / CCF Run Time					CCF	Avg. #Nodes / CCF Avg. #Nodes				
	Time (s)	RF	Rot-For	CCF-Bag	RRF	RRF-Bag	#Nodes	RF	Rot-For	CCF-Bag	RRF	RRF-Bag
Balance scale	2.41	1.44	1.24	**0.81**	1.29	1.09	125	1.49	1.34	**0.613**	1.31	**0.857**
Banknote	0.716	1.72	1.65	1.13	1.37	1.53	16.5	2.18	2.16	**0.943**	1.88	1.72
Breast tissue	0.928	1.13	1.68	**0.925**	1.35	1.19	43.4	**0.758**	1.03	**0.579**	1.16	**0.801**
Climate crashes	0.835	1.27	2.36	**0.912**	1.54	1.34	35.0	**0.76**	**0.899**	**0.562**	1.20	**0.83**
Fertility	0.804	1.16	1.26	**0.915**	1.14	1.02	26.1	**0.879**	**0.984**	**0.525**	**0.947**	**0.616**
Heart-SPECT	1.54	1.48	2.63	**0.78**	1.47	1.18	79.7	**0.974**	1.05	**0.526**	**0.966**	**0.608**
Heart-SPECTF	1.05	1.51	4.65	**0.821**	1.67	1.43	58.0	**0.765**	**0.718**	**0.502**	1.02	**0.697**
Hill valley	0.847	14.6	7.55	1.02	11.7	10.6	25.1	14.0	**0.576**	**0.785**	13.0	10.1
Hill valley noisy	4.39	2.64	12.6	**0.764**	2.93	2.47	319	1.00	**0.709**	**0.511**	1.42	**0.985**
ILPD	3.14	1.25	1.90	**0.812**	1.58	1.26	196	**0.746**	**0.878**	**0.553**	**0.992**	**0.672**
Ionosphere	0.945	1.56	4.07	**0.922**	1.77	1.61	47.7	**0.773**	**0.782**	**0.571**	1.11	**0.80**
Iris	0.706	1.14	1.05	1.02	1.05	1.12	12.0	1.03	1.27	**0.721**	1.47	1.09
Landsat satellite	15.7	1.76	5.96	**0.824**	1.94	1.66	946	**0.853**	**0.81**	**0.54**	1.10	**0.756**
Letter	124	1.15	2.85	**0.936**	1.34	1.24	4656	**0.849**	**0.932**	**0.658**	1.28	**0.933**
Libras	2.74	1.63	9.58	**0.89**	1.80	1.59	130	1.01	**0.819**	**0.632**	1.23	**0.938**
Magic	47.0	1.43	2.53	**0.804**	1.82	1.54	3901	**0.641**	**0.874**	**0.553**	1.03	**0.69**
Nursery	11.0	1.25	2.03	1.04	2.14	2.25	562	**0.733**	1.12	**0.784**	1.73	1.47
ORL	15.9	1.22	613	**0.968**	1.41	1.35	200	**0.756**	**0.542**	**0.611**	1.26	**0.92**
Optical digits	20.9	1.40	6.75	**0.899**	1.70	1.51	849	**0.91**	**0.724**	**0.63**	1.36	**0.986**
Parkinsons	0.782	1.26	2.19	**0.942**	1.37	1.22	30.3	**0.808**	**0.78**	**0.607**	1.06	**0.785**
Pen digits	18.0	1.39	3.05	**0.993**	1.49	1.44	590	1.05	**0.957**	**0.721**	1.36	1.06
Polya	31.6	2.43	21.3	**0.703**	2.41	2.00	2245	**0.94**	**0.648**	**0.459**	1.07	**0.726**
Seeds	0.724	1.29	1.35	1.01	1.27	1.20	20.8	1.08	1.23	**0.665**	1.43	1.03
Skin seg	29.3	1.93	1.66	1.28	1.47	1.78	367	1.26	**0.791**	**0.763**	1.11	**0.866**
Soybean	4.00	1.67	5.61	**0.924**	1.79	1.72	119	1.02	1.01	**0.699**	1.20	**0.949**
Spirals	9.00	1.06	**0.923**	**0.982**	**0.945**	**0.929**	263	1.17	1.36	1.00	**0.835**	**0.839**
Splice	9.21	4.52	21.0	**0.751**	6.25	5.14	429	1.20	**0.57**	**0.403**	1.85	1.28
Vehicle	3.39	1.66	3.47	**0.807**	2.01	1.68	237	**0.881**	**0.926**	**0.562**	1.24	**0.865**
Vowel-c	4.59	1.91	4.39	**0.959**	2.12	2.02	225	1.10	1.14	**0.70**	1.47	1.21
Vowel-n	4.66	1.36	2.20	**0.956**	1.57	1.52	255	**0.933**	1.08	**0.728**	1.25	1.01
Waveform (1)	12.5	1.75	4.10	**0.765**	2.16	1.80	991	**0.791**	**0.834**	**0.522**	1.16	**0.789**
Waveform (2)	13.4	1.92	6.84	**0.763**	2.67	2.19	1038	**0.786**	**0.777**	**0.506**	1.34	**0.916**
Wholesale-c	1.17	1.21	1.83	**0.845**	1.47	1.30	70.8	**0.655**	**0.872**	**0.57**	1.05	**0.705**
Wholesale-r	2.94	1.10	1.74	**0.803**	1.37	1.11	225	**0.657**	**0.906**	**0.56**	1.01	**0.65**
Wisconsin cancer	0.883	1.61	1.73	**0.931**	1.44	1.31	43.9	1.00	**0.887**	**0.564**	1.03	**0.707**
Yeast	12.0	1.12	1.84	**0.793**	1.24	1.03	862	**0.748**	**0.966**	**0.575**	1.03	**0.662**
Zoo	0.749	1.17	1.46	1.11	1.17	1.30	20.6	**0.932**	**0.841**	**0.786**	1.07	**0.885**
Average	11.2	1.92	20.9	**0.906**	2.03	1.83	548	1.30	**0.941**	**0.627**	1.54	1.15

node is less than the number of classes present, once there are fewer classes than λ, CCFs needs to search over fewer splits than the alternatives at that node and all its descendants. Therefore fewer candidate splits are considered for the tree as a whole, leading to a faster training speed.

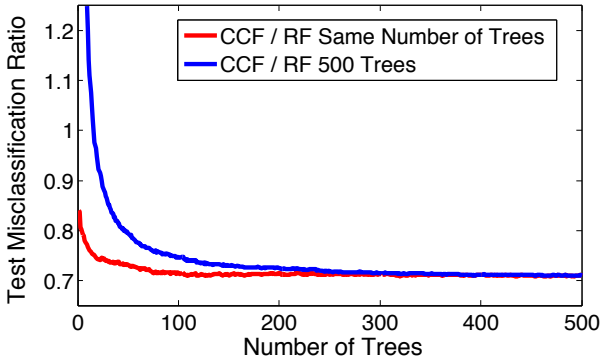


Figure 6: Mean misclassification ratios for 37 datasets in Table 2. The red line shows the average across the datasets of the ratio between the number of CCF and RF test misclassifications for forests of the same size. The blue line gives the same ratio when comparing CCFs of varying number of trees to a RF with 500 trees.

7.4 Effect of Number of Trees

To investigate the variation in performance with the number of trees we used the individual tree predictions from Section 4.1 to evaluate the accuracy of RF and CCF for different ensemble sizes. Figure 6 summarises the results and shows that, as expected, it is always beneficial to use more trees, but that there are diminishing returns. Some datasets require more trees than others before the accuracy begins to saturate (not shown), but the advantages of CCFs over RFs is maintained across all ensemble sizes (though for a very small number of trees this is slightly less pronounced). A further result of interest is that on average it only takes around 15 CCTs to match the accuracy of 500 RF trees. One could therefore envisage using a smaller CCF as a fast and accurate alternative to a larger RF under a restricted computational budget.

8. Conclusions

We have introduced canonical correlation forests (CCFs), a new decision tree ensemble method for classification, regression, and multiple output prediction. The improvements of CCFs over previous decision tree ensemble approaches are rooted in two core innovations: the use of CCA for generating candidate hyperplane splits and a novel new alternative to bagging for oblique decision trees, projection bootstrapping, which retains the full dataset for split selection in the projected space. We believe that CCFs represent a new performance benchmark for decision tree ensembles and have provided substantial empirical evidence to suggest that they outperform a number of prominent approaches such as random forests (RFs), rotation forests, and randomly rotated alternatives, while maintaining a similar or better computational complexity. We have also provided evidence suggesting that classification CCFs outperform all of the 179 classifiers considered in a recent survey (Fernández-Delgado et al., 2014), including a number of prominent SVM and neural network packages, over a large selection of datasets, even when CCFs are run in an out-of-box fashion using default parameters and extensive parameter tuning is applied for the alternatives. Although it would be presumptive to conclude from this that CCFs create a new benchmark for black-box classifiers more generally, it does emphatically underline their robustness and wide ranging potential utility.

Acknowledgments

Tom Rainforth is supported by a BP industrial grant. Frank Wood is supported under DARPA PPAML through the U.S. AFRL under Cooperative Agreement number FA8750-14-2-0006, Sub Award number 61160290-111668.

Appendix A. Numerically Stable CCA Algorithm

As explained in the main paper, the aim of CCA is to find pairs of linear projections $\{A_\nu, B_\nu\}_{\nu=1:\nu_{\max}}$ that maximize the correlations between two matrices $W \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{n\times k}$ in the co-projected space $\{WA_\nu, VB_\nu\}$. Let Σ_{WV} be the cross covariance matrix between W and V such that the (i,j) entry of Σ_{WV} (which we denote $(\Sigma_{WV})_{(i,j)}$) is the empirical covariance between the i^{th} column of W and the j^{th} column of V, namely

$$(\Sigma_{WV})_{(i,j)} = \frac{1}{N-1}\sum_{n=1}^{N}\left(W_{(n,i)} - \bar{W}_i\right)\cdot\left(V_{(n,j)} - \bar{V}_j\right) \tag{17}$$
$$\text{where} \quad \bar{W}_i = \frac{1}{N}\sum_{n=1}^{N} W_{(n,i)}, \quad \text{and} \quad \bar{V}_j = \frac{1}{N}\sum_{n=1}^{N} V_{(n,j)}.$$

Defining Σ_{WW} and Σ_{VV} in the same way, then a simple approach to calculate the canonical coefficients $\{A_\nu, B_\nu\}_{\nu=1:\nu_{\max}}$ is to note that the satisfy (see e.g. Borga (2001))

$$\begin{aligned}\Sigma_{WW}^{-1}\Sigma_{WV}\Sigma_{VV}^{-1}\Sigma_{VW}A_\nu &= \rho_\nu^2 A_\nu \\ \Sigma_{VV}^{-1}\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}B_\nu &= \rho_\nu^2 B_\nu\end{aligned} \tag{18}$$

such that they are the eigenvectors of $\Sigma_{WW}^{-1}\Sigma_{WV}\Sigma_{VV}^{-1}\Sigma_{VW}$ and $\Sigma_{VV}^{-1}\Sigma_{VW}\Sigma_{WW}^{-1}\Sigma_{WV}$ respectively. The common eigenvalues, which correspond to the squares of the canonical correlations ρ_ν, make the pairing between the coefficients of W and V apparent and the order of the coefficients is found by the corresponding decreasing order of the canonical correlations, $\rho_1 \geq \rho_2 \geq \cdots \geq \rho_{\nu_{\max}}$.

However, this solution can be numerically unstable as it requires an inversion of the often degenerate covariance matrices. We instead take the numerically stable approach laid out in Algorithm 3, corresponding to the approach introduced by Björck and Golub (1973). Here $[q, r, p] = \text{QR}(\alpha)$ refers to a QR decomposition with pivoting such that $qr = \alpha_{(:,p)}$ where q is an orthogonal matrix, r is upper triangular matrix and p is a column ordering defined implicitly such that $|r\,(i,i)| > |r\,(j,j)| \;\; \forall i < j$. $[u, \Omega, z] = \text{SVD}(\alpha)$ refers to a singular value decomposition such that $u\Omega z^T = \alpha$ where u and z are unitary matrix and Ω is diagonal matrix of singular values, with the ordering defined such that $\Omega\,(i,i) > \Omega\,(j,j) \;\; \forall i < j$.

After centring both inputs, a QR decomposition with pivoting is carried out on each. For example $qr = wp$ where q is a unitary matrix, r is an upper triangular matrix and p is a pivot matrix such that the diagonal elements of p are of decreasing magnitude. If

$$\zeta = \max\left\{i : \left|r_{(i,i)}\right| > \varepsilon\left|r_{(1,1)}\right|\right\} \tag{19}$$

is the number of non-zero main diagonal terms within some tolerance ε, then the first ζ columns of q will describe an orthonormal basis for the span of w. Therefore by applying the reductions $q' \leftarrow q_{(:\,,1:\zeta)}$ and $r' \leftarrow r_{(1:\zeta,1:\zeta)}$, then $q'r'$ will be a pivoted reduction of w that is full rank and r'

Algorithm 3: Numerically Stable CCA

Inputs: First array $w \in \mathbb{R}^{N\times D}$, second array $v \in \mathbb{R}^{N\times K}$, tolerance parameter $\varepsilon \in [0^+, 1^-]$

Outputs: Projection matrices for first and second arrays A and B, correlations ρ

$\mu^w = \frac{1}{N}\sum_{n=1:N} w_{(n,:)}, \quad \mu^v = \frac{1}{N}\sum_{n=1:N} v_{(n,:)}$ ▷ First centre the inputs

$w_{(:,d)} \leftarrow w_{(:,d)} - \mu^w_{(d)} \quad \forall d \in \{1,\dots,D\}$

$v_{(:,k)} \leftarrow v_{(:,k)} - \mu^v_{(k)} \quad \forall k \in \{1,\dots,K\}$

$[q^w, r^w, p^w] = \mathrm{QR}(w), \quad [q^v, r^v, p^v] = \mathrm{QR}(v)$ ▷ Carry out pivoted QR decompositions

$\zeta^w = \max\{i : |r^w_{(i,i)}| < \varepsilon |r^w_{(1,1)}|\}, \quad \zeta^v = \max\{i : |r^v_{(i,i)}| < \varepsilon |r^v_{(1,1)}|\}$

$q^w \leftarrow q^w_{(:,1:\zeta^w)}, \quad r^w \leftarrow r^w_{(1:\zeta^w,1:\zeta^w)}, \quad q^v \leftarrow q^v_{(:,1:\zeta^v)}, \quad r^v \leftarrow r^v_{(1:\zeta^v,1:\zeta^v)}$ ▷ Reduce to full rank

$\nu_{\max} = \min(\zeta^w, \zeta^v)$ ▷ Number of coefficient pairs that will be returned

if $\zeta^w > \zeta^v$ **then** ▷ Select which of two equivalent SVD decompositions is faster

  $[u, \Omega, z] = \mathrm{SVD}((q^w)^T q^v)$

else

  $[z, \Omega, u] = \mathrm{SVD}((q^v)^T q^w)$

end if

$u \leftarrow u_{(:,1:\nu_{\max})}, \quad z \leftarrow z_{(:,1:\nu_{\max})}$ ▷ Remove meaningless components

$A = (r^w)^{-1} u, \quad B = (r^v)^{-1} z$ ▷ Using back substitution as r^w and r^v are upper triangular

$\rho = \mathrm{DIAG}(\Omega)$

$A_{(p^w,:)} \leftarrow \left[A^T, \mathbf{0}\right]^T$ ▷ Reorder rows to match corresponding columns in original matrices

$B_{(p^v,:)} \leftarrow \left[B^T, \mathbf{0}\right]^T$ ▷ Some rows are set to zero if original matrix was not full rank

return A, B, ρ

will be invertible. The algorithm then proceeds with the reduced matrices q' and r' for each input to carry out the CCA in a numerical stable manner.

Although for analytical application then the rank tolerance parameter ε should be taken as 0^+, we recommend taking a finite value (we use $\varepsilon = 10^{-4}$) to guard against numerical error and because this can act as a regularization term against individual splits overfitting the inputs. Note that packaged applications of this algorithm are available, for example CANONCORR in MATLAB, but in general these do not allow ε to be set manually.

Appendix B. Detailed Training Algorithms

In this section we provide more detailed versions of the training Algorithms 1 and 2 given in Algorithms 4 and 5 respectively. These differ by providing explicit consideration for categorical features, missing data, and edge cases (e.g. when a node only has two unique nodes).

Algorithm 4: CCF training algorithm (detailed version)

Inputs: ordinal features $X^r \in \mathbb{R}^{N \times D_r}$, categorical features $X^c \in \mathcal{S}^{N \times D_c}$, outputs $Y \in \mathbb{Y}^N$, number of trees L (default is 500), number of features to sub-sample $\lambda \in \{1, \ldots, D_r + D_c\}$ (default is $\lceil \log_2(D_r + D_c) + 1 \rceil$), impurity measure g (default is entropy as per (12) for classification and mean squared error as per (13) for regression), stopping criteria c (see Section 3.3)

Outputs: CCF T

Convert X^c to 1-of-K encoding $X^b \in \mathbb{I}^{N \times D_b}$
$X = \left\{X^r, X^b\right\}$
for $d \in 1 : D_b + D_c$ **do**
    $\mu_{(d)} = \frac{1}{N} \sum_{n=1}^{N} X_{(n,d)}$ ▷ Calculate feature means and stand deviations (ignore missing terms)
    $\sigma_{(d)} = \sqrt{\frac{1}{N} \sum_{n=1}^{N} {X_{(n,d)}}^2 - {\mu_{(d)}}^2}$
    $X_{(:,d)} \leftarrow \left(X_{(:,d)} - \mu_{(d)}\right) / \sigma_{(d)}$ ▷ Convert X to z scores
end for
if $\lambda < (D_r + D_c)$ **then** ▷ Don't projection bootstrap if $\lambda = (D_r + D_c)$
    $b = \text{true}$
else
    $b = \text{false}$
end if
for $i = 1 : L$ **do**
    $X' \leftarrow X$
    Randomly assign each missing value in X' to an independent draw from a unit normal
    if b **then**
        $Y' \leftarrow Y$
    else
        $\{X', Y'\} \leftarrow$ bootstrap sample N rows from $\{X', Y\}$
    end if
    $t_i \leftarrow \text{GROWTREE}(0, X', Y', \{1, \ldots, D_r + D_c\}, \lambda, g, b, c)$
end for
return $T = \{t_i\}_{i=1 \ldots L}$

Appendix C. Inverted Cross Validation

To investigate the performance of CCFs on datasets containing few data points relative to the complexity of the underlying structure, we carried out inverted cross validations, training on one fold and testing on the other nine. 15 such tests were carried out, using the same folds as the original cross validation. Table 14 gives the results on each of the individual datasets, Table 15 gives a summary of the significant victories and losses, and Table 16 shows the average rank. The parameters used were the same as for the standard cross validation except that only 200 trees were used.

The comparative performances were overall similar to the standard cross-validation case, though there were a number of relative changes for individual datasets. Two results of particular note were that all methods performed worse than predicting the most popular class for the *fertility* and *wholesale-r* datasets, which gives misclassification rates of 12.0% and 28.2% respectively. This suggests that it may be beneficial to incorporate information about the overall class ratios into the training algorithms.

Algorithm 5: GROWTREE (detailed version)

Inputs: unique node identifier for root node j, $X^j = X_{(\omega_j,:)} \in \mathbb{R}^{N_j \times (D_r+D_b)}$, $Y^j = Y_{(\omega_j,:)} \in \mathbb{Y}^{N_j}$, available feature ids $D_j \subseteq \{1,\ldots,D_r+D_c\}$, λ, g, whether to projection bootstrap b, c

Outputs: subtree $\{\Psi_j, \Theta_j\}$ where Ψ_j are discriminant nodes and Θ_j leaf nodes

Sample $\delta_j \subseteq D_j$ by taking $\min(\lambda, |D_j|)$ samples without replacement from D_j
While δ_j contains features without variation, eliminate these from D_j and δ_j and resample
$\gamma_j = \delta_j$ mapped to the column indices of X^j in accordance with the 1-of-K encoding of X^c
if b **then**
    $\{\mathcal{X}', \mathcal{Y}'\} \leftarrow$ bootstrap sample N_j data points from $\{X^j_{(:,\gamma)}, Y^j\}$
else
    $\{\mathcal{X}', \mathcal{Y}'\} \leftarrow \left\{X^j_{(:,\gamma)}, Y^j\right\}$
end if
if all rows in $\mathcal{X}'$ or $\mathcal{Y}'$ are identical **then**
    if all rows in $\mathcal{X}^j_{(:,\gamma)}$ or Y^j are identical **then return** $\left\{\cdot, \left\{j, \frac{1}{N_j}\sum_{n=1}^{N_j} Y^j_{(n,:)}\right\}\right\}$ **end if**
    $\{\mathcal{X}', \mathcal{Y}'\} \leftarrow \{X^j_{(:,\gamma)}, Y^j\}$
end if
if $\mathcal{X}'$ contains only two unique rows **then**
    $\tilde{\mathcal{X}} = \text{UNIQUEROWS}(X')$
    $\phi_j \leftarrow \tilde{\mathcal{X}}_{(2,:)} - \tilde{\mathcal{X}}_{(1,:)}$
    $s_j \leftarrow \frac{1}{2}\left(\tilde{\mathcal{X}}_{(1,:)} + \tilde{\mathcal{X}}_{(2,:)}\right)\phi_j$
else
    $\{\Phi, \Omega\} \leftarrow \text{CCA}(\mathcal{X}', \mathcal{Y}')$ ▷ Calculate CCA coefficients using bootstrap sample
    $U \leftarrow \mathcal{X}\Phi$ ▷ Project *original features* into canonical component space
    $G_{\text{base}} \leftarrow g\left(Y^j\right)$
    for $\nu \in 1 : \nu_{\max}$ **do** ▷ $\nu_{\max} = \min(\text{rank}(\mathcal{X}'), \text{rank}(\mathcal{Y}'))$ is number of canonical coefficient pairs
        $u \leftarrow \text{SORT}(U_{(:,\nu)})$
        for $i \in 2 : N_j$ **do** ▷ Exhaustive search on unique splits
            $S_{(i,\nu)} \leftarrow (u_{(i-1)} + u_{(i)})/2$ ▷ Split halfway between consecutive points.
            $\tau^\ell \leftarrow \left\{n \in \{1,\ldots,N_j\} : U_{(n,\nu)} \le S_{(i,\nu)}\right\}$
            $N_{\chi_{j,\ell}} \leftarrow$ number of elements in τ^ℓ
            $\tau^r \leftarrow \{1,\ldots,N_j\} \setminus \tau^\ell$
            $G_{(i,\nu)} \leftarrow G_{\text{base}} - \frac{N_{\chi_{j,\ell}}}{N_j} g\left(Y_{(\tau^\ell,:)}\right) - \frac{N_j - N_{\chi_{j,\ell}}}{N_j} g\left(Y_{(\tau^r,:)}\right)$ ▷ Split gain
        end for ▷ In practise gain is calculated by iterating from previous value to avoid N_j^2 complexity.
    end for
    $\{i^*, \nu^*\} \leftarrow \text{argmax}_{i,\nu} G_{(i,\nu)}$ ▷ Choose best split
    if $G_{(i^*,\nu^*)} \le 0$ or any of c satisfied **then** ▷ Node is a leaf
        $\theta_j \leftarrow \frac{1}{N_j}\sum_{n=1}^{N_j} Y^j_{(n,:)}$
        return $\{\cdot, \{j, \theta_j\}\}$
    end if
    $\phi_j \leftarrow \Phi_{(:,\nu^*)}, \quad s_j \leftarrow S_{(i^*,\nu^*)}$ ▷ Chosen projection and split point
end if
Generate unique identifiers for children $\chi_{j,\ell}$ and $\chi_{j,r}$
$\psi_j \leftarrow \{j, \delta_j, \phi_j, s_j, \chi_{j,\ell}, \chi_{j,r}\}$
$\tau^\ell \leftarrow \left\{n \in \{1,\ldots,N_j\} : U_{(n,\nu^*)} \le S_{(i^*,\nu^*)}\right\}, \quad \tau^r \leftarrow \{1,\ldots,N_j\} \setminus \tau^\ell$ ▷ Assign datapoints to children
$\{\Psi_\ell, \Theta_\ell\} \leftarrow \text{GROWTREE}(\chi_{j,\ell}, X^j_{(\tau^\ell,:)}, Y^j_{(\tau^\ell,:)}, D_j, \lambda, g, b, c)$ ▷ Recurse for left child and right child
$\{\Psi_r, \Theta_r\} \leftarrow \text{GROWTREE}(\chi_{j,r}, X^j_{(\tau^r,:)}, Y^j_{(\tau^r,:)}, D_j, \lambda, g, b, c)$
return $\{\psi_j \cup \Psi_\ell \cup \Psi_r, \Theta_\ell \cup \Theta_r\}$

Table 14: Mean and standard deviations of percentage of test cases misclassified for inverted cross validation. Method with best accuracy is shown in bold. • and ○ indicate that CCFs were significantly better and worse respectively at the 1% level of a Wilcoxon signed rank test.

Dataset	CCF	RF	Rot-For	CCF-Bag	RRF	RRF-Bag
Balance scale	14.86 ± 2.54	21.02 ± 2.74 •	15.27 ± 2.52 •	**14.08 ± 2.59** ○	16.10 ± 2.32	14.75 ± 2.21
Banknote	**0.64 ± 0.57**	3.96 ± 1.66 •	0.89 ± 0.70 •	0.78 ± 0.58 •	1.16 ± 0.93 •	1.46 ± 0.96 •
Breast tissue	**50.51 ± 8.19**	52.06 ± 7.72 •	51.03 ± 8.10	54.76 ± 8.87 •	50.70 ± 7.11	52.05 ± 7.64
Climate crashes	7.23 ± 0.65	7.22 ± 0.51	7.22 ± 0.79	7.22 ± 0.54	**7.20 ± 0.53**	7.21 ± 0.48
Fertility	18.97 ± 8.16	14.20 ± 5.36 ○	15.38 ± 5.92 ○	16.76 ± 7.20 ○	15.73 ± 5.05 ○	**13.28 ± 2.88** ○
Heart-SPECT	20.63 ± 3.08	20.76 ± 3.06	20.09 ± 3.22 ○	20.04 ± 2.83 ○	19.10 ± 2.50 ○	**18.88 ± 2.51** ○
Heart-SPECTF	21.11 ± 1.87	20.79 ± 1.93 ○	20.93 ± 2.05 ○	21.38 ± 2.56	**20.60 ± 1.99** ○	20.61 ± 1.74 ○
Hill valley	**0.14 ± 0.39**	48.34 ± 1.75 •	7.18 ± 1.77 •	0.17 ± 0.45	34.89 ± 3.69 •	37.71 ± 2.94 •
Hill valley noisy	**20.37 ± 4.07**	49.27 ± 1.48 •	22.22 ± 3.50 •	21.63 ± 4.57 •	41.16 ± 2.75 •	42.90 ± 2.34 •
ILPD	30.65 ± 1.79	30.68 ± 1.71	**30.14 ± 1.52** ○	30.29 ± 1.62 ○	31.14 ± 1.70 •	30.43 ± 1.63
Ionosphere	**11.90 ± 4.05**	13.90 ± 4.44 •	12.95 ± 4.28 •	16.78 ± 4.82 •	12.81 ± 4.34 •	15.45 ± 5.26 •
Iris	5.93 ± 3.83	7.59 ± 4.31 •	9.74 ± 5.64 •	**5.65 ± 4.13**	8.35 ± 4.71 •	9.42 ± 4.98 •
Landsat satellite	11.58 ± 0.41	12.07 ± 0.49 •	**11.41 ± 0.45** ○	12.19 ± 0.42 •	11.60 ± 0.46	12.10 ± 0.46 •
Letter	**10.14 ± 0.47**	13.46 ± 0.57 •	11.14 ± 0.48 •	11.22 ± 0.48 •	11.80 ± 0.49 •	13.24 ± 0.54 •
Libras	**49.86 ± 4.83**	61.57 ± 4.31 •	55.27 ± 4.82 •	52.30 ± 4.66 •	53.94 ± 4.67 •	56.17 ± 4.82 •
MAGIC	**13.46 ± 0.22**	14.03 ± 0.28 •	14.19 ± 0.29 •	13.55 ± 0.23 •	14.41 ± 0.26	14.51 ± 0.27
Nursery	3.15 ± 0.34	3.97 ± 0.43 •	**2.89 ± 0.39** ○	3.67 ± 0.36 •	3.86 ± 0.39 •	4.48 ± 0.42 •
ORL	**54.95 ± 3.80**	60.35 ± 4.04 •	-	61.22 ± 3.74 •	55.74 ± 3.81 •	61.20 ± 4.13 •
Optical digits	**3.54 ± 0.38**	4.79 ± 0.47 •	3.91 ± 0.42 •	3.93 ± 0.38 •	3.91 ± 0.42 •	4.42 ± 0.45 •
Parkinsons	**17.06 ± 4.05**	18.75 ± 4.02 •	18.55 ± 4.43 •	18.24 ± 4.62 •	17.64 ± 4.11 •	18.35 ± 4.23 •
Pen digits	**1.43 ± 0.22**	2.97 ± 0.35 •	1.84 ± 0.27 •	1.67 ± 0.24 •	1.81 ± 0.26 •	2.15 ± 0.29 •
Polya	23.80 ± 0.51	24.16 ± 0.61 •	**23.08 ± 0.50** ○	23.84 ± 0.50	24.37 ± 0.67 •	25.04 ± 0.75 •
Seeds	8.84 ± 3.14	13.34 ± 3.21 •	11.71 ± 3.70 •	**8.53 ± 3.21** ○	10.50 ± 2.53 •	10.34 ± 2.83 •
Skin seg	0.06 ± 0.01	0.13 ± 0.02 •	0.10 ± 0.01 •	0.06 ± 0.01	**0.06 ± 0.01**	0.07 ± 0.01 •
Soybean	**18.96 ± 3.95**	21.99 ± 4.00 •	21.22 ± 4.73 •	20.23 ± 4.06 •	19.20 ± 4.01	20.84 ± 3.98 •
Spirals	**0.68 ± 0.16**	3.29 ± 0.49 •	3.80 ± 0.57 •	**0.68 ± 0.16**	0.75 ± 0.16 •	0.75 ± 0.16 •
Splice	7.53 ± 1.75	**4.81 ± 0.68** ○	6.72 ± 1.11 ○	9.06 ± 2.12 •	22.34 ± 4.99 •	27.43 ± 5.62 •
Vehicle	**26.02 ± 2.14**	32.76 ± 2.66 •	27.79 ± 2.31 •	26.78 ± 2.26 •	30.76 ± 2.47 •	32.29 ± 2.49 •
Vowel-c	**40.71 ± 3.17**	46.42 ± 3.00 •	41.63 ± 2.87 •	41.46 ± 3.44 •	50.26 ± 2.74 •	52.28 ± 2.71 •
Vowel-n	**34.15 ± 2.99**	41.52 ± 2.94 •	35.82 ± 3.16 •	35.37 ± 2.92 •	37.42 ± 3.09 •	39.74 ± 3.16 •
Waveform (1)	14.76 ± 0.41	16.52 ± 0.53 •	14.91 ± 0.42 •	**14.66 ± 0.41** ○	14.81 ± 0.40	14.74 ± 0.45
Waveform (2)	14.83 ± 0.48	16.24 ± 0.49 •	14.83 ± 0.49	**14.70 ± 0.47** ○	15.66 ± 0.62 •	15.84 ± 0.73 •
Wholesale-c	11.41 ± 2.14	10.82 ± 2.00 ○	**10.58 ± 1.84** ○	11.16 ± 2.13 ○	11.42 ± 2.16	11.19 ± 1.93
Wholesale-r	35.64 ± 4.09	33.38 ± 3.66 ○	**31.23 ± 3.59** ○	33.08 ± 3.81 ○	36.74 ± 4.10 •	33.14 ± 3.76 ○
Wisconsin	4.12 ± 1.02	4.31 ± 0.98 •	3.53 ± 0.69 ○	4.10 ± 1.03	3.55 ± 0.70 ○	**3.43 ± 0.63** ○
Yeast	46.12 ± 1.86	45.98 ± 1.91	45.32 ± 1.82 ○	45.37 ± 1.83 ○	46.01 ± 1.90	**44.96 ± 1.69** ○
Zoo	**23.43 ± 10.80**	25.36 ± 11.26 •	24.10 ± 10.66	24.12 ± 11.15 •	23.75 ± 10.65	24.75 ± 11.07 •

Table 15: Number of victories column vs row at 1% significance level of Wilcoxon signed rank test for inverted cross-validation. Victories and losses for CCFs are shown in blue and red respectively.

	CCF	RF	Rotation Forest	CCF-Bag	RRF	RRF-Bag
CCF	-	**5**	**12**	**10**	**4**	**6**
RF	**28**	-	26	26	23	19
Rotation Forest	**20**	5	-	15	10	7
CCF-Bag	**19**	8	12	-	8	7
RRF	**22**	7	16	20	-	8
RRF-Bag	**24**	8	20	20	22	-

Table 16: Mean rank across datasets of mean test misclassification rate for inverted cross-validation. Lower rank corresponds to better performance (average rank $= 3.5$). *ORL* dataset is omitted from comparison to avoid biasing against rotation forests.

Algorithm	Mean Rank
CCF	2.51
CCF-Bag	2.89
Rotation Forest	3.14
RRF	3.55
RRF-Bag	4.03
RF	4.88

Appendix D. Trade-off between Tree Diversity and Accuracy

As we asserted in the introduction, the two key factors in the predictive accuracy of any decision tree ensemble method are the predictive accuracy of the individual trees and the diversity of the tree predictions. In this section, we investigate whether the improvements of CCFs over alternatives originate from one or both of these factors. To do so we will construct so-called diversity-error diagrams (Margineantu and Dietterich, 1997) for the different methods. These consider the error and similarity between the predictions between of pairs of trees. Given an ensemble of L trees, $L(L-1)/2$ such pairwise comparisons can be made, and therefore a scatter plot of accuracy vs diversity for pairs of trees can be constructed, providing a representation for the ensemble as a whole. As recommended by Kuncheva and Whitaker (2003), we use as our diversity metric the Q statistic (Yule, 1900). Given a pair of classifiers, then if E^{11}, E^{10}, E^{01}, and E^{00} are respectively the number test instances correctly classified by both classifiers, only the first classifier, only the second classifier, and neither classifier respectively, then the Q statistic is defined as

$$Q = \frac{E^{11}E^{00} - E^{01}E^{10}}{E^{11}E^{00} + E^{01}E^{10}}. \tag{20}$$

The possible values of Q vary between -1 and $+1$ with the former corresponding to classifiers that *never* make the same error (or are never correct at the same time), the latter to classifiers that *always* make the same error, and $Q = 0$ indicating independence. We can therefore use Q as a measure of diversity (with lower Q indicating more diversity), as it characterizes how often pairs of classifiers have the same errors relative to how often they have different errors.

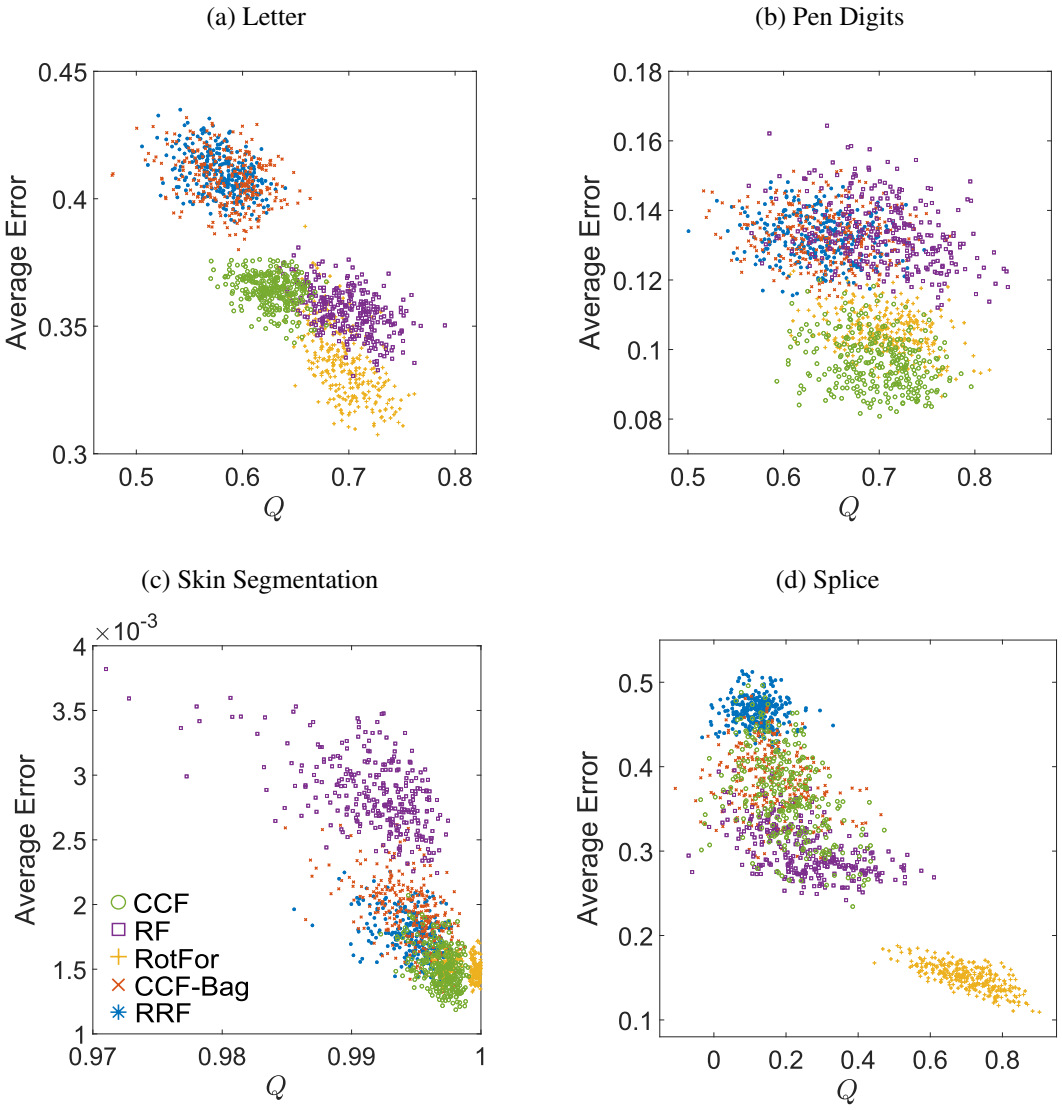


Figure 7: Diversity-error plots for four different datasets. Each point represents a pair of trees in the ensemble of 25 trees. Q is a diversity measure as defined in 20. The error corresponds to the average test misclassification rate of the two trees.

Diversity-error diagrams for CCFs, RFs, rotation forests, CCF-Bag, and RRFs on four different datasets are shown in Figure 7. These are based on training trees using 10% of the data an then calculating the average error and Q statistic for each pair of trees using the remaining 90% as per the inverted cross validation carried out in Appendix C. This inverted train/test split was used to ensure there were enough test points to calculate stable Q values. Note that, by necessity, the diagrams are produced by only a single train/test split, but for the presented datasets then qualitative variations in the generated plots were relatively small.

For the *letter* and *pen digits* datasets, Table 14 shows that CCFs outer-performed all the other approaches in predictive accuracy. However, Figures 7a and 7b suggest these advantages may stem

from different causes. Figure 7b shows a similarly diversity for all the approaches on the *pen digits* dataset, but a better tree accuracy for CCFs than the others. Figure 7a suggests that CCFs may have outperformed CCF-Bag and RRFs on the *letter* dataset due to having better individual tree accuracy, but the advantages compared with RFs now seem to stem from better diversity. Rotation forests had a better tree accuracy but worse diversity than CCFs, with the better ensemble performance of CCFs suggesting they had a better trade-off for this dataset.

For the *skin segmentation* dataset then Table 14 shows that CCFs outperformed RFs and rotation forests, but had similar performance to CCF-Bag and RRFs. Figure 7c suggests that the advantage over RFs comes from better tree accuracy and the advantage over rotation forests from better diversity. It shows that CCFs gave more accurate but less diverse trees than CCF-Bag and RRFs.

For the *splice* dataset then CCFs outperformed CCF-Bag and RRFs (with the latter doing particularly poorly), but were outperformed by RFs and rotation forests. From Figure 7d, the poor tree accuracy of RRFs is very apparent, while RFs seem to give better accuracy than CCFs, at the expense of slightly less diversity. Rotation forests, which were outperformed by RFs, gave significantly more accurate trees than all the other approaches, but far less diversity.

Other than suggesting a tendency for rotation forests to produce relatively accurate but low diversity trees, the diagrams do not appear to show a clear pattern of differences. This suggests that the advantages of CCFs can originate from either improved tree accuracy or diversity depending on the dataset.

Appendix E. Detailed Results for Comparisons to Other Classifiers

Table 17 gives a detailed comparison of CCFs to MATLAB's TREEBAGGER algorithm, along with a comparison to a summary of the performance of the 179 classifiers tested in Fernández-Delgado et al. (2014) for 82 UCI datasets. Both CCFs and TREEBAGGER used 500 trees and the default tree parameter settings laid out in Section 3.3.3. For significance comparisons with the TREEBAGGER algorithm a 10% significance level was necessary as the Wilcoxon signed rank test cannot give a p-value lower than 6.25% for 4-fold cross validation. This gave a win/draw/loss ratio for CCFs of 22/56/4. It should be noted that the number of losses is no more than would be expected by chance and that the reason that there are many more draws than in the tests of Section 3.1 is that a many fewer tests were carried out for each dataset (4 instead of 150). There were a number of datasets were CCFs performed noticeably better then any of the other classifiers, with the performance on the *hill-valley-noisy* dataset particularly impressive, giving a error rate of 6.93% with the next best classifier only achieving 25.7%.

Table 17: Dataset summaries and mean and standard deviations of percentage of test cases misclassified for CCFs and TREEBAGGER on 82 datasets used for comparison to classifiers used in Fernández-Delgado et al. (2014). Datasets were a predefined train / test split was used instead of a 4-fold cross validation have no standard deviation given. K = number of classes, N = number of data points and D = number of features (note all datasets only contain ordinal features). Method with best accuracy is shown in bold. • and ◦ indicate that CCFs were significantly better and worse respectively at the 10% level of a Wilcoxon signed rank test. Also shown is the average and standard deviation, best and worst for the error rate across the 179 classifiers (not including CCF and TREEBAGGER), and N_B and N_W the number of classifiers that were better and worse than CCFs respectively.

Dataset	K	N	D	CCF	TREEBAGGER		Average	Best	Worst	N_B	N_W
abalone	3	4177	8	**34.15 ± 1.53**	35.42 ± 0.99	•	39.91 ± 7.52	32.6	65.4	14	157
acute-inflammation	2	120	6	**0.00 ± 0.00**	**0.00 ± 0.00**		5.14 ± 12.97	0.0	50.9	0	50
acute-nephritis	2	120	6	**0.00 ± 0.00**	**0.00 ± 0.00**		4.04 ± 10.20	0.0	41.7	0	56
balance-scale	3	625	4	**8.81 ± 0.95**	15.87 ± 1.23	•	20.68 ± 17.84	1.0	92.8	23	151
balloons	2	16	4	**12.50 ± 12.50**	25.00 ± 17.68		36.79 ± 15.19	0.0	81.3	7	159
blood	2	748	4	**24.20 ± 3.47**	25.00 ± 1.75		24.17 ± 4.35	19.7	62.6	137	41
breast-cancer-wisc	2	699	9	**2.71 ± 1.17**	2.86 ± 0.99		6.54 ± 7.74	2.6	34.5	1	176
breast-cancer-wisc-diag	2	569	30	**2.46 ± 1.06**	3.87 ± 1.27		7.24 ± 8.14	1.8	37.3	10	162
breast-cancer-wisc-prog	2	198	33	**15.31 ± 1.77**	17.35 ± 2.28		24.53 ± 4.36	17.2	44.9	0	178
breast-tissue	6	106	9	**28.85 ± 11.05**	**28.85 ± 5.77**		39.08 ± 15.35	20.2	83.1	33	144
car	4	1728	6	**0.87 ± 0.60**	1.39 ± 0.33		14.01 ± 13.27	0.8	98.3	1	177
cardiotocography-10	10	2126	21	13.14 ± 1.54	**12.05 ± 1.15**	◦	31.68 ± 20.22	11.5	84.5	4	170
cardiotocography-3	3	2126	21	6.12 ± 0.21	**5.18 ± 0.54**	◦	12.82 ± 9.35	4.4	93.7	10	165
congressional-voting	2	435	16	**39.91 ± 2.29**	**39.91 ± 2.38**		39.58 ± 3.97	36.8	84.4	131	44
conn-bench-sonar	2	208	60	**13.94 ± 2.50**	14.90 ± 3.43		25.42 ± 8.74	9.6	53.4	12	166
conn-bench-vowel	11	462	11	**0.00**	1.30		30.02 ± 29.52	0.0	98.1	0	164
contrac	3	1473	9	49.18 ± 2.60	**48.37 ± 2.31**		50.41 ± 5.55	42.8	68.5	96	78
dermatology	6	366	34	2.75 ± 0.55	**2.47 ± 0.91**		13.81 ± 20.13	1.4	69.6	30	140
echocardiogram	2	131	10	17.42 ± 2.51	**15.15 ± 2.14**		18.49 ± 5.74	12.3	46.2	106	68
ecoli	8	336	7	14.58 ± 2.96	**12.50 ± 2.15**		22.39 ± 12.92	9.1	79.5	50	123
energy-y1	3	768	8	**2.86 ± 1.07**	3.26 ± 1.30		14.05 ± 14.03	2.2	94.5	6	170
energy-y2	3	768	8	**9.24 ± 1.30**	9.64 ± 0.94		15.74 ± 12.43	6.6	95.4	39	136
fertility	2	100	9	**11.00 ± 1.73**	**11.00 ± 1.73**		14.22 ± 5.65	10.0	54.0	1	174
glass	6	214	9	25.94 ± 1.56	**24.06 ± 5.72**		39.01 ± 11.25	21.5	68.1	14	163
haberman-survival	2	306	3	**29.61 ± 2.18**	33.55 ± 1.47		27.43 ± 3.63	22.9	51.4	151	26
heart-cleveland	5	303	13	**41.45 ± 3.89**	42.76 ± 2.87		44.04 ± 4.49	35.2	80.3	38	134
heart-hungarian	2	294	12	**16.10 ± 3.54**	18.84 ± 3.27	•	20.04 ± 5.15	13.4	36.1	25	153
heart-switzerland	5	123	12	**56.45 ± 3.61**	59.68 ± 2.79		62.00 ± 5.77	46.8	87.1	24	149
heart-va	5	200	12	**67.00 ± 3.00**	67.50 ± 4.56		68.19 ± 3.83	60.0	77.5	68	103
hepatitis	2	155	19	**17.95 ± 3.14**	18.59 ± 2.13		19.97 ± 5.96	10.3	69.2	54	123
hill-valley-noisy	2	606	100	**6.93**	50.83		45.91 ± 7.53	25.7	93.9	0	176
ilpd-indian-liver	2	583	9	29.79 ± 1.03	**29.28 ± 2.44**		30.08 ± 5.18	22.4	70.5	115	60
ionosphere	2	351	33	**5.11 ± 1.70**	7.10 ± 1.68		14.05 ± 9.58	4.5	64.2	3	173
iris	3	150	4	**2.70 ± 1.91**	4.73 ± 2.24		10.61 ± 17.57	0.7	99.3	29	147
led-display	10	1000	7	27.30 ± 1.77	**27.20 ± 1.47**		39.76 ± 22.15	25.2	91.2	37	135
lenses	3	24	4	**16.67 ± 16.67**	**16.67 ± 16.67**		26.02 ± 13.60	4.2	83.3	30	119
letter	26	20000	16	**2.27 ± 0.07**	3.55 ± 0.14	•	37.28 ± 32.22	2.6	96.1	0	163
libras	15	360	90	**11.39 ± 3.46**	20.00 ± 2.22	•	42.91 ± 25.55	10.8	93.4	1	176
low-res-spect	9	531	100	**7.52 ± 0.53**	8.08 ± 0.98		19.99 ± 12.21	6.6	62.0	1	176
magic	2	19020	10	**12.31 ± 0.19**	12.76 ± 0.26	•	19.98 ± 6.75	11.7	41.4	7	159

Dataset	K	N	D	CCF	TREEBAGGER		Average	Best	Worst	N_B	N_W
miniboone	2	130064	50	**6.13 ± 0.07**	6.32 ± 0.08	•	17.41 ± 15.01	6.2	71.6	0	132
musk-1	2	476	166	**11.55 ± 1.82**	11.97 ± 4.88		20.59 ± 9.65	6.3	56.6	29	148
musk-2	2	6598	166	2.96 ± 0.20	**2.29 ± 0.16**	○	7.50 ± 7.94	0.2	73.5	38	134
nursery	5	12960	8	**0.15 ± 0.02**	0.34 ± 0.12	•	16.15 ± 21.35	0.0	76.7	3	157
oocytes_m_nucleus_4d	0	1022	41	**15.59 ± 2.36**	22.45 ± 3.67	•	27.33 ± 10.82	14.0	91.7	2	175
oocytes_m_states_2f	0	1022	25	**6.76 ± 0.98**	7.75 ± 0.85	•	14.18 ± 12.19	6.0	99.7	1	176
oocytes_t_nucleus_2f	0	912	25	**14.91 ± 1.24**	19.41 ± 1.62	•	28.79 ± 12.16	13.2	90.2	5	171
oocytes_t_states_5b	0	912	32	**6.36 ± 1.30**	7.46 ± 0.44		15.58 ± 12.61	4.9	100.0	12	163
optical	10	1797	62	2.62	**2.56**		26.67 ± 29.52	1.3	90.1	5	163
ozone	2	2536	72	2.96 ± 0.13	**2.92 ± 0.08**		5.02 ± 6.92	2.6	48.9	84	72
page-blocks	5	5473	10	2.87 ± 0.35	**2.76 ± 0.39**		5.93 ± 5.55	2.5	59.7	20	148
parkinsons	2	195	22	**6.63 ± 3.02**	10.71 ± 2.65	•	14.94 ± 7.27	5.6	62.2	8	169
pendigits	10	3498	16	**3.20**	4.35		25.31 ± 28.39	2.2	89.7	16	157
pima	2	768	8	**24.48 ± 1.61**	25.39 ± 1.00		25.73 ± 4.05	21.0	41.8	85	87
planning	2	182	12	32.22 ± 3.69	**31.67 ± 1.84**		32.32 ± 5.57	27.2	58.8	123	55
plant-margin	100	1600	64	**11.06 ± 0.48**	14.44 ± 1.30	•	50.61 ± 31.22	12.8	99.0	0	163
plant-shape	100	1600	64	**24.31 ± 0.89**	34.94 ± 1.79	•	62.16 ± 23.43	27.7	99.4	0	161
plant-texture	100	1599	64	**14.19 ± 0.84**	16.63 ± 1.35	•	49.80 ± 30.84	13.4	99.1	2	156
ringnorm	2	7400	20	**2.24 ± 0.36**	4.38 ± 0.40	•	16.09 ± 14.33	1.3	50.5	25	149
seeds	3	210	7	5.29 ± 2.50	**4.81 ± 2.15**		14.02 ± 16.34	2.8	87.0	21	146
semeion	10	1593	256	4.96 ± 1.44	**4.65 ± 1.30**		30.74 ± 27.22	3.6	89.9	9	164
spambase	2	4601	57	**4.02 ± 0.34**	4.76 ± 0.22	•	12.89 ± 9.73	3.9	43.6	1	170
spect	2	186	22	31.72	**25.81**		42.88 ± 6.07	27.8	92.5	6	170
spectf	2	187	44	**8.02**	**8.02**		17.44 ± 21.91	7.5	94.1	4	128
statlog-image	7	2310	18	**1.39 ± 0.49**	2.04 ± 0.72	•	18.51 ± 25.22	1.4	85.9	0	176
statlog-landsat	6	2000	36	**9.10**	**9.10**		24.28 ± 19.65	8.1	78.9	6	166
statlog-shuttle	7	14500	9	0.03	**0.01**		6.61 ± 8.77	0.0	57.6	0	159
statlog-vehicle	4	846	18	**17.65 ± 2.71**	24.64 ± 1.74	•	34.61 ± 16.17	14.9	74.4	10	168
steel-plates	7	1941	27	**21.24 ± 0.87**	21.34 ± 0.83		36.40 ± 16.11	19.6	92.0	4	172
synthetic-control	6	600	60	**0.83 ± 0.73**	1.50 ± 1.09		18.19 ± 26.13	0.3	87.3	5	171
trains	2	10	29	**12.50 ± 21.65**	**12.50 ± 21.65**		31.89 ± 17.36	0.0	87.5	7	131
twonorm	2	7400	20	**2.11 ± 0.40**	2.72 ± 0.22	•	10.44 ± 13.71	2.0	50.1	2	145
vertebral-column-2	2	310	6	**16.23 ± 1.95**	**16.23 ± 3.84**		19.85 ± 6.46	12.6	67.8	57	120
vertebral-column-3	3	310	6	**13.96 ± 3.36**	15.91 ± 2.96		23.00 ± 11.63	12.6	67.8	7	170
wall-following	4	5456	24	2.93 ± 0.45	**0.46 ± 0.20**	○	21.40 ± 20.01	0.1	76.7	47	126
waveform	3	5000	21	**13.70 ± 0.93**	15.28 ± 0.99	•	23.99 ± 14.81	12.9	74.9	29	145
waveform-noise	3	5000	40	**12.78 ± 0.51**	13.88 ± 0.72	•	24.38 ± 14.75	12.6	76.3	1	174
wine	3	178	13	**0.00 ± 0.00**	1.14 ± 1.14		9.95 ± 16.15	0.0	98.3	0	176
wine-quality-red	6	1599	11	**29.94 ± 1.08**	30.88 ± 1.66		44.48 ± 12.26	31.0	98.1	0	177
wine-quality-white	7	4898	11	**30.29 ± 0.12**	31.58 ± 0.78		48.44 ± 12.39	30.9	98.2	0	173
yeast	10	1484	8	**37.60 ± 3.05**	37.80 ± 2.48		47.56 ± 10.20	36.3	70.3	3	173
zoo	7	101	16	**1.00 ± 1.73**	**1.00 ± 1.73**		13.49 ± 16.79	1.0	99.0	0	175

Table 18: Comparison of all classifiers on 82 UCI datasets. R is the mean rank according to error rate; E is the mean error rate (%); κ is the mean Cohen's κ Carletta (1996); E_{CCF} and κ_{CCF} are the respective values for CCFs on the datasets where the competing classifier successfully ran (note CCFs successfully ran on all datasets), N_v and N_l are the number of datasets where the CCFs κ was higher and lower than the classifier respectively, and p is the p-value for whether the CCFs κ mean is higher using a Wilcoxon signed ranks test.

Classifier	R	E	E_{CCF}	κ	κ_{CCF}	N_v	N_l	p
CCF	**28.87**	**14.08**	-	**70.67**	-	-	-	-
svmPoly_t	31.53	15.73	14.27	65.10	69.61	54	25	0.00023
svmRadialCost_t	31.84	15.33	14.27	66.55	69.61	43	36	0.11
svm_C	32	15.67	14.18	67.65	70.49	46	32	0.18
elm_kernel_m	32.19	15.20	14.54	69.01	69.75	42	36	0.16
parRF_m	33.03	15.54	14.08	67.73	70.67	52	27	0.01
svmRadial_t	33.77	15.68	14.27	65.88	69.61	50	28	0.0016
rf_caret	34.48	15.56	14.18	67.67	70.49	54	23	0.00063
rforest_R	40.70	15.82	14.18	66.67	70.49	57	20	2e-05
TreeBagger	40.91	15.75	14.08	67.51	70.67	55	23	3.5e-05
Bag_LibSVM_w	42.28	16.65	14.25	58.13	70.27	70	12	3.2e-12
C50_t	42.61	16.85	14.08	66.11	70.67	57	22	0.0004
nnet_t	42.87	18.74	14.08	64.72	70.67	54	26	0.0015
avNNet_t	43.26	18.77	14.08	64.88	70.67	50	29	0.001
RotationForest_w	44.62	16.64	14.08	65.34	70.67	64	15	7.1e-09
pcaNNet_t	45.86	19.28	14.08	63.83	70.67	54	25	0.00015
mlp_t	46.06	17.38	14.08	66.75	70.67	54	26	0.0016
LibSVM_w	46.50	16.65	14.08	63.80	70.67	57	21	2.9e-06
MultiBoostAB_LibSVM_w	46.90	16.82	14.25	64.47	70.27	61	17	1.8e-06
RRF_t	49.56	16.71	14.18	66.30	70.49	59	20	1.1e-05
adaboost_R	50.48	18.33	14.01	64.24	71.07	58	21	1.8e-06
RRFglobal_caret	51.88	16.83	14.18	65.97	70.49	58	20	2.7e-07
RandomForest_weka	52.66	16.75	14.24	63.42	69.64	62	15	1.6e-08
svmLinear_caret	53.14	17.96	14.27	61.51	69.61	60	19	1.3e-08
MultiBoostAB_RandomForest_weka	53.38	16.51	13.93	62.91	69.78	66	13	2.8e-09
gausspRadial_R	53.75	18.41	14.69	61.92	70.70	62	17	1.9e-09
MultiBoostAB_MultilayerPerceptron_weka	56.49	17.20	14.08	66.29	70.67	61	18	2.3e-06
pnn_matlab	56.56	18.03	14.36	62.98	70.13	66	13	2.2e-08
mda_caret	57.04	20.96	14.08	62.25	70.67	59	20	4.3e-07
cforest_caret	58.27	19.52	14.61	59.80	69.50	70	9	1.9e-11
svmlight_C	58.35	18.46	15.80	61.29	65.70	57	21	5.2e-07
mlp_C	58.50	18.47	14.08	64.35	70.67	67	13	9.4e-09
Decorate_weka	58.69	17.87	14.12	64.58	70.56	65	14	7.1e-07
Bagging_RandomForest_weka	59.75	17.39	14.09	58.82	67.12	66	12	5.8e-11
rbfDDA_caret	59.91	19.23	15.80	60.96	66.34	67	11	8.7e-11
MultiBoostAB_PART_weka	60.27	18.08	14.08	65.01	70.67	61	18	2e-06
dkp_C	60.57	17.93	14.08	45.14	70.67	66	12	1e-10
knn_caret	60.94	18.93	14.08	61.20	70.67	65	14	7.7e-09
MultilayerPerceptron_weka	61.97	18.00	14.08	64.79	70.67	64	15	1.2e-07
glmnet_R	62	19.35	14.08	61.25	70.67	60	17	7.1e-09
multinom_caret	62.04	19.77	14.08	61.20	70.67	65	15	2.1e-09
Bagging_PART_weka	62.45	18.43	14.08	63.97	70.67	63	16	7.1e-08
rda_R	62.63	18.40	14.08	62.68	70.67	57	20	1.9e-07
knn_R	62.64	18.21	14.08	62.91	70.67	63	16	1.5e-08

Classifier	R	E	E_{CCF}	κ	κ_{CCF}	N_v	N_l	p
fda_caret	63.32	19.05	14.08	62.65	70.67	67	13	2.7e-09
elm_matlab	63.79	18.86	14.08	61.64	70.67	67	14	2e-09
SMO_weka	63.94	18.41	14.08	62.63	70.67	64	16	2.5e-08
RandomCommittee_weka	64.11	18.19	14.08	63.62	70.67	63	16	4.5e-09
MultiBoostAB_J48_weka	65.09	18.70	14.18	63.78	70.49	65	14	2e-08
mlpWeightDecay_caret	65.72	22.12	14.18	59.53	70.49	64	16	4.1e-09
Bagging_RandomTree_weka	65.85	18.50	14.08	63.84	70.67	64	15	3.1e-09
pda_caret	66.80	19.59	14.08	61.04	70.67	62	17	1e-09
mlm_R	66.82	19.94	14.08	60.78	70.67	65	14	1.1e-08
ClassificationViaRegression_weka	67.05	19.50	14.08	61.09	70.67	66	14	8.8e-10
Bagging_J48_weka	67.42	19.02	14.08	62.90	70.67	66	13	2.1e-09
AdaBoostM1_J48_weka	69.14	18.42	14.18	64.49	70.49	63	16	8.7e-08
treebag_caret	69.31	18.82	14.18	62.82	70.49	65	15	3.2e-09
rbf_caret	69.49	22.06	14.08	58.67	70.67	65	15	6e-09
SimpleLogistic_weka	69.76	20.26	14.08	58.47	70.67	66	14	7.2e-11
fda_R	70.26	20.09	14.08	60.35	70.67	63	16	4.4e-09
ldaBag_R	70.31	20.21	14.08	60.00	70.67	62	16	2e-09
gcvEarth_caret	70.46	20.42	14.08	60.38	70.67	66	14	1.1e-10
lda_R	70.64	20.21	14.08	60.22	70.67	63	16	3.2e-09
lssvmRadial_caret	72.38	19.13	14.51	62.62	69.66	69	10	4e-09
LibLINEAR_weka	72.68	20.36	14.08	60.29	70.67	69	11	1.7e-11
lda2_caret	72.76	20.19	14.08	59.75	70.67	67	13	2.1e-10
Bagging_Logistic_weka	73.91	19.70	13.96	60.82	70.61	67	13	1.2e-10
MultiBoostAB_RandomTree_weka	74.09	19.04	14.08	62.81	70.67	69	11	2.8e-10
Logistic_weka	75.35	20.29	13.96	59.52	70.61	69	11	1.1e-11
MultiBoostAB_REPTree_weka	75.37	19.99	14.08	61.17	70.67	72	9	1.7e-11
END_weka	75.48	19.34	14.08	61.30	70.67	66	14	3.8e-10
Bagging_IBk_weka	75.65	19.74	14.08	60.75	70.67	74	7	2.2e-12
Bagging_LWL_weka	75.65	19.74	14.08	60.75	70.67	74	7	2.2e-12
Bagging_weka	75.65	19.74	14.08	60.23	70.67	74	7	1.7e-12
mda_R	76.31	20.04	14.08	59.93	70.67	68	13	1.5e-10
sda_caret	76.40	20.34	14.08	59.40	70.67	65	13	5e-11
MultiBoostAB_Logistic_weka	76.62	20.48	14.08	60.71	70.67	65	15	2e-10
svmBag_R	76.80	25.06	13.98	55.03	70.52	65	13	2.4e-09
RandomSubSpace_weka	78.20	20.71	14.08	56.89	70.67	74	5	7.3e-14
hdda_R	79.29	20.48	14.08	59.99	70.67	62	17	6e-09
lvq_caret	79.69	19.85	14.09	58.23	70.00	70	11	3.2e-11
pls_caret	79.98	24.08	14.89	54.92	68.93	68	12	9.4e-12
ctreeBag_R	80.06	21.01	14.27	56.26	69.82	72	8	7.7e-13
MultiClassClassifier_weka	81.43	21.75	14.08	58.42	70.67	68	12	2.1e-11
LogitBoost_weka	83.48	20.92	14.08	58.88	70.67	69	10	3.3e-11
C50Rules_caret	83.87	20.95	14.08	59.97	70.67	71	8	2.5e-11
JRip_caret	83.95	22.05	14.08	56.94	70.67	69	12	6.3e-12
PART_caret	84.03	20.95	14.08	57.65	70.67	71	9	1.2e-11
RBFNetwork_weka	85.81	20.79	14.14	54.81	69.84	71	7	2.5e-12
J48_caret	86.10	20.88	14.08	56.85	70.48	68	11	1e-11
C50Tree_caret	87.52	21.49	14.08	59.16	70.67	69	10	5.5e-12
IBk_weka	87.93	20.30	14.08	61.11	70.67	68	10	1.2e-10
qda_caret	89.09	21.96	14.18	55.45	70.47	71	11	8.4e-13
PART_weka	89.14	21.53	14.08	60.35	70.67	67	13	6.4e-10
IB1_weka	89.25	20.53	14.08	60.17	70.67	71	7	1.4e-11
KStar_weka	89.32	20.74	14.18	58.60	70.49	75	4	4.6e-13
NBTree_weka	89.73	21.41	14.08	58.91	70.67	69	10	2.7e-12

Classifier	R	E	E_{CCF}	κ	κ_{CCF}	N_v	N_l	p
J48_weka	89.94	21.75	14.18	59.29	70.49	70	10	8.1e-11
Bagging_DecisionTable_weka	91.38	22.73	14.18	55.73	70.49	75	4	7.3e-14
MultiBoostAB_DecisionTable_weka	91.52	23.78	14.18	55.37	70.49	72	8	9.5e-13
obliqueTree_R	92.02	23.77	14.14	55.43	69.84	71	8	6.9e-12
AttributeSelectedClassifier_weka	92.12	22.36	14.08	56.63	70.67	71	8	2e-12
NNge_weka	93.20	21.31	14.18	58.21	70.49	73	6	1.7e-13
bagging_R	94.07	31.45	14.18	49.22	70.49	70	11	1.7e-12
rbf_matlab	94.39	26.03	16.39	51.92	65.73	66	12	1.5e-11
DTNB_weka	96.04	22.10	14.08	56.56	70.67	72	7	3.2e-12
ctree2_caret	96.23	25.83	14.18	52.85	70.49	72	10	7.9e-12
lvq_R	96.51	27.97	14.08	52.58	70.67	72	7	2.5e-12
REPTree_weka	96.69	23.05	14.08	55.64	70.67	75	6	2.3e-13
rrlda_R	97.07	24.39	14.08	56.76	70.67	69	10	1e-10
cascor_C	97.54	23.07	14.08	57.12	70.67	71	9	7e-13
ctree_caret	98.09	24.20	14.33	53.72	69.96	73	8	5.6e-12
JRip_weka	98.14	23.08	14.18	56.73	70.49	73	8	3.4e-12
mlp_matlab	98.26	27.00	14.08	48.38	70.67	74	6	1e-13
OrdinalClassClassifier_weka	98.77	24.40	14.08	56.17	70.67	71	9	6.1e-12
nbBag_R	98.81	22.94	14.17	56.08	70.48	70	11	7.7e-12
Ridor_weka	99.05	21.65	14.14	56.21	69.84	72	8	5.3e-13
bdk_R	99.14	21.99	14.08	58.06	70.67	74	5	2.7e-13
Dagging_weka	99.29	24.46	14.08	50.08	70.67	76	4	3.5e-14
rpart_caret	99.90	25.03	14.08	53.93	70.67	74	8	3.4e-12
BayesNet_weka	100.64	23.16	14.08	55.20	70.67	69	10	8.1e-13
naiveBayes_R	101.61	23.50	14.08	56.55	70.67	70	9	4.7e-12
FilteredClassifier_weka	101.65	23.82	14.08	54.16	70.67	75	4	6.7e-14
plsBag_R	101.70	31.34	14.18	44.50	70.49	72	6	3.8e-13
rpart2_caret	102.34	24.06	14.08	55.35	70.48	72	10	1.8e-11
logitboost_R	102.60	24.16	14.08	64.70	70.67	67	12	1.5e-08
rpart_R	103.37	25.84	14.08	52.04	70.67	73	8	7.3e-13
slda_caret	105.58	26.13	14.08	48.75	70.67	76	3	4e-14
nnetBag_R	105.84	39.24	14.08	35.37	70.67	70	10	2.5e-12
pam_caret	107.69	25.52	14.08	46.49	70.67	78	2	1.2e-14
mars_R	108.10	35.05	14.18	49.44	70.49	73	7	1.9e-13
MultiBoostAB_NaiveBayes_weka	109.33	25.66	14.08	52.92	70.67	70	11	1.4e-12
RandomTree_weka	111.01	23.91	14.08	55.65	70.67	76	6	3.4e-14
sddaLDA_R	111.26	26.96	14.08	44.47	70.67	76	4	3.8e-14
simpls_R	112.09	37.47	14.08	42.78	70.67	71	9	3.6e-13
widekernelpls_R	112.30	36.93	14.73	41.31	69.36	71	9	2.4e-13
Bagging_NaiveBayes_weka	113.50	26.54	14.08	51.39	70.67	70	10	1.4e-12
NaiveBayes_weka	113.84	26.49	14.08	51.54	70.67	70	10	8e-13
stepQDA_caret	114.34	27.16	14.26	45.82	70.12	75	6	5.6e-14
DecisionTable_weka	114.41	27.07	14.08	50.31	70.67	76	4	3.1e-14
QdaCov_caret	114.59	26.47	14.28	50.35	70.29	77	5	9.3e-14
kernelpls_R	114.73	39.72	14.08	40.72	70.67	71	9	3.6e-13
sparseLDA_R	114.96	30.47	13.96	43.73	70.61	72	9	3.2e-13
NaiveBayesUpdateable_weka	115.50	27.73	14.08	51.54	70.67	70	10	8e-13
bayesglm_caret	116.20	42.40	14.08	34.14	70.67	72	6	4e-13
PenalizedLDA_R	116.24	32.62	14.08	42.79	70.67	66	13	2.6e-12
sddaQDA_R	116.75	29.97	14.08	41.12	70.67	77	4	4.4e-14
stepLDA_caret	117	27.88	14.08	43.57	70.48	78	3	1.3e-14
NaiveBayesSimple_weka	124.85	26.95	13.07	50.31	70.52	75	7	1.4e-13
glmStepAIC_caret	124.85	43.05	14.20	34.13	70.37	74	5	1.3e-13

Classifier	R	E	E_{CCF}	κ	κ_{CCF}	N_v	N_l	p
LWL_weka	126.43	30.60	14.18	42.89	70.49	75	4	6.1e-14
gpls_R	126.52	45.94	14.84	33.86	69.16	71	7	2.4e-13
dpp_C	127.03	31.59	14.08	49.22	70.67	68	12	3.3e-12
AdaBoostM1_weka	128.16	37.45	14.08	36.59	70.67	75	4	3.7e-14
glm_R	130.88	51.04	14.08	31.62	70.67	72	8	2.8e-13
Bagging_HyperPipes_weka	133.59	36.56	14.08	32.96	70.67	80	1	6.5e-15
MultiBoostAB_weka	133.60	38.57	14.08	33.58	70.67	76	3	2.2e-14
MultiBoostAB_IBk_weka	133.60	38.57	14.08	31.80	70.67	76	3	2e-14
MultiBoostAB_OneR_weka	133.72	35.98	14.08	36.85	70.67	78	2	8.8e-15
Bagging_OneR_weka	133.90	36.19	14.08	35.35	70.67	78	3	8.1e-15
VFI_weka	135.01	32.76	14.08	47.06	70.67	75	6	5e-14
Bagging_DecisionStump_weka	138.01	38.98	14.08	30.20	70.67	79	3	6.6e-15
OneR_caret	138.07	37.68	14.08	38.16	70.67	77	5	1.1e-14
HyperPipes_weka	139.46	39.62	14.08	31.01	70.67	80	2	5.5e-15
OneR_weka	139.71	37.78	14.08	34.57	70.67	79	3	7.9e-15
spls_R	140.37	46.35	14.08	19.96	70.67	78	4	9.2e-15
RacedIncrementalLogitBoost_weka	140.37	44.08	14.08	16.78	70.67	80	1	6.5e-15
DecisionStump_weka	140.85	40.81	14.08	27.77	70.67	79	3	6.8e-15
ConjunctiveRule_weka	140.92	41.12	14.08	28.71	70.67	79	3	8.5e-15
Bagging_MultilayerPerceptron_weka	143.11	46.63	14.08	16.33	70.67	77	3	1.8e-14
StackingC_weka	154.88	53.05	14.10	3.43	70.62	80	1	6.2e-15
CVParameterSelection_weka	154.90	53.05	14.10	3.45	70.62	80	1	6.2e-15
Grading_weka	154.90	53.05	14.10	3.45	70.62	80	1	6.2e-15
Stacking_weka	154.90	53.05	14.10	3.45	70.62	80	1	6.2e-15
MetaCost_weka	154.94	53.16	14.08	7.26	70.67	79	2	7e-15
CostSensitiveClassifier_weka	155.02	53.13	14.08	5.78	70.67	79	2	7.5e-15
MultiScheme_weka	155.02	53.13	14.08	2.74	70.67	80	1	6.2e-15
Vote_weka	155.02	53.13	14.08	2.74	70.67	80	1	6.2e-15
ZeroR_weka	155.02	53.13	14.08	2.74	70.67	80	1	6.2e-15
ClassificationViaClustering_weka	156.78	46.81	14.08	28.61	70.67	79	2	7.8e-15

References

Simon Bernard, Laurent Heutte, and Sébastien Adam. Influence of hyperparameters on random forest accuracy. In *MCS*, pages 171–180. Springer, 2009.

GÃŠrard Biau. Analysis of a random forests model. *Journal of Machine Learning Research*, 13 (Apr):1063–1095, 2012.

Åke Björck and Gene H Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of computation*, 27(123):579–594, 1973.

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

Magnus Borga. Canonical correlation: a tutorial. *On line tutorial http://people.imt.liu.se/magnus/cca*, 4, 2001.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

Jean Carletta. Assessing agreement on classification tasks: the kappa statistic. *Computational linguistics*, 22(2):249–254, 1996.

Claude Cohen and Adi Ben-Israel. On the computation of canonical correlations. *Cahiers Centre èEtudes Recherche Opèer*, 11:121–132, 1969.

Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision*, (7):81–227, 2011.

Fernando De la Torre. A least-squares framework for component analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(6):1041–1055, 2012.

Glenn De'Ath. Multivariate regression trees: a new technique for modeling species–environment relationships. *Ecology*, 83(4):1105–1117, 2002.

Persi Diaconis and Mehrdad Shahshahani. The subgroup algorithm for generating uniform random variables. *Probability in the engineering and informational sciences*, 1(01):15–32, 1987.

Haytham Elghazel, Alex Aussem, and Florence Perraud. Trading-off diversity and accuracy for optimal ensemble tree selection in random forests. In *Ensembles in Machine Learning Applications*, pages 169–179. Springer, 2011.

Jean Baptiste Faddoul, Boris Chidlovskii, Rémi Gilleron, and Fabien Torre. Learning multiple tasks with boosted decision trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 681–696. Springer, 2012.

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

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2):179–188, 1936.

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

João Gama. Functional trees. *Machine Learning*, 55(3):219–250, 2004.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006a.

Pierre Geurts, Louis Wehenkel, and Florence d'Alché Buc. Kernelizing the output of tree-based methods. In *Proceedings of the 23rd international conference on Machine learning*, pages 345–352. Acm, 2006b.

Ben Glocker, Olivier Pauly, Ender Konukoglu, and Antonio Criminisi. Joint classification-regression forests for spatially structured multi-object segmentation. *Computer Vision–ECCV 2012*, pages 870–881, 2012.

Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

Trevor Hastie, Andreas Buja, and Robert Tibshirani. Penalized discriminant analysis. *The Annals of Statistics*, pages 73–102, 1995.

Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

Tin Kam Ho. The random subspace method for constructing decision forests. 20:832–844, 1998.

Harold Hotelling. Relations between two sets of variates. *Biometrika*, pages 321–377, 1936.

Alston S Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM (JACM)*, 5(4):339–342, 1958.

Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

Tracy D Lemmond, Andrew O Hatch, Barry Y Chen, David Knapp, Lawrence Hiller, Marshall Mugge, and William G Hanley. Discriminant random forests. In *DMIN*, pages 55–61, 2008.

Jinyan Li and Huiqing Liu. Kent ridge bio-medical data set repository. *Institute for Infocomm Research. http://sdmc.lit.org.sg/GEDatasets/Datasets*, 2002.

M. Lichman. UCI machine learning repository, 2013.

Henrik Linusson. Multi-output random forests. 2013.

Wei-Yin Loh. Regression tress with unbiased variable selection and interaction detection. *Statistica Sinica*, pages 361–386, 2002.

Wei-Yin Loh and Yu-Shan Shih. Split selection methods for classification trees. *Statistica sinica*, 7 (4):815–840, 1997.

Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

Dragos D Margineantu and Thomas G Dietterich. Pruning adaptive boosting. In *ICML*, volume 97, pages 211–218, 1997.

Bjoern H Menze, B Michael Kelm, Daniel N Splitthoff, Ullrich Koethe, and Fred A Hamprecht. On oblique random forests. In *Machine Learning and Knowledge Discovery in Databases*, pages 453–469. Springer, 2011.

Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 2005.

Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *arXiv preprint cs/9408103*, 1994.

Carlos Pardo, José F Diez-Pastor, César García-Osorio, and Juan J Rodríguez. Rotation forests for regression. *Applied Mathematics and Computation*, 219(19):9914–9924, 2013.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

C Radhakrishna Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203, 1948.

Marko Robnik-Šikonja. Improving random forests. In *Machine Learning: ECML 2004*, pages 359–370. Springer, 2004.

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

Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

Ferdinando S Samaria and Andy C Harter. Parameterisation of a stochastic model for human face identification. In *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pages 138–142, 1994.

Mark Segal and Yuanyuan Xiao. Multivariate random forests. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):80–87, 2011.

Peter Sollich and Anders Krogh. Learning with ensembles: How overfitting can be useful. *Advances in neural information processing systems*, pages 190–196, 1996.

Liang Sun, Shuiwang Ji, and Jieping Ye. A least squares formulation for canonical correlation analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1024–1031. ACM, 2008.

Tyler M Tomita, Mauro Maggioni, and Joshua T Vogelstein. Randomer forests. *arXiv preprint arXiv:1506.03410*, 2015.

Tyler M Tomita, Mauro Maggioni, and Joshua T Vogelstein. Roflmao: Robust oblique forests with linear matrix operations. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 498–506. Society for Industrial and Applied Mathematics, 2017.

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. *Data mining and knowledge discovery handbook*, pages 667–685, 2010.

Michael Wainberg, Babak Alipanahi, and Brendan J Frey. Are random forests truly the best classifiers? *The Journal of Machine Learning Research*, 17(1):3837–3841, 2016.

G Yule. On the association of attributes in statistics, volume a 194. *Phil. Trans*, pages 257–319, 1900.

Heping Zhang. Classification trees for multiple binary responses. *Journal of the American Statistical Association*, 93(441):180–193, 1998.

Le Zhang and Ponnuthurai Nagaratnam Suganthan. Random forests with ensemble of feature spaces. *Pattern Recognition*, 47(10):3429–3437, 2014.