Journal of Machine Learning Research 17 (2016) 1-26 Submitted 2/14; Revised 3/15; Published 3/16

Random Rotation Ensembles

Rico Blaser R.Blaser@lse.ac.uk
Piotr Fryzlewicz P.Fryzlewicz@lse.ac.uk
Department of Statistics
London School of Economics
Houghton Street
London, WC2A 2AE, UK

Editor: Charles Elkan

Abstract

In machine learning, ensemble methods combine the predictions of multiple base learners to construct more accurate aggregate predictions. Established supervised learning algorithms inject randomness into the construction of the individual base learners in an effort to promote diversity within the resulting ensembles. An undesirable side effect of this approach is that it generally also reduces the accuracy of the base learners. In this paper, we introduce a method that is simple to implement yet general and effective in improving ensemble diversity with only modest impact on the accuracy of the individual base learners. By randomly rotating the feature space prior to inducing the base learners, we achieve favorable aggregate predictions on standard data sets compared to state of the art ensemble methods, most notably for tree-based ensembles, which are particularly sensitive to rotation.

Keywords: feature rotation, ensemble diversity, smooth decision boundary

1. Introduction

Modern statistical learning algorithms combine the predictions of multiple base learners to form ensembles, which typically achieve better aggregate predictive performance than the individual base learners (Rokach, 2010). This approach has proven to be effective in practice and some ensemble methods rank among the most accurate general-purpose supervised learning algorithms currently available. For example, a large-scale empirical study (Caruana and Niculescu-Mizil, 2006) of supervised learning algorithms found that decision tree ensembles consistently outperformed traditional single-predictor models on a representative set of binary classification tasks. Data mining competitions also frequently feature ensemble learning algorithms among the top ranked competitors (Abbott, 2012).

Achieving a good balance between the accuracy of the individual predictors and the diversity of the full ensemble is of critical importance: if the individual predictors are accurate but highly correlated, the benefits of combining them are modest; injecting randomness into the predictors reduces the correlation and promotes diversity but often does so at the expense of reduced accuracy for the individual predictors (Elghazel et al., 2011). A number of techniques have been devised to manage this trade-off and to promote diversity in learning ensembles in a constructive fashion; some methods merely perturb the training data, while others modify the internal structure of the predictors themselves. We now mention

some key examples. In bootstrap aggregation (Breiman, 1996), bootstrap replicates are used to construct multiple versions of a base predictor, which are subsequently aggregated via averaging or majority vote. This approach was found to be particularly effective for predictor classes that are unstable, in the sense that small variations of the input data lead to the construction of vastly different predictors (Hastie et al., 2009). Output smearing or flipping (Breiman, 2000) adds a different noise component to the dependent variable of each base predictor, which has a smoothing effect on the resulting decision boundary, leading to improved generalization performance. Boosting (Freund and Schapire, 1996) is an iterative procedure, where base learners are added sequentially in a forward stagewise fashion. By reweighting the data set at each iteration, later base learners are specialized to focus on the learning instances that proved the most challenging to the existing ensemble. In contrast to bootstrap aggregation, where each bootstrap sample is generated independently, boosting therefore does not lend itself naturally to parallel processing. Random decision forests (Ho, 1995, 1998) randomly select a feature subspace a priori and train a base learner in the chosen subspace using all available data. Instead of randomizing the training data, the structure of each predictor is altered by only including the chosen subset of predictors. Random forests (Breiman, 1999, 2001) combine bootstrap aggregation with the random projection method. At each tree node, a subset of the available predictors is randomly selected and the most favorable split point is found among these candidate predictors. This approach differs from random decision forests, where the selection of predictors is only performed once per tree. More generally, the framework also offers the possibility of using random linear combinations of two or more predictors. A summary of recent enhancements and applications of random forests can be found in Fawagreh et al. (2014). Perfect random tree ensembles (Cutler and Zhao, 2001), extremely random trees / extra trees (Geurts et al., 2006), and completely random decision trees (Liu et al., 2005; Fan et al., 2006) take randomization even further by not only selecting random predictor(s), as in random forests, but by also selecting a random split point, sometimes deterministically chosen from a small set of random candidate split points.

Some of the ensemble methods described specifically require the base learners to be decision trees. This is because decision trees are efficient to create (by recursive binary splitting), the models are straightforward to aggregate, and the individual trees can easily be turned into weak learners (which perform only slightly better than random) by restricting their depth (Kuhn and Johnson, 2013). Furthermore, decision trees exhibit a high variance and this inherent instability is beneficial to the diversity of the ensemble. In addition, decision trees contain a number of desirable features for general purpose data mining, including robustness to outliers and an ability to handle input variables of mixed type and scale, such as continuous and categorical variables, and even missing values (Hastie et al., 2009). However, a decision tree is merely an efficient representation for a set of hyper-rectangles that partition the decision space. For ordinary decision trees, each hyper-rectangle is aligned with at least one of the axes of the chosen coordinate system, resulting in axis parallel decision boundaries. This results in very characteristic piecewise constant stair shapes, even when the number of trees in the ensemble is large, as can be observed visually in low dimensional graphical examples. As a consequence, a much greater number of trees is needed to accurately approximate an oblique decision boundary than a decision boundary that is axis aligned with standard tree ensembles. In order to overcome this limitation, nonlinear

boosting projections (Garca-Pedrajas et al., 2007) provide a different, nonlinear view of the data to each base learner and oblique random forests (Menze et al., 2011) use linear discriminative models or ridge regression to select optimal oblique split directions at each tree node. Another approach that is related to but different from the method proposed in the present paper is embodied by rotation forests (Rodriguez et al., 2006; Kuncheva and Rodriguez, 2007), which take a subset of features and a bootstrap sample of the data and perform a principal component analysis (PCA), rotating the entire feature space before building the next base predictor. In addition to PCA, Kuncheva and Rodriguez (2007) experimented with nonparametric discriminate analysis (NDA) and sparse random projections and in De Bock and Poel (2011), independent component analysis (ICA) is found to yield the best performance.

The premise of the present paper is that it makes sense to rotate the feature space in ensemble learning, particularly for decision tree ensembles, but that it is neither necessary nor desirable to do so in a structured way. This is because structured rotations reduce diversity. Instead, we propose to rotate the feature space randomly before constructing the individual base learners. The random rotation effectively generates a unique coordinate system for each base learner, which we show increases diversity in the ensemble without a significant loss in accuracy. In addition to rotation, affine transformations also include translation, scaling, and shearing (non-uniform scaling combined with rotation). However, only transformations involving rotation have an impact on base learners that are insensitive to monotone transformations of the input variables, such as decision trees. Furthermore, a key difference between random rotation and random projection is that rotations are reversible, implying that there is no loss of information.

The remainder of this paper is structured as follows. Section 2 provides a motivational example for the use of random rotations using a well-known data set. In Section 3 we formally introduce random rotations and provide guidance as to their construction. Section 4 evaluates different application contexts for the technique and performs experiments to assesses its effectiveness. Conclusions and future research are discussed in Section 5. It is our premise that random rotations provide an intuitive, optional enhancement to a number of existing machine learning techniques. For this reason, we provide random rotation code in C/C++ and R in Appendix A, which can be used as a basis for enhancing existing software packages.

2. Motivation

Figure 1 motivates the use of random rotations on the binary classification problem from chapter 2 of Hastie et al. (2009). The goal is to learn the decision boundary, which separates the two classes, from a set of training points. In this example, the training data for each class came from a mixture of ten low-variance Gaussian distributions, with individual means themselves distributed as Gaussian. Since the data is artificially generated, the optimal decision boundary is known by construction.

In this motivational example, we compare two approaches: (1) a standard random forest classifier and (2) a random forest classifier in which each tree is generated on a randomly rotated feature space. It is evident that the random feature rotation has a significant impact on the resulting data partition: despite using the same sequence of random numbers

in the tree induction phase of the random forest algorithm – resulting in the same bootstrap samples and related feature subset selections at each decision branch for the two trees – the resulting tree is not merely a rotated version of the unrotated tree but is, in fact, a very different tree altogether, with a different orientation and a vastly different data partition. This demonstrates the power of the method; diversity is achieved with only a modest loss of information. However, the real benefit is illustrated on the bottom row of Figure 1 and arises from the aggregation of multiple randomly rotated trees. The rotated ensemble exhibits a visibly smoother decision boundary and one that is very close to optimal for this problem. The decision boundary is uncharacteristically smooth for a tree ensemble and is reminiscent of a kernel method, such as a k-nearest neighbor method or a support vector machine. In contrast, even with 10000 trees, the decision boundary for the standard random forest is still notably rectangular shaped. Another striking feature of the random rotation ensemble is the existence of a nearly straight diagonal piece of the decision boundary on the far left. This would be difficult to achieve with an axis-parallel base learner without rotation and it agrees well with the true decision boundary in this example.

3. Random Rotations

In this section, we formally introduce random rotations and describe two practical methods for their construction.

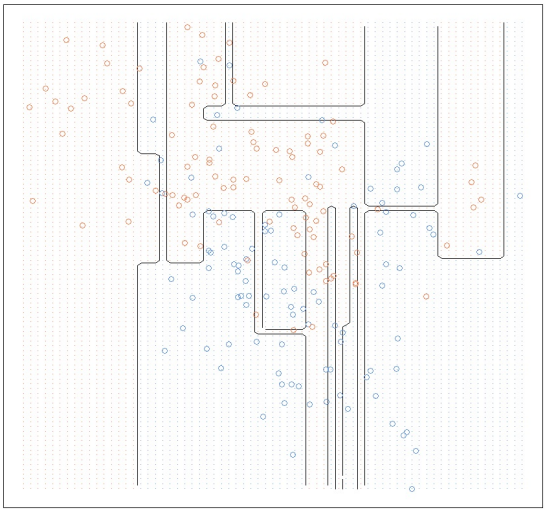
A (proper) rotation matrix R is a real-valued $n \times n$ orthogonal square matrix with unit determinant, that is

$$R^T = R^{-1} \text{ and } |R| = 1. \tag{1}$$

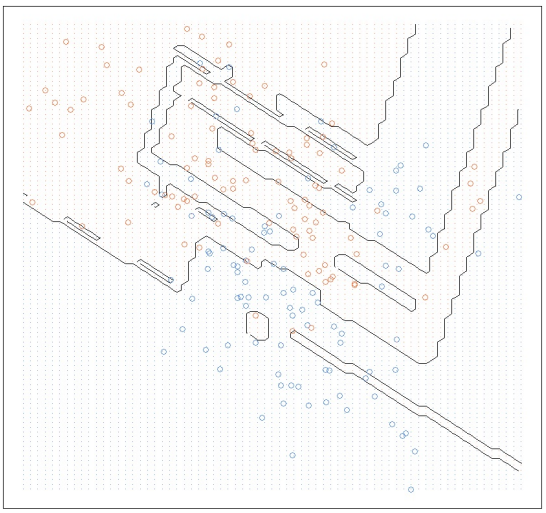
Using the notation from Diaconis and Shahshahani (1987), the set of all such matrices forms the special orthogonal group $SO(n)$, a subgroup of the orthogonal group $O(n)$ that also includes so-called improper rotations involving reflections (with determinant -1). More explicitly, matrices in $SO(n)$ have determinant $|R| = 1$, whereas matrices in $O(n)$ may have determinant $|R| = d$, with $d \in \{-1, 1\}$. Unless otherwise stated, the notation $O(n)$ always refers to the orthogonal group in this paper and is not related to the Bachman-Landau asymptotic notation found in complexity theory.

In order to perform a random rotation, we uniformly sample over all feasible rotations. Randomly rotating each angle in spherical coordinates does not lead to a uniform distribution across all rotations for $n > 2$, meaning that some rotations are more likely to be generated than others. It is easiest to see this is in 3 dimensions: suppose we take a unit sphere, denoting the longitude and latitude by the two angles $\lambda \in [-\pi, \pi]$ and $\phi \in [-\pi/2, \pi/2]$. If we divide the surface of this sphere into regions by dividing the two angles into equal sized intervals, then the regions closer to the equator ($\phi = 0$) are larger than the regions close to the poles ($\phi = \pm\pi/2$). By selecting random angles, we are equally likely to arrive in each region but due to the different sizes of these regions, points tend to cluster together at the poles. This is illustrated for $n = 3$ in Figure 2, where the undesirable concentration of rotation points near the two poles is clearly visible for the naive method. In this illustration, the spheres are tilted to better visualize the areas near the poles.

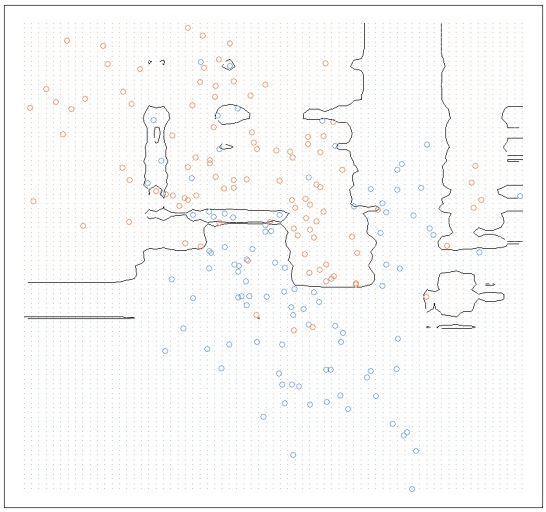
The group $O(n)$ does have a natural uniform distribution called the Haar measure, which offers the distribution we need. Using the probabilistic notation from Diaconis and Shahshahani (1987), the random matrix R is said to be uniformly distributed if $P(R \in U) =$



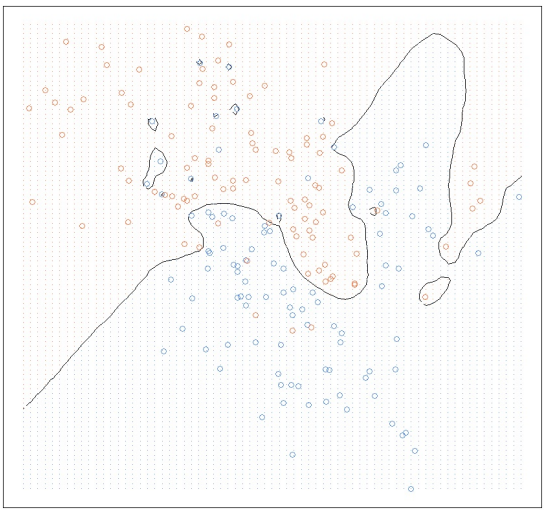
(a) **RF** (ntree=1, mtry=1)



(b) **RR-RF** (ntree=1, mtry=1)

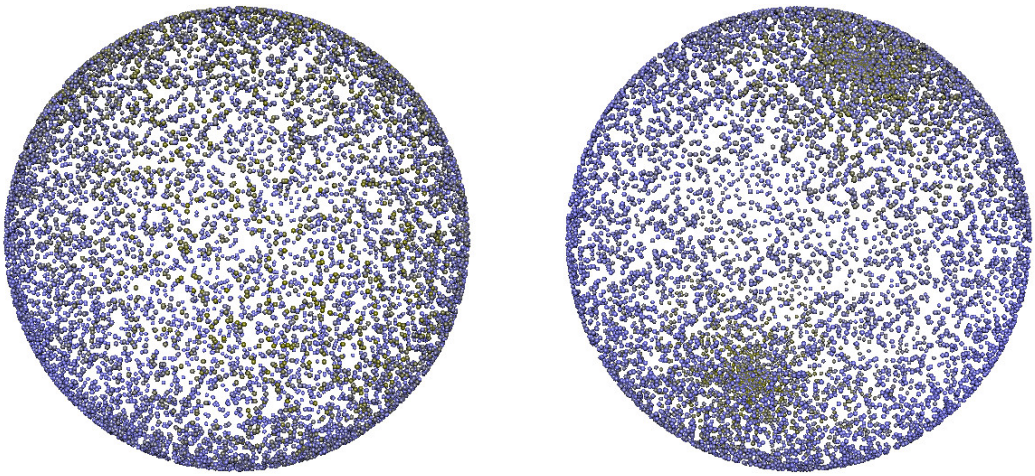


(c) **RF** (ntree=10000, mtry=1)



(d) **RR-RF** (ntree=10000, mtry=1)

Figure 1: Comparison of the decision boundary for the standard random forest algorithm (RF, left column) and the modified version with randomly rotated feature space for each tree (RR-RF, right column) on the binary classification task of chapter 2 of Hastie et al. (2009). The top row illustrates a typical decision boundary for a single tree, while the bottom row depicts a fully grown ensemble comprised of 10000 trees in each case. Ntree is the total number of trees in the forest, mtry the number of randomly selected features considered at each decision node.



(a) Uniformly random rotations (b) Naive random rotations

Figure 2: Comparison of correctly executed uniformly random rotation (left) in three dimensions versus naive method of selecting two random angles in spherical coordinates (right). 10000 random rotations of the same starting vector were generated for each method and distances were computed between each pair of rotations to produce a rank-based gradient, with green dots representing those vectors with the lowest sums of distances.

$P(R \in \Gamma U)$ for every $U \subset O(n)$ and $\Gamma \in O(n)$. Several algorithms exist to generate random orthogonal matrices distributed according to the Haar measure over $O(n)$, some of which are documented in Anderson et al. (1987); Diaconis and Shahshahani (1987); Mezzadri (2007); Ledermann and Alexander (2011). We will focus on two basic approaches to illustrate the concept.

1. (*Indirect Method*) Starting with an $n \times n$ square matrix A, consisting of n^2 independent univariate standard normal random variates, a Householder QR decomposition (Householder, 1958) is applied to obtain a factorization of the form $A = QR$, with orthogonal matrix Q and upper triangular matrix R with positive diagonal elements. The resulting matrix Q is orthogonal by construction and can be shown to be uniformly distributed. In other words, it necessarily belongs to $O(n)$. Unfortunately, if Q does not feature a positive determinant then it is not a proper rotation matrix according to definition (1) above and hence does not belong to $SO(n)$. However, if this is the case then we can flip the sign on one of the (random) column vectors of A to obtain A^+ and then repeat the Householder decomposition. The resulting matrix Q^+ is identical to the one obtained earlier but with a change in sign in the corresponding column and $|Q^+| = 1$, as required for a proper rotation matrix.

2. (*Direct Method*) A second method of obtaining random rotations involves selecting random points on the unit n-sphere directly (Knuth, 1997). This can be accom-

plished by drawing n independent random normal $N(0,1)$ variates $\{v_1, v_2, ..., v_n\}$ and normalizing each by the square root of the sum of squares of all n variates, that is, $x_i = v_i/\sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$ for $i \in \{1, 2, \ldots, n\}$. In other words, x is a unit vector pointing to a random point on the n-sphere. This construction takes advantage of spherical symmetry in the multivariate Normal distribution. The method is asymptotically faster than the QR approach and an implementation is available in the GNU Scientific Library (Galassi, 2009) as function gsl_ran_dir_nd. It should also be noted that it is straightforward to obtain the individual rotation angles from the random vector x, which makes it possible to move from the (more compact) random vector notation to the random rotation matrix used in the indirect method.

Generating random rotations in software for problems involving fewer than 1000 dimensions is straightforward and fast, even using the simple algorithm described above. Listings 2 and 3 in Appendix A provide examples of the indirect method in C++ and R respectively, both presented without error checking or optimizations. The C++ code takes less than 0.5 seconds on a single core of an Intel Xeon E5-2690 CPU to generate a 1000x1000 random rotation matrix. It uses the Eigen template library (Guennebaud et al., 2010) and a Mersenne Twister (Matsumoto and Nishimura, 1998) pseudorandom number generator. Larger rotation matrices can be computed with GPU assistance (Kerr et al., 2009) and may be pre-computed for use in multiple applications. In addition, for problems exceeding 1000 dimensions it is practical to only rotate a random subset of axes in order to reduce the computational overhead. We recommend that a different random subset is selected for each rotation in this case.

For categorical variables, rotation is unnecessary and ill defined. Intuitively, if a category is simply mapped to a new rotated category, there is no benefit in performing such a rotation.

4. Experiments

Random rotations complement standard learning techniques and are easily incorporated into existing algorithms. In order to examine the benefit of random rotations to ensemble performance, we modified three standard tree ensemble algorithms to incorporate random rotations before the tree induction phase. The necessary modifications are illustrated in pseudo code in Listing 1 below.

All methods tested use classification or regression trees that divide the predictor space into disjoint regions G_j, where $1 \leq j \leq J$, with J denoting the total number of terminal nodes of the tree. Extending the notation in Hastie et al. (2009), we represent a tree as

$$T(x; \theta, \Omega) = \sum_{j=1}^{J} c_j I(R(x) \in G_j), \tag{2}$$

with optimization parameters $\Omega = \{G_j, c_j\}_1^J$, random parameters $\theta = \{R, \omega\}$, where R is the random rotation associated with the tree and ω represents the random sample of (x, y) pairs used for tree induction; $I(\cdot)$ is an indicator function. Each randomly rotated input $R(x)$ is thus mapped to a constant c_j, depending on which region G_j the input belongs to.

For regression, c_j is typically just the average or median of all y_j in region G_j. If we let $|G_j|$ denote the cardinality of G_j, this can be written as

$$c_j = \frac{1}{|G_j|} \sum_{R(x_k) \in G_j} y_k. \tag{3}$$

For classification trees, one of the modes is typically used instead.

Given a loss function $L(y_i, f(x_i))$, for example exponential loss for classification or squared loss for regression, a tree-induction algorithm attempts to approximate the optimization parameters for which the overall loss is minimized, that is

$$\hat{\Omega} = \arg\min_{\Omega} \sum_{j=1}^{J} \sum_{R(x_i) \in G_j} L(y_i, f(R(x_i))) = \arg\min_{\Omega} \sum_{j=1}^{J} \sum_{R(x_i) \in G_j} L(y_i, c_i). \tag{4}$$

This optimization is performed across all parameters Ω but the rotation is explicitly excluded from the search space ($R \in \theta$, but $R \notin \Omega$) because we are advocating a random rotation in this paper. However, conceptually it would be possible to include the rotation in the optimization in an attempt to focus on the most helpful rotations.

Listing 1: Testing Random Rotations (Pseudo Code)

```
Inputs: – training feature matrix X
        – testing feature matrix S
        – total number of classifiers M
        – standard base learner B
        – aggregation weights w

(A) Scale or rank numeric predictors x (Sect. 4.2):
    e.g. x' := (x − Q_k(x))/(Q_{1−k}(x) − Q_k(x))

(B) For m ∈ {1, 2, ..., M} do
  (1) generate random parameters: θ_m := {R_m, ω_m}
  (2) train standard learner B on R_m(x):
        Ω̂_m = arg min_Ω Σ_{j=1}^{J} Σ_{R(x_i)∈G_j} L(y_i, f(R_m(x_i)))
  (3) compute test or out−of−bag predictions
        T(x, θ_m, Ω_m), x ∈ R_m(S)

(C) Aggregate predictions (vote or average)
      f_M(x) = Σ_{m=1}^{M} w_m T(x; θ_m, Ω_m)
```

In all algorithms considered in this paper, the tree-induction is performed using standard greedy, top-down recursive binary partitioning. This approach will generally not arrive at the globally optimal solution to (4) but constructs a reasonable approximation quickly (Hastie et al., 2009).

The (regression) tree ensemble $f_M(x)$ can then be written as a weighted sum of the individual trees, that is

$$f_M(x) = \sum_{m=1}^{M} w_m T(x;\theta_m, \Omega_m), \tag{5}$$

where M denotes the total number of trees in the ensemble. For classification ensembles, a vote is typically taken instead. It should be noted that a separate rotation is associated with each tree in this notation but the same rotation could theoretically be associated with an entire group of trees. In particular, we can recover the standard setting without random rotation by setting R_m to the identity rotation for all m.

The difference between non-additive ensemble methods like random forests (Breiman, 2001) or extra trees (Geurts et al., 2006) and additive ensembles like boosted trees (Freund and Schapire, 1996) arises in the formulation of the joint model for multiple trees. As we will see, this difference makes testing random rotation with existing additive ensemble libraries much more difficult than with non-additive ensemble libraries. Specifically, random forests and extra trees place an equal weight of $w_m = 1/M$ on each tree, and trees are constructed independently of each other, effectively producing an average of M independent predictions:

$$\hat{\Omega}_m = \arg\min_{\Omega_m} \sum_{j=1}^{J} \sum_{R(x_i)\in G_j} L(y_i, T(x_i;\theta_m, \Omega_m)). \tag{6}$$

In contrast, boosted trees use $w_m = 1$ and each new tree in the sequence is constructed to reduce the residual error of the full existing ensemble $f_{m-1}(x)$, that is

$$\hat{\Omega}_m = \arg\min_{\Omega_m} \sum_{j=1}^{J} \sum_{R(x_i)\in G_j} L(y_i, f_{m-1}(x_i) + T(x_i;\theta_m, \Omega_m)). \tag{7}$$

There are other differences between the two approaches: for example, J, the number of leaf nodes in each tree is often kept small for boosting methods in order to explicitly construct weak learners, while non-additive methods tend to use large, unpruned trees in an effort to reduce bias, since future trees are not able to assist in bias reduction in this case.

We mainly focus on random forest and extra tree ensembles in this paper because both of these algorithms rely on trees that are constructed independently of each other. This provides the advantage that the original tree induction algorithm can be utilized unmodified as a black box in the rotated ensemble, ensuring that any performance differences are purely due to the proposed random rotation and are not the result of any subtle differences (or dependencies) in the construction of the underlying trees.

4.1 Data Sets & Preprocessing

For our comparative study of random rotation, we selected UCI data sets (Bache and Lichman, 2013) that are commonly used in the machine learning literature in order to make the results easier to interpret and compare. Table 5 in Appendix C summarizes the data sets, including relevant dimensional information.

Some algorithms tested were not able to handle categorical input variables or missing values and we performed the following automatic preprocessing steps for each data column:

1. Any column (predictors or response) with at least 10 distinct numeric values was treated as numeric and missing values were imputed using the column median.

2. Any column with fewer than 10 distinct values (numeric or otherwise) or with mostly non-numeric values was treated as categorical, and a separate category was explicitly created for missing values.

3. Categorical predictors with C categories were converted into $(C-1)$ 0/1 dummy variables, with the final dummy variable implied from the others to avoid adding multicollinearity.

Note that after evaluating these three rules, all predictors were either numeric without missing values or categorical dummy variables, with a separate category for missing values.

4.2 Variable Scaling

Rotation can be sensitive to scale in general and outliers in particular. In order to avoid biasing the results, we tested three different scaling methods, all of which only use in-sample information to calibrate the necessary parameters for out-of-sample scaling:

1. (*Basic Scaling*) Numeric values were scaled to $[0,1]$ using the in-sample min and max values, that is $x' = \min(1, \max(0, (x - \min(x_{is}))/(\max(x_{is}) - \min(x_{is}))))$. This scaling method deals with scale but only avoids out-of-sample outliers. Outliers are dealt with in a relatively crude fashion by applying a fixed cutoff.

2. (*Quantile Scaling*) Numeric values were linearly scaled in such a way that the 5th and 95th percentile of the in-sample data map to 0 and 1 respectively, that is $x' = (x - Q_5(x_{is}))/(Q_{95}(x_{is}) - Q_5(x_{is}))$. In addition, any values that exceed these thresholds were nonlinearly winsorized by adding/subtracting $0.01 \times \log(1 + \log(1 + \Delta))$, where Δ is the absolute difference to the in-sample bounds $Q_5(x_{is})$ or $Q_{95}(x_{is})$. This robust scaling has a breakdown point of 5% and maintains the order of inputs that exceed the thresholds.

3. (*Relative Ranking*) In-sample numeric values v_i were augmented with $\{-\infty, +\infty\}$ and ranked as $R(v_i)$, such that $R(-\infty)$ maps to 0 and $R(+\infty)$ maps to 1. Out-of-sample data was ranked relative to this in-sample map. To accomplish this, the largest v_i is found that is smaller or equal to the out-of-sample data v_o (call it v_i^{max}) and the smallest v_i is found that is greater or equal to the out-of-sample data v_o (v_i^{min}). The out-of-sample rank is then $0.5 \times R(v_i^{min}) + 0.5 \times R(v_i^{max})$. In this way, test elements that match in-sample elements obtain the same rank, test elements that fall in between two elements obtain a rank in between, and because the in-sample values are augmented with infinity, it is never possible to encounter test elements that cannot be mapped. This is the most robust approach to outliers that was tested.

4.3 Method & Evaluation

In order to collect quantitative evidence of the effect of random rotations, we built upon the tree induction algorithms implemented in the widely used R packages *randomForest* (Liaw and Wiener, 2002) and *extraTrees* (Simm and de Abril, 2013). For comparison, we also used the following implementation of rotation forests: https://github.com/ajverster/RotationForest.

Prior to the tests, all data sets were preprocessed and scaled using each of the three techniques described in the previous section (basic scaling, quantile scaling, and ranking).

Random forest and extra trees were tested with and without random rotation (for each scaling), while rotation forests included their own deterministic PCA rotation (Rodriguez et al., 2006) but were also run for each scaling method. Random rotations were tested with and without flip rotations. The combination of tree induction algorithms, scalings, and rotation options resulted in a total of 21 distinct experiments per data set.

For each experiment we performed a random 70-30 split of the data; 70% training data and the remaining 30% served as testing data. The split was performed uniformly at random but enforcing the constraint that at least one observation of each category level had to be present in the training data for categorical variables. This constraint was necessary to avoid situations, where the testing data contained category levels that were absent in the training set. Experiments were repeated 100 times (with different random splits) and the average performance was recorded.

In all cases we used default parameters for the tree induction algorithms, except that we built 5000 trees for each ensemble in the hope of achieving full convergence.

To evaluate the performance of random rotations, we ranked each method for each data set and computed the average rank across all data sets. This allowed us to compare performance of each method across scaling methods and tree induction algorithms in a consistent, nonparametric fashion. In addition, we determined the number of data sets for which each method performed within one cross-sectional standard deviation of the best predictor in order to obtain a measurement of significance. This approach is advocated in (Kuhn and Johnson, 2013) and it can be more informative when there is a cluster of strong predictors that is distinct from the weaker predictors and which would not get detected by simple ranking.

4.4 Results

Table 6 in Appendix C displays the raw results of the detailed testing. As indicated in the previous section, we opted to compare ranks – with low ranks indicating better performance – in order to avoid the problem of comparing problems of different difficulty or comparing regression with classification problems. This is illustrated in Table 1.

	B-SCALE					Q-SCALE					RANKED				
NAME	**rf**	**rrrf**	**et**	**rret**	**rot**	**rf**	**rrrf**	**et**	**rret**	**rot**	**rf**	**rrrf**	**et**	**rret**	**rot**
anneal	11	7	**1**	**6**	14	10	11	**1**	**1**	9	8	11	**5**	**1**	15
audiology	**10**	**8**	**1**	**6**	13	**9**	**7**	**5**	**1**	14	12	**10**	**3**	**3**	15
balance	6	8	14	14	**1**	7	4	13	10	**2**	8	5	10	12	**2**
breast-w	**4**	**1**	9	**4**	14	**2**	**4**	9	8	15	**7**	**3**	9	9	13
breast-y	**1**	**6**	9	10	15	**1**	**4**	10	8	12	**5**	**1**	7	12	14
chess	10	7	**5**	**2**	13	9	11	**4**	**3**	15	12	8	**1**	**6**	13
cleveland	**6**	**2**	10	**5**	14	**8**	**4**	9	**1**	15	10	**7**	10	**2**	13
credit-a	**3**	9	13	15	12	**1**	**5**	11	14	8	**1**	**6**	**7**	10	**4**
flare	**1**	**1**	10	11	**1**	**1**	**1**	13	13	**1**	**1**	**1**	11	15	**1**
glass	**1**	12	**4**	11	13	**3**	9	**6**	7	14	**2**	8	**5**	9	15
hayes-ro	10	**7**	**1**	**1**	14	9	10	**1**	**1**	13	**7**	10	**1**	**1**	14
hepatitis	**5**	**2**	14	15	7	**1**	**4**	10	10	8	**6**	**3**	13	10	9
horse-c	**7**	**3**	13	10	**1**	**6**	**5**	13	11	**1**	**8**	**9**	15	12	4
ionosph	8	**1**	5	**3**	13	10	**2**	6	**4**	14	11	9	7	12	15
iris	14	7	10	9	**2**	15	7	10	6	**1**	13	3	12	4	5
led24	**2**	**6**	9	8	15	**1**	**3**	7	10	13	**5**	**4**	12	11	14
liver	**7**	**10**	14	15	**6**	**2**	**4**	13	12	**5**	**3**	**1**	11	**9**	**8**
lymph	15	5	10	8	**1**	14	5	12	4	**3**	13	8	11	5	**2**
nursery	8	11	**1**	**1**	15	9	9	**4**	**5**	14	12	7	**3**	**6**	13
pima	**8**	**1**	15	14	**3**	**7**	**2**	10	11	**4**	**6**	**9**	12	13	**5**
segment	**4**	10	**2**	9	14	**6**	12	**3**	11	13	**4**	**8**	**1**	**6**	15
solar	**4**	**8**	10	12	**1**	**7**	**5**	12	12	**3**	**9**	**5**	11	12	**2**
sonar	10	**7**	**1**	**5**	13	9	12	**2**	**6**	14	10	**3**	**3**	8	15
soybean	**12**	**7**	**6**	**5**	14	**11**	**7**	**1**	**1**	13	**9**	**10**	**3**	**3**	15
threeOf9	**9**	**7**	**1**	**3**	14	**10**	**12**	**3**	**1**	13	**7**	**11**	**3**	**3**	15
tic-tac-toe	**9**	**8**	**1**	**3**	15	**12**	**7**	**4**	**1**	13	**11**	**9**	**6**	**4**	14
votes	**8**	**5**	**1**	**12**	13	**1**	**8**	**10**	**10**	14	**5**	**5**	**1**	**1**	14
waveform	11	**1**	**7**	**2**	13	12	**3**	**8**	**5**	15	10	**4**	**9**	**6**	14
wine	**5**	**10**	**1**	**8**	14	**5**	**11**	**3**	**8**	15	**4**	**12**	**1**	**5**	13

Table 1: Ranked performance comparison between random forest with and without random rotation (rf, rrrf), extra trees with and without random rotation (et, rret), and rotation trees (rot). For all data sets (left-hand side), the comparison is performed over the three scaling methods described in the text as basic scaling (b-scale), quantile scaling (q-scale), and ranking (ranked). The values represent ranks of the classification errors for classification problems and the ranks of the RMSE for regression problems. Values that are within one cross-sectional standard deviation of the minimum error are in bold.

In this table, we have omitted flip rotations because their performance was comparable to the simple random rotation, with flip rotations outperforming in 36% of cases, simple rotations outperforming in 45% of cases and ties in the remaining 19% of cases.

The best overall average rank of 6.10 (of 15) was achieved by the random rotation random forest algorithm with simple scaling, followed by the same algorithm with complex scaling (6.48) and ranking (6.55). This algorithm outperformed regardless of scaling. The next lowest rank of 6.72 was achieved by random rotation extra trees using quantile scaling.

It is interesting to note that the average ranks for each scaling type were 7.50 for the complex scaling, 7.65 for the simple scaling and 7.81 for ranking. This indicates that the scaling method was less influential than the selection of the algorithm. In particular, the new method often improved on the original method even when only the ranks of the data were considered. We believe this to be an interesting result because it indicates that even rotating ranks can improve performance. Obviously, ranked data is completely robust to scaling effects and outliers.

The best average rank across scalings was achieved by random rotation random forests with 6.38, followed by extra trees (7.06), random forests (7.20), random rotation extra trees (7.26), and rotation forests (10.38).

In our tests, rotation forests underperformed overall but showed strong performance in some particular cases. The problem here was that when rotation forests did not excel at a problem, they often were the worst performer by a large margin, which had an impact on the average rank. In contrast, random rotation random forests rarely displayed the very best performance but often were among the top predictors. This insight led us to consider predictors that were within one cross-sectional standard deviation of the best predictor for each data set.

Random rotation random forests were within one standard deviation of the best result (highlighted in bold in Table 1) in 67.8% of cases, random forests without rotation in 64.3% of cases, extra trees (with and without rotation) in 49.4% of cases, and rotation forests in 27.6%. It appears to be clear that random rotation can improve performance for a variety of problems and should be included as a user option for standard machine learning packages.

Random rotation appears to work best when numerical predictors outnumber categorical predictors, which are not rotated, and when these numerical predictors exhibit a relatively smooth distribution (rather than a few pronounced clusters). An example of a suitable dataset is Cleveland, with more than half of the variables continuous and spread out evenly. In contrast, Balance is an example of a dataset for which we cannot expect random rotation to perform well. However, in general it is difficult to judge the utility of rotation in advance and we recommend running a small test version of the problem with and without rotation to decide which to use: when the approach is successful, this tends to be apparent early.

Constructing a random rotation matrix using the indirect method described above requires of the order of p^3 operations, where p is the number of predictors to be rotated (time complexity of QR factorization). Multiplying the resulting random rotation matrix with an input vector requires of the order of p^2 operations. During training, this step needs to be performed k times, where k is the number of instances in the training data, for a total of $k \times p^2$ operations. All but one of the UCI datasets contained fewer than 100 predictors, and it takes less than a millisecond to compute a 100x100 random rotation matrix. Hence, with 5000 trees in each ensemble, the additional computational overhead was at most a few

seconds. However, as indicated above, for larger problems it does make sense to restrict the number of rotated predictors to maintain adequate performance.

Next, we consider the question of robustness.

4.5 Parameter Sensitivity and Extensions

In addition to the full tests described above, we also ran a few smaller examples to examine the sensitivity of random rotation to the choice of parameters in the underlying base learners. For this, we used the well known UCI *iris* data set (4 numeric predictors, 3 classes, 150 rows). Table 2 compares the performance of the standard random forest algorithm (RF) and a modified version including random feature rotation (RR-RF) on this data set.

Random Forest Comparison (iris)

parameters		% error		% wins per method		
ntree	mtry	RF	RR-RF	RF	RR-RF	Ties
50	1	5.269	**4.464**	16.98	**53.40**	29.62
	2	5.011	**4.237**	15.80	**50.20**	34.00
	3	4.960	**4.155**	16.14	**51.10**	32.76
	4	4.963	**4.077**	15.30	**52.75**	31.95
500	1	5.246	**4.414**	11.76	**52.98**	35.26
	2	4.981	**4.226**	13.53	**48.41**	38.06
	3	4.904	**4.144**	14.90	**49.22**	35.88
	4	4.944	**4.096**	13.80	**51.53**	34.67
5000	1	5.227	**4.385**	10.29	**52.52**	37.19
	2	4.975	**4.196**	13.48	**49.57**	36.95
	3	4.860	**4.133**	15.23	**47.76**	37.01
	4	4.964	**4.132**	14.22	**51.14**	34.64

Table 2: Performance comparison of the standard random forest algorithm (RF) and a modified version with randomly rotated feature space for each tree (RR-RF) on the *iris* data set. Ntree is the total number of trees in the forest, mtry the number of randomly selected features considered at each decision node. Statistically significant differences in mean error percentage and win percentage at the 1% level are denoted in bold.

As in the detailed tests, both classifiers made use of the same tree induction algorithm, implemented in the *randomForest* R package, but the feature space was randomly rotated prior to the construction of each tree for the RR-RF algorithm. Since the iris data set only includes 4 predictors, the number of randomly selected features at each decision node (mtry) only has feasible values in 1-4, allowing for an exhaustive comparison. For each parameter setting, we selected 50% of the data (75 cases) at random as the learning data set, while the other half served as the test data set. The experiment was repeated 10000 times for each parameter setting and we kept track of the average error percentage of each method, as well as the percentage of times each method outperformed the other. Once

completed, a Wilcoxon signed-rank test was performed to compare the classification error percentage and win percentage of the original method with that of the modified classifier and to ascertain statistical significance at the 1% level. The modified ensemble featuring random rotation appears to universally outperform the original classifiers on this data set, regardless of parameter settings and in a statistically significant manner. However, it should be noted that the goal of this experiment was not to demonstrate the general usefulness of random rotations – this is achieved by the detailed experiments in the previous section – but rather to show the robustness to parameter changes for a specific data set.

Table 3 shows the analogous results for extra trees, which select both the split feature and the split point at random. Here we used the tree induction algorithm implemented in the *extraTrees* R package. In theory, there exist an infinite number of feasible split points (ncut) that could be chosen but for simplicity, we have only attempted ncut values in the range 1-4, meaning that at most 4 random split points were considered in the tests. The improvement due to rotation is again universal and statistically significant. For reasonable parameters (e.g. ntree ≥ 50), the new method matches or outperforms the original method in over 94% of the randomly generated cases and the performance improvement is 21.7% on average. This is again a very encouraging result, as it demonstrates that the results above are robust, even if non-default parameters are used for the base learners. It is also interesting to note that randomly rotated extra tree ensembles outperform randomly rotated random forests here and they tend to do best with lower ncut values, indicating that more randomness (via rotation, feature selection, and split selection) is helpful for this particular problem.

Table 4 shows comparable results with a gradient boosting machine from the *gbm* R package (Ridgeway, 2013). Since boosting is an additive procedure, where later trees have an explicit dependence on earlier trees in the ensemble, the comparison of the two methods is not as straightforward. More specifically, step (B).(2) in Listing 1 cannot be performed without knowing (and being able to reuse) f_{m-1} in the case of boosting. Unfortunately, the most common software packages for boosting (and *gbm* in particular) do not provide an interface for this. Of course, we could have implemented our own boosting library but then it would not be obvious that the improvement in predictive performance was entirely due to the rotation. For this reason, we opted to demonstrate boosting with a widely used package but on groups of trees, with one rotation per group of 50 trees. The rationale for this choice of group size was that the first few boosting iterations often lead to rapid improvements. In this case, we compared the original method, consisting of 5000 trees in a single ensemble, to a modified version with 100 sub-forests of 50 trees each, whereby each sub-forest was created on a randomly rotated feature space. In other words, the 50 trees in each sub-forest had a dependency, whereas the sub-forests themselves were independent of each other. The final classification was achieved through voting. As is evident from Table 4, randomly rotated gradient boosting machines even outperformed random forests and extra trees on this data set in terms of percentage error. Even when we handicapped the new method by only providing it with relative ranks of the data it outperformed the original (unrotated) method, although not by the same margin.

Extra Tree Ensemble Comparison (iris)

parameters		% error		% wins per method		
ntree	ncut	ET	RR-ET	ET	RR-ET	Ties
50	1	5.335	**3.994**	7.18	**66.11**	26.71
	2	5.281	**4.101**	7.84	**63.04**	29.12
	3	5.183	**4.078**	8.41	**60.69**	30.90
	4	5.238	**4.152**	8.86	**60.14**	31.00
500	1	5.244	**3.971**	4.09	**66.02**	29.89
	2	5.157	**4.045**	4.75	**60.85**	34.40
	3	5.118	**4.056**	5.16	**59.67**	35.17
	4	5.114	**4.111**	5.54	**57.68**	36.78
5000	1	5.257	**4.044**	4.73	**66.09**	29.18
	2	5.175	**4.003**	3.32	**60.86**	35.82
	3	5.038	**4.079**	4.71	**56.20**	39.09
	4	5.046	**4.053**	5.55	**56.92**	37.53

Table 3: Performance comparison of the standard extra trees algorithm (ET) and a modified version with randomly rotated feature space for each tree (RR-ET) on the *iris* data set. Ntree is the total number of trees in the ensemble, ncut the number of randomly selected split points considered at each decision node. Statistically significant differences in mean error percentage and win percentage at the 1% level are denoted in bold.

Gradient Boosting Comparison (iris)

parameters				performance	
rank only	type	ntree	shrinkage	%error	%wins
no	GBM	1x5000	0.0005	5.063	9.62
no	RR-GBM	100x50	0.0500	**3.831**	**57.26**
yes	GBM	1x5000	0.0005	5.063	22.97
yes	RR-GBM	100x50	0.0500	**4.385**	**46.17**

Table 4: Performance comparison of the standard gradient boosting machine (GBM) and a modified version with randomly rotated feature space for each sub-forest of 50 trees (RR-GBM) on the *iris* data set. A classifier was trained for each parameter setting on a random half of the data and tested on the remaining half. Ntree is the total number of trees in the ensemble, expressed as the product of the number of generated sub-forests (each on a randomly rotated feature space) times the number of trees in each sub-forest. The procedure was repeated 10000 times. Statistically significant differences in mean error percentage and win percentage at the 1% level are denoted in bold. For the robust rank only version, a ranking of each predictive variable was performed using the train data, while the test vectors received an interpolated ranking based solely on relative order information with respect to the train data.

4.6 A Note on Diversity

In Appendix B we closely follow Breiman (2001) to derive an ensemble diversity measure that is applicable to the case of random rotation ensembles. In particular, we show that just like for random forests we can express the average correlation of the raw margin functions across all classifiers in the ensemble in terms of quantities we can easily estimate, specifically

$$\bar{\rho}(\cdot) = \frac{V_{x,y}[\Psi(x,y)]}{E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta))]^2}. \tag{8}$$

That is, average correlation $\bar{\rho}$ is the variance of the margin across instances $V_{x,y}[\Psi(x,y)]$, divided by the expectation of the standard deviation σ of the raw margin across (randomized) classifiers squared. Full definitions of these quantities can be found in Appendix B.

As an example of the usefulness of this correlation measure, we estimated $\bar{\rho}$ with $m_{try} = \{1, 4\}$ on the iris example and achieved a correlation of 0.32 and 0.61, respectively. Clearly, the random split selection decorrelates the base learners. We then performed the same calculation including random rotation and achieved 0.22 and 0.39, respectively. In both cases, the correlation decreased by approximately one third. In contrast, the expected margin only decreased by 3.5%, meaning that the accuracy of the individual base learners was only very modestly affected.

5. Conclusion

Random rotations provide a natural way to enhance the diversity of an ensemble with minimal or no impact on the performance of the individual base learners. Rotations are particularly effective for base learners that exhibit axis parallel decision boundaries, as is the case for all of the most common tree-based learning algorithms. The application of random rotation is most effective for continuous variables and is equally applicable to higher dimensional problems.

A generalization of random rotations only uses a subset of rotations for out of sample predictions. This subset is chosen by observing the out-of-bag performance of each rotation in sample. Initial tests revealed that dropping the least effective decile of all random rotations generally improved out of sample performance but more research is needed because the procedure potentially introduces model bias.

Random rotations may also prove to be useful for image analysis. For example, axis-aligned methods for image processing, such as wavelet smoothing, may benefit from repeated random rotations to ensure that the methods become axis-independent.

While random rotations are certainly not a panacea, they are helpful frequently enough that we contend standard data mining packages should provide users the option to randomly rotate the feature space prior to inducing each base learner.

Appendix A

The following listings provide illustrations in two commonly used programming languages for the generation of a random rotation matrix using the indirect method described in section 3 above. The code is kept simple for illustrative purposes and does not contain error checking or performance optimizations.

Listing 2: Random Rotation in C++ using Eigen

```
#include "MersenneTwister.h"
#include <Eigen/Dense>
#include <Eigen/QR>

using namespace Eigen;

// C++: generate random n x n rotation matrix
void random_rotation_matrix(MatrixXd& M, int n)
{
  MTRand mtrand; // twister with random seed

  MatrixXd A(n,n);
  const VectorXd ones(VectorXd::Ones(n));

  for(int i=0; i<n; ++i)
    for(int j=0; j<n; ++j)
      A(i,j) = mtrand.randNorm(0,1);

  const HouseholderQR<MatrixXd> qr(A);
  const MatrixXd Q = qr.householderQ();
  M = Q * (qr.matrixQR().diagonal().array()
           < 0).select(-ones,ones).asDiagonal();

  if(M.determinant() < 0)
    for(int i=0; i<n; ++i)
      M(i,0) = -M(i,0);
}
```

Listing 3: Random Rotation in R

```
# generate random member of orthogonal group O(n)
random_rotation_matrix_incl_flip <- function(n)
{
  QR <- qr(matrix(rnorm(n^2), ncol=n))          # A = QR
  M <- qr.Q(QR) %*% diag(sign(diag(qr.R(QR)))) # diag(R) > 0
  return(M)
}

# generate random member of special orthogonal group SO(n)
random_rotation_matrix <- function(n)
{
  M <- random_rotation_matrix_incl_flip(n)
  if(det(M)<0) M[,1] <- -M[,1]                  # det(M) = +1
  return(M)
}
```

Appendix B

In this appendix, we closely follow Breiman (2001) to derive an ensemble diversity measure that is applicable to random rotation ensembles.

For a given input vector x, we define the label of the class to which classifier $f_M(x)$ assigns the highest probability, save for the correct label y, to be j^M_{max}, that is

$$j^M_{max} := \arg\max_{j \neq y} P(f_M(x) = j). \tag{9}$$

Using this definition, we denote the raw margin function $\psi(x, y)$ for a classification tree ensemble $f_M(x)$ as

$$\psi(x, y, \theta) = I(f_M(x) = y) - I(f_M(x) = j^M_{max}), \tag{10}$$

with indicator function $I(\cdot)$. This expression evaluates to +1 if the classification is correct, -1 if the most probable incorrect class is selected, and 0 otherwise. The margin function $\Psi(x, y)$ is its expectation, that is

$$\begin{aligned} \Psi(x, y) &= E_\theta[\psi(x, y, \theta)] \\ &= P(f_M(x) = y) - P(f_M(x) = j^M_{max}). \end{aligned} \tag{11}$$

The margin function $\Psi(x, y)$ represents the probability of classifying an input x correctly minus the probability of selecting the most probable incorrect class.

If we denote the out-of-bag instances for classification tree $T(x; \theta_m, \Omega_m)$ as O_m, then these probabilities can be estimated as

$$\hat{P}(f_M(x) = k) = \frac{\sum_{m=1}^M I(T(x; \theta_m, \Omega_m) = k \wedge (x, y) \in O_m)}{\sum_{m=1}^M I((x, y) \in O_m)}, \tag{12}$$

where the denominator counts the number of base learners for which (x, y) is out-of-bag. If we were to use a separate testing data set S, as we do in our examples, this can be further simplified to

$$\hat{P}(f_M(x) = k) = \frac{1}{M} \sum_{m=1}^M I(T(x; \theta_m, \Omega_m) = k), \tag{13}$$

where any instance (x, y) must be selected from S. From this, the expected margin can be estimated as

$$\hat{E}_{x,y}[\Psi(x, y)] = \hat{E}_{x,y}[\hat{P}(f_M(x) = y) - \hat{P}(f_M(x) = j^M_{max})] \tag{14}$$

and its variance as

$$\hat{V}_{x,y}[\Psi(x, y)] = \hat{E}_{x,y}[(\hat{P}(f_M(x) = y) - \hat{P}(f_M(x) = j^M_{max}))^2] - \hat{E}_{x,y}[\Psi(x, y)]^2. \tag{15}$$

The expectations are computed over the training set for the out-of-bag estimator or the testing data set respectively, depending on which approach is used.

Using Chebyshev's inequality, we can derive a bound for the probability of achieving a negative margin, a measure of the generalization error:

$$\begin{aligned} P(\Psi(x,y)<0) &\leq P(|E_{x,y}[\Psi(x,y)]-\Psi(x,y)|\geq E_{x,y}[\Psi(x,y)]) \\ &\leq \frac{V_{x,y}[\Psi(x,y)]}{E_{x,y}[\Psi(x,y)]^2}, \end{aligned} \tag{16}$$

which can be estimated from equations (14) and (15). Clearly, this inequality is only useful if the expected margin is positive because otherwise the classification is no better than random.

We now follow Breiman's argument for obtaining a measure of ensemble diversity in terms of the random classifier parameters θ, which in our case include the random rotation in addition to the bootstrap samples. First, we note that for independent and identically distributed (i.i.d.) random parameters θ_1 and θ_2, we have

$$\begin{aligned} E_{\theta_1,\theta_2}[\psi(x,y,\theta_1)\times\psi(x,y,\theta_2)] &= E_{\theta_1}[\psi(x,y,\theta_1)]\times E_{\theta_2}[\psi(x,y,\theta_2)] \\ &= \Psi(x,y)\times\Psi(x,y) \\ &= \Psi(x,y)^2. \end{aligned} \tag{17}$$

Therefore, the variance of $\Psi(x,y)$ can be reformulated as

$$\begin{aligned} V_{x,y}[\Psi(x,y)] &= E_{x,y}[\Psi(x,y)^2]-E_{x,y}[\Psi(x,y)]^2 \\ &= E_{x,y}[E_{\theta_1,\theta_2}[\psi(x,y,\theta_1)\times\psi(x,y,\theta_2)]]-E_{x,y}[E_{\theta_1}[\psi(x,y,\theta_1)]]\times E_{x,y}[E_{\theta_2}[\psi(x,y,\theta_2)]] \\ &= E_{\theta_1,\theta_2}[E_{x,y}[\psi(x,y,\theta_1)\times\psi(x,y,\theta_2)]-E_{x,y}[\psi(x,y,\theta_1)]\times E_{x,y}[\psi(x,y,\theta_2)]] \\ &= E_{\theta_1,\theta_2}[Cov_{x,y}[\psi(x,y,\theta_1),\psi(x,y,\theta_2)]] \\ &= E_{\theta_1,\theta_2}[\rho(\psi(x,y,\theta_1),\psi(x,y,\theta_2))]\times E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta_1))\times\sigma(\psi(x,y,\theta_2))] \\ &= E_{\theta_1,\theta_2}[\rho(\psi(x,y,\theta_1),\psi(x,y,\theta_2))]\times E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta))]^2. \end{aligned} \tag{18}$$

This result allows us to express the average correlation of the raw margin functions across all classifiers in the ensemble in terms of quantities we can easily estimate, specifically

$$\bar{\rho}(\cdot)=\frac{V_{x,y}[\Psi(x,y)]}{E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta))]^2}, \tag{19}$$

where we can use (15) as an estimate of the numerator. In other words, correlation represents the variance of the margin across instances, divided by the expectation of the standard deviation of the raw margin across (randomized) classifiers squared.

To estimate the denominator across all random parameters θ – i.e. the individual base learners and rotations – we can use

$$\begin{aligned} E_{\theta_1,\theta_2}[\sigma(x,y,\theta)]=\frac{1}{M}\sum_{m=1}^{M}&\sqrt{P(f_m(x)=y)+P(f_m(x)=j^m_{max})-} \\ &\overline{(P(f_m(x)=y)-P(f_m(x)=j^m_{max}))^2}, \end{aligned} \tag{20}$$

where the probabilities are calculated for each individual classifier across all instances (x, y) in the out-of-bag or test set respectively, i.e. in the case of a test set S we would use

$$\hat{P}(f_m(x) = k) = \frac{1}{|S|} \sum_{(x_i,y)\in S} I(T(x_i; \theta_m, \Omega_m) = k), \tag{21}$$

with $|S|$ denoting the cardinality of S.

Appendix C

name	cases	preds
anneal	798	51
audiology	200	85
balance	625	16
breast-w	699	80
breast-y	286	34
chess	28056	34
cleveland	303	22
credit-a	690	14
flare	1066	21
glass	214	9
hayes-roth	132	4
hepatitis	155	29
horse-colic	300	80
ionosphere	351	33
iris	150	4
led24	3200	24
liver	345	44
lymph	148	18
nursery	12960	19
pima	768	167
segmentation	210	19
solar	323	22
sonar	208	60
soybean	307	97
threeOf9	512	9
tic-tac-toe	958	18
votes	435	32
waveform	5000	21
wine	178	13

Table 5: Description of UCI datasets used to perform the detailed tests of random rotations. The the total number of available instances, as well as the number of available predictor variables after preprocessing (including dummy variables for categories) is shown for each data set. The tests use a random 70% of the available instances as training set and the remaining 30% as a test set.

	B-SCALE					Q-SCALE					RANKED				
NAME	**rf**	**rrrf**	**et**	**rret**	**rot**	**rf**	**rrrf**	**et**	**rret**	**rot**	**rf**	**rrrf**	**et**	**rret**	**rot**
anneal	173	165	100	103	188	170	173	100	100	168	167	173	102	100	207
audiology	2060	2040	1687	1720	2927	2053	2027	1700	1687	2947	2400	2060	1693	1693	2960
balance	2694	2715	3728	3728	1957	2700	2679	3721	3715	1966	2715	2683	3715	3719	1966
breast-w	383	373	389	383	415	379	383	389	387	417	385	381	389	389	408
breast-y	2586	2623	2879	2888	2902	2586	2600	2888	2865	2893	2614	2586	2851	2893	2898
chess	4907	4905	3539	3536	4911	4906	4908	3537	3537	4912	4911	4906	3535	3540	4911
cleveland	4255	4233	4316	4251	4356	4273	4246	4312	4229	4378	4316	4268	4316	4233	4352
credit-a	1323	1431	1479	1546	1471	1313	1375	1448	1515	1394	1313	1383	1385	1446	1331
flare	336	336	418	419	336	336	336	420	420	336	336	336	419	423	336
glass	2191	2966	2246	2929	3218	2215	2763	2314	2671	3286	2203	2714	2283	2763	3415
hayes-ro	2100	2080	1840	1840	2500	2090	2100	1840	1840	2490	2080	2100	1840	1840	2500
hepatitis	1489	1455	1872	1881	1660	1447	1481	1838	1838	1694	1506	1464	1847	1838	1762
horse-c	1520	1489	1711	1680	1484	1516	1511	1711	1684	1484	1524	1547	1751	1698	1507
ionosph	543	343	475	415	611	547	389	483	426	645	555	547	532	604	736
iris	524	444	462	453	338	569	444	462	436	320	516	400	507	409	427
led24	2763	2769	2848	2847	2970	2759	2765	2845	2849	2967	2768	2765	2858	2852	2969
liver	2931	3046	3408	3496	2927	2900	2915	3315	3285	2923	2912	2846	3177	3008	2965
lymph	7270	6889	6952	6921	5831	7143	6889	7016	6762	5867	7111	6921	6984	6889	5849
nursery	487	489	79	79	896	487	487	80	80	889	490	485	79	80	888
pima	2506	2397	2810	2807	2428	2497	2424	2658	2706	2431	2488	2539	2731	2800	2469
segment	743	978	705	914	1168	762	1098	730	1035	1130	743	794	603	762	1340
solar	2775	2796	2994	3006	2763	2784	2779	3006	3006	2771	2804	2779	2998	3006	2767
sonar	1721	1530	1283	1403	1981	1714	1740	1365	1511	2076	1721	1397	1397	1606	2190
soybean	951	933	920	916	1419	946	933	903	903	1381	938	942	908	908	1432
threeOf9	109	104	75	78	366	112	122	78	75	364	104	114	78	78	369
tic-tac-toe	104	101	90	92	688	108	100	93	90	678	106	104	94	93	681
votes	369	366	363	379	437	363	369	373	373	440	366	366	363	363	440
waveform	1462	1330	1381	1335	1563	1465	1347	1386	1362	1594	1459	1353	1411	1378	1588
wine	200	230	126	207	556	200	244	141	207	570	185	259	126	200	430

Table 6: Raw performance comparison between random forest with and without random rotation (rf, rrrf), extra trees with and without random rotation (et, rret), and rotation trees (rot). For all data sets (left-hand side), the comparison is performed over the three scaling methods described in the text as basic scaling (b-scale), quantile scaling (q-scale), and ranking (ranked). The values represent classification errors for classification problems and RMSE for regression problems ($\times$10000).

References

Dean Abbott. Why ensembles win data mining competitions. In *Predictive Analytics Centre of Excellence Tech Talks*, University of California, San Diego, 2012.

Theodore W. Anderson, Ingram Olkin, and Les G. Underhill. Generation of random orthogonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 8:625–629, 1987.

Kevin Bache and Moshe Lichman. *UCI machine learning repository*. University of California, Irvine, School of Information and Computer Science, 2013. URL `http://archive.ics.uci.edu/ml`.

Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

Leo Breiman. Random forests – random features. Technical report, University of California at Berkeley, Berkeley, California, 1999.

Leo Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40: 229–242, 2000.

Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, 2006.

Adele Cutler and Guohua Zhao. PERT-perfect random tree ensembles. *Computing Science and Statistics*, 33:490–497, 2001.

Koen W. De Bock and Dirk Van den Poel. An empirical evaluation of rotation-based ensemble classifiers for customer churn prediction. *Expert Systems with Applications*, 38: 12293–12301, 2011.

Persi Diaconis and Mehrdad Shahshahani. The subgroup algorithm for generating uniform random variables. *Probability in the Engineering and Informational Sciences*, 1:15–32, 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*, Studies in Computational Intelligence, pages 169–179. Springer Berlin Heidelberg, 2011.

Wei Fan, Joe McCloskey, and Philip S. Yu. A general framework for accurate and fast regression by data summarization in random decision trees. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 136–146, New York, NY, USA, 2006. ACM.

Khaled Fawagreh, Mohamed Medhat Gaber, and Eyad Elyan. Random forests: from early developments to recent advancements. *Systems Science & Control Engineering*, 2(1): 602–609, September 2014.

Yoav Freund and Robert Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy, 1996. Morgan Kaufmann Publishers Inc.

Mark Galassi. *GNU scientific library reference manual - third edition*. Network Theory Ltd., January 2009.

Nicols Garca-Pedrajas, Csar Garca-Osorio, and Colin Fyfe. Nonlinear boosting projections for ensemble construction. *Journal of Machine Learning Research*, 8:1–33, 2007.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63:3–42, 2006.

Gal Guennebaud, Benot Jacob, et al. *Eigen: linear algebra template library*. 2010. URL `http://eigen.tuxfamily.org`.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, New York, 2009.

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

Tin K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:832–844, 1998.

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

Andrew Kerr, Dan Campbell, and Mark Richards. QR decomposition on GPUs. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, pages 71–78, New York, NY, USA, 2009. ACM.

Donald E. Knuth. *Art of computer programming, volume 2: seminumerical algorithms*. Addison-Wesley Professional, Reading, Mass, 3 edition edition, November 1997.

Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Springer, New York, 2013 edition edition, September 2013. ISBN 9781461468486.

Ludmila I. Kuncheva and Juan J. Rodriguez. An experimental study on rotation forest ensembles. In *Proceedings of the 7th International Conference on Multiple Classifier Systems*, MCS'07, pages 459–468, Berlin, Heidelberg, 2007. Springer-Verlag.

Dan Ledermann and Carol Alexander. ROM simulation with random rotation matrices. SSRN Scholarly Paper ID 1805662, Social Science Research Network, Rochester, NY, 2011.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

Fei Tony Liu, Kai Ming Ting, and Wei Fan. Maximizing tree diversity by building complete-random decision trees. In *Proceedings of the 9th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'05, pages 605–610, Berlin, Heidelberg, 2005. Springer-Verlag.

Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

Bjoern H. Menze, B. Michael Kelm, Daniel N. Splitthoff, Ullrich Koethe, and Fred A. Hamprecht. On oblique random forests. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, ECML PKDD'11, pages 453–469, Berlin, Heidelberg, 2011. Springer-Verlag.

Francesco Mezzadri. How to generate random matrices from the classical compact groups. *Notices of the AMS*, 54:592–604, 2007. NOTICES of the AMS, Vol. 54 (2007), 592-604.

Greg Ridgeway. gbm: generalized boosted regression models, 2013. URL `http://CRAN.R-project.org/package=gbm`. R package version 2.1.

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

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

Jaak Simm and Ildefons Magrans de Abril. Extratrees: extratrees method, 2013. URL `http://CRAN.R-project.org/package=extraTrees`. R package version 0.4-5.