London School of Economics

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Random rotations in machine learning

Rico Blaser

September 1, 2021

Thesis supervisor

Prof. Piotr Fryzlewicz

To Lea & Jana

Rico Blaser
Random rotations in machine learning
PhD Thesis, September 1, 2021
Supervisor: Prof. Piotr Fryzlewicz

London School of Economics
Department of Statistics
Houghton Street
London WC2A 2AE
United Kingdom

Declaration

The opposite of a correct statement is a false statement. But the opposite of a profound truth may well be another profound truth.

— **Niels Bohr**
(Physicist)

General

Co-authored work

I confirm that parts of Chapter 1 were adapted into a paper entitled *Random Rotation Ensembles*, co-authored with Prof. Piotr Fryzlewicz, and published in the Journal of Machine Learning Research (JMLR).

I further confirm that parts of Chapter 2 were adapted into a paper entitled *Regularizing axis-aligned ensembles via data rotations that favor simpler learners*, co-authored with Prof. Piotr Fryzlewicz, and published in Statistics and Computing (STCO). The corresponding R package was published on GitHub and is entitled *random.rotation*.

Finally, I confirm that a shortened version of Section 4.5 was co-authored with Prof. Piotr Fryzlewicz and published in the Journal of the Royal Statistical Society, Statistical Methodology, Series B in the discussion section of 3rd party paper on Random Projections.

It is expected that the insights from Chapter 3 will eventually be turned into a publication too, again with joint authorship.

London, September 1, 2021

[illegible]

Rico Blaser

Abstract

This thesis discusses applications of random rotations in machine learning. Rotations of the feature space can lead to more diverse ensembles, better predictions, less complex classifiers and smoother decision boundaries.

In Chapter 1 of this thesis, the feature space is randomly rotated and one or more independent base learner is constructed on each rotation. In the case of classification, each base learner receives one vote in the final ensemble prediction; for regressions, predictions are averaged. An empirical study demonstrates the efficacy of random rotations.

Observing that not all rotations are equally effective, Chapter 2 is dedicated to the analysis of what makes a rotation effective and whether it is possible to emphasize such rotations in the final ensemble prediction. It is demonstrated that focusing on rotations that lead to simpler base learners leads to more compact ensembles and often increases predictive accuracy. In this chapter, predictions are aggregated in a parametric fashion, providing more weight to less complex predictors in the final ensemble. Multiple parametric forms are explored.

Instead of constructing one or more predictor for each rotation, it is also possible to provide multiple rotations of the feature space to a single predictor. This effectively provides a single predictor with multiple simultaneous viewpoints on the same feature space. The first half of Chapter 3 explores this idea. A great benefit of this approach, when compared to the methods described in the earlier chapters, is that the aggregation of the predictions across multiple rotations becomes part of the training algorithm of the classifier, rather than being constructed exogenously. This also makes the approach viable for ensemble architectures with an interdependence between the base learners, such as boosting. Finally, an importance measure can be used not

only to select the most salient features but also to determine the most helpful rotations.

A different method of combining multiple rotations is to form a meta- or stacking predictor that leverages the base predictions on each rotation as inputs. This results in a generalization of the results of Chapter 2, whereby the aggregation becomes nonparametric in nature and local with respect to the decision boundary. In this context, extra care must be taken to avoid data snooping biases. A repeated, nested cross-validation technique is described in the second half of Chapter 3 to facilitate this process. The procedure directly answers the question if rotations are helpful for a specific data set and provides an avenue for selecting effective rotations.

Chapter 4 is concerned with the impact random rotations have had on the scientific literature and open source software community since their introduction with the publication of our initial paper on the topic.

Acknowledgement

I would like to express my heartfelt gratitude to my supervisor Prof. Piotr Fryzlewicz. It has been a long journey and I'm very thankful for his incredible patience, continuous encouragement and strong guidance. It has been an honor to work with such an immensely talented researcher and I'm grateful for the invaluable insights. His positive attitude, undying curiosity and humble demeanor continue to be an inspiration to me.

Furthermore, I wish to express my gratitude to the staff and students of the London School of Economics and the Department of Statistics in particular for providing me with such a productive and supportive environment. The first-class reputation of this institution is well deserved.

Last but not least, I would like to thank my family for their strong support and ongoing encouragement.

Contents

1 Random Rotation Ensembles

Where you stand determines what you see and what you do not see; it determines also the angle you see it from; a change in where you stand changes everything.

— **Steve de Shazer**
(Author, psychotherapist)

1.1 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).

The main findings of this chapter are based on Blaser and Fryzlewicz (2016), originally published in the Journal of Machine Learning Research.

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; Ho, 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; Breiman, 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 (F. T. 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 (García-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 this chapter 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 Van den Poel (2011), independent component analysis (ICA) is found to yield the best performance.

The premise of the present chapter 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 chapter is structured as follows. Section 1.3 provides a motivational example for the use of random rotations using a well-known data set. In Section 1.4 we formally introduce random rotations and provide guidance as to their construction. Section 1.6 evaluates different application contexts for the technique and performs experiments to assesses its effectiveness. Conclusions are discussed in Section 1.7.

Random rotations provide an intuitive, optional enhancement to a number of existing machine learning techniques. For this reason, we also provide random rotation code in C/C++ and R in this chapter, which can be used as a basis for enhancing existing software packages. A full implementation of the methods described in this thesis in the form of an open source R package called *random.rotation* is introduced in Chapter 5. The package contains a reference implementation of random rotations, including the weighting- and regularisation methods described in later chapters. The package can be downloaded from GitHub without registration. The easiest way to accomplish this is directly within an R command-line shell:

```
library("devtools")
install_github("randomrotation/random.rotation")
```

1.2 Recent literature

The attentive reader will have noticed that most of the literature references in this chapter stop at around 2014. This is because the material in this chapter was originally published as Blaser and Fryzlewicz (2016). All of Chapter 4 is dedicated to the impact this paper has had in the field, including a detailed analysis of the publications referencing this work. The aim of the present section is merely to bring to the attention of the reader a few recent developments that have taken place since the initial publication and independently of our paper. For example, there are a number of more recent studies that have reiterated the efficacy of ensemble learning for specific areas of application, including Delgado et al. (2014), Nawar and Mouazen (2017) and Treboux et al. (2018).

Data science competitions also continue to be won by tree-based ensembles, particularly since the introduction of *XGBoost* in Chen and Guestrin (2016). At the end of 2016 the CEO of the Kaggle competition revealed in a blog post (Goldbloom, 2016) that since 2014, ensembles of boosted trees as well as neural networks were far more likely to win the competition than any other supervised learning algorithms.

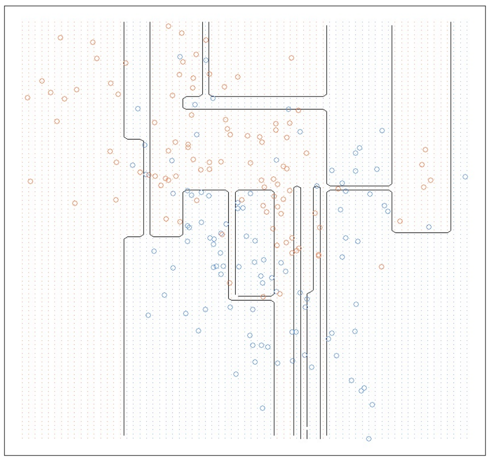
One of the most relevant trends in machine learning over the course of the past decade has been the success of deep learning, particular in the areas of natural language processing as well as image- and video analysis. In the case of structured tabular data, tree-based ensembles still outperform state-of-the-art specialized neural networks. However, the latest research is even challenging this notion. For example, Badirli et al. (2020) discusses a combination of regularization techniques for neural networks leading to an architecture that outperforms *XGBoost* on 40 structured data sets.

It should also be noted that there is no inherent requirement for using tree-based models for ensemble learning. Just like decision trees, neural networks are known to exhibit a high model variance: small changes to the weight initialization or network architecture (e.g. the number of hidden layers, the type of activation function, or the number of units deployed) can lead to vastly different networks. Combined with accurate, low-bias predictions, neural networks are therefore well suited for ensemble learning. This has been known for decades but computational resources proved to be a bottleneck. In recent years, this area of research as proven to be fruitful. For example, in Kadra et al. (2021) the authors use gradient boosting of shallow neural networks to achieve performance comparable to deep neural networks for specific tasks but with much shorter training times and a more compact model. Goodfellow et al. (2016) also make an argument for using model averaging in the context of deep learning because differences in hyperparameters and training batches cause different members of the ensemble to make partially independent errors.

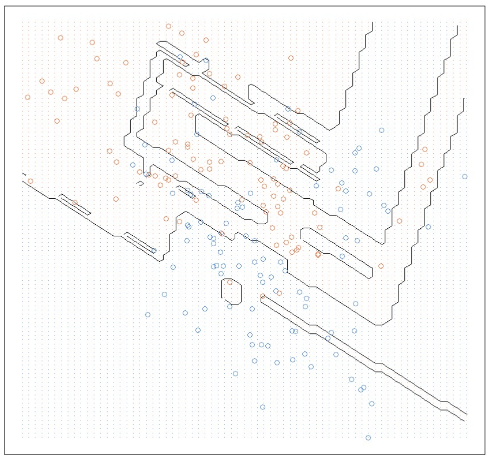
1.3 Motivation

Figure 1.1 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.

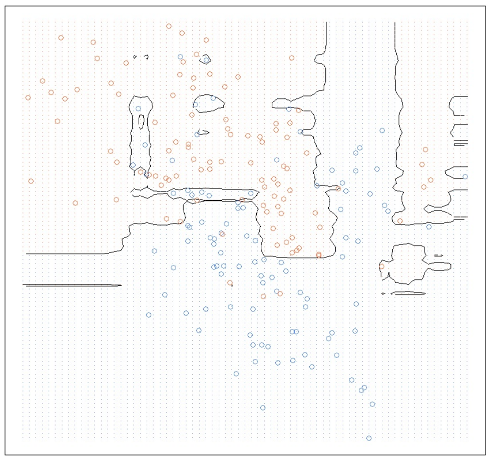
In this motivational example, we compare two approaches: (1) a standard random forest classifier and (2) a random forest classifier in which each tree is generated on a randomly rotated feature space. It is evident that the random feature rotation has a significant impact on the resulting data partition: 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 1.1 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.



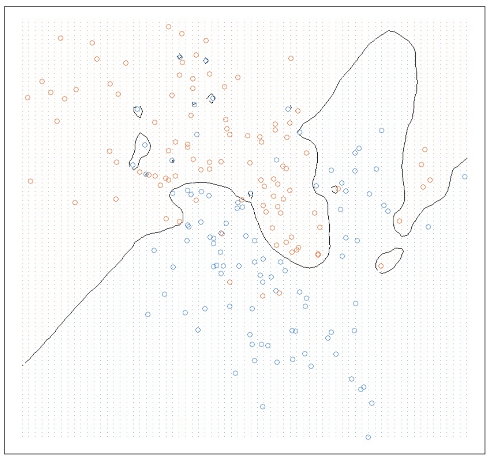
(a) **RF** (ntree=1, mtry=1)



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



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



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

Fig. 1.1: 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.

1.4 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

$$R^T = R^{-1} \text{ and } |R| = 1. \tag{1.1}$$

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 chapter 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 $\lambda \in$ [-π, π] 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

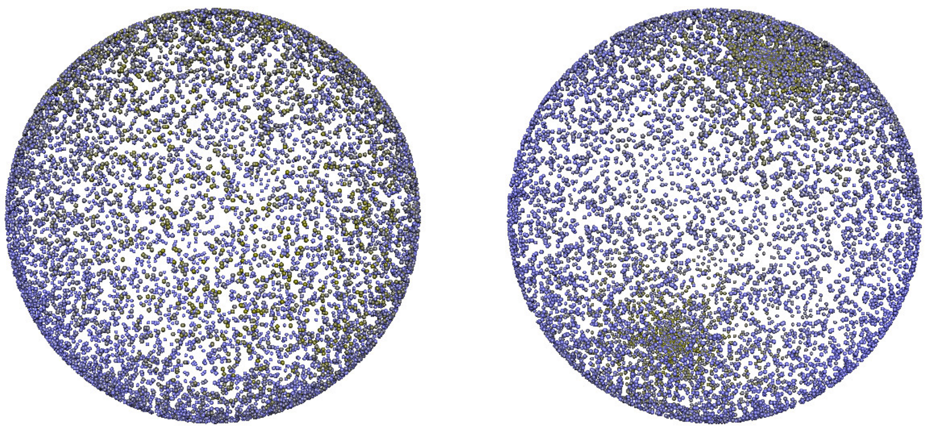

(a) Uniformly random rotations (b) Naive random rotations

Fig. 1.2: 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.

sizes of these regions, points tend to cluster together at the poles. This is illustrated for $n = 3$ in Figure 1.2, 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 $P(R \in U) = P(R \in \Gamma U)$ 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), and Ledermann and Alexander (2011). We will focus on two basic approaches to illustrate the concept.

1. (*Direct Method*) One method of obtaining random rotations involves selecting random points on the unit n-sphere directly (Knuth, 1997). This can be accomplished 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 $i \in \{1, 2, \ldots, n\}$. Note that 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 of the indirect method below and an implementation named *gsl_ran_dir_nd* is available in the GNU Scientific Library (Galassi, 2009). The most direct way to obtain the rotation matrix from the more compact random vector notation is by applying Rodrigues' rotation formula (O. Rodrigues, 1840).

2. (*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 (1.1) 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.

1.5 Generation of Random Rotations

Generating random rotations in software for problems involving fewer than 1000 dimensions is straightforward and fast, even using the simple algorithm described above. Listing 1.1 provides an example implementation of both the indirect and direct method in R. Listing 1.2 shows an example of the indirect method in C++. Both listings are 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, Jacob, 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. However, other randomization strategies, such as randomly combining subsets of the categories, are possible but not further addressed in this thesis.

The following listings provide illustrations in two commonly used programming languages for the generation of a random rotation matrix using the methods described in Section 1.4 above. The code is kept simple for illustrative purposes and does not contain error checking or performance optimizations.

Listing 1.1: 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
  return(qr.Q(QR) %*% diag(sign(diag(qr.R(QR))))) # diag(R) > 0
}

# 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)
}

# direct method of generating random member of O(n)
random_rotation_matrix_direct <- function(n) {
  a <- rnorm(n); a <- a / sqrt(sum(a^2))
  b <- c(1,rep(0,n-1))
  M <- 2*(a+b) %*% t(a+b) / sum((a+b)^2)        # Rodrigues
  return(M - diag(1,n,n))
}
```

Listing 1.2: 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);
}
```

1.6 Rotations and tree-based classifiers

Random rotations complement standard learning techniques and are easily incorporated into existing algorithms. Algorithm 1 shows a simple implementation in pseudo code. In Line 4, the random rotation matrices are computed and stored. In Line 5, the base learners are trained on the rotated data. During out-of-sample predictions, the data needs to be rotated using the same sequence of rotations. Therefore, the rotations are retrieved, applied and used for the predictions in Line 13, leveraging the stored base learners. Finally, in Line 15, the predictions made by the base learners on each rotation are combined, for example via averaging.

Algorithm 1 Random Rotation Ensemble (Pseudocode)

```
1:  procedure RANDOM_ROTATION_ENSEMBLE($R, X^{train}$)   ▷ $R$: #rotations, $X^{train}$: normalized training data
2:      ensemble ← rotations ← ()
3:      for $i$ ← 1 to $R$ do
4:          rotations[$i$] ← generate_random_rotation(dim($X^{train}$))
5:          ensemble[$i$] ← train_base_learner(rotate_data($X^{train}$, rotations[$i$]))
6:      end for
7:      return (ensemble, rotations)
8:  end procedure
9:
10: procedure RANDOM_ROTATION_ENSEMBLE.PREDICT(ensemble, rotations, $R, X^{test}$)
11:     predictions ← ()
12:     for $i$ ← 1 to $R$ do
13:         predictions[$i$] ← ensemble[$i$].predict(rotate_data($X^{test}$, rotations[$i$]))
14:     end for
15:     return combine_predictions(predictions)
16: end procedure
```

In order to examine the benefit of random rotations to ensemble performance, we modified three standard tree ensemble implementations to incorporate random rotations before the tree induction phase. The necessary modifications are illustrated in pseudo code in Listing 1.3 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

$$T(x;\theta,\Omega) = \sum_{j=1}^{J} c_j I(R(x) \in G_j), \tag{1.2}$$

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

$$c_j = \frac{1}{|G_j|} \sum_{R(x_k) \in G_j} y_k. \tag{1.3}$$

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

$$\hat{\Omega} = \arg\min_{\Omega} \sum_{j=1}^{J} \sum_{R(x_i) \in G_j} L(y_i, f(R(x_i))) = \arg\min_{\Omega} \sum_{j=1}^{J} \sum_{R(x_i) \in G_j} L(y_i, c_i). \tag{1.4}$$

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 chapter. However, conceptually it would be possible to include the rotation in the optimization in an attempt to focus on the most helpful rotations. Indeed, in Chapters 2 and 3 we will attempt to find the best rotations for a given data set and, more generally, the best method to combine predictions from multiple different rotations.

Listing 1.3: 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 $x$ (see 1.6.2):
      e.g. $x' := (x - Q_k(x))/(Q_{1-k}(x) - Q_k(x))$

(B) For m $\in$ $\{1,2,\ldots,M\}$ do
   (1) generate random parameters: $\theta_m := \{R_m, \omega_m\}$
   (2) train standard learner $B$ on $R_m(x)$:
         $\hat{\Omega}_m = \arg\min_{\Omega} \sum_{j=1}^{J} \sum_{R(x_i)\in G_j} L(y_i, f(R_m(x_i)))$
   (3) compute test or out-of-bag predictions
         $T(x, \theta_m, \Omega_m)$, $x \in R_m(S)$

(C) Aggregate predictions (vote or average)
      $f_M(x) = \sum_{m=1}^{M} w_m T(x; \theta_m, \Omega_m)$
```

In all algorithms considered in this chapter, 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 (1.4) 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

$$f_M(x) = \sum_{m=1}^{M} w_m T(x;\theta_m, \Omega_m), \tag{1.5}$$

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:

$$\hat{\Omega}_m = \arg\min_{\Omega_m} \sum_{j=1}^{J} \sum_{R(x_i)\in G_j} L(y_i, T(x_i;\theta_m, \Omega_m)). \tag{1.6}$$

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

$$\hat{\Omega}_m = \arg\min_{\Omega_m} \sum_{j=1}^{J} \sum_{R(x_i)\in G_j} L(y_i, f_{m-1}(x_i) + T(x_i;\theta_m, \Omega_m)). \tag{1.7}$$

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 chapter 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.

1.6.1 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 1.1 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.

1.6.2 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

Tab. 1.1 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.

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

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.

1.6.3 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 an implementation of rotation forests (Rodriguez et al., 2006): 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 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.

1.6.4 Results

Table 1.6 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 1.2.

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

Tab. 1.2 Ranked performance comparision 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.

	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

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 1.2) 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. A more robust approach to this challenge is described in Chapter 3.

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.

1.6.5 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 1.3 compares the performance of the standard random forest algorithm (RF) and a modified version including random feature rotation (RR-RF) on this data set.

Tab. 1.3 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.

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

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 1.4 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 $\geq$ 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.

Tab. 1.4 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.

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 1.5 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 1.3 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

Tab. 1.5 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.

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**

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 1.5, 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.

1.6.6 Derivation of Diversity

In this section we will 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

$$\bar{\rho}(\cdot) = \frac{V_{x,y}[\Psi(x,y)]}{E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta))]^2}. \tag{1.8}$$

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.

We start by defining for a given input vector x, 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

$$j^M_{max} := \arg\max_{j \neq y} P(f_M(x) = j). \tag{1.9}$$

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

$$\psi(x,y,\theta) = I(f_M(x) = y) - I(f_M(x) = j^M_{max}), \tag{1.10}$$

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

$$\begin{aligned} \Psi(x,y) &= E_\theta[\psi(x,y,\theta)] \\ &= P(f_M(x) = y) - P(f_M(x) = j^M_{max}). \end{aligned} \tag{1.11}$$

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

$$\hat{P}(f_M(x)=k)=\frac{\sum_{m=1}^{M} I(T(x;\theta_m,\Omega_m)=k \wedge (x,y)\in O_m)}{\sum_{m=1}^{M} I((x,y)\in O_m)}, \tag{1.12}$$

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

$$\hat{P}(f_M(x)=k)=\frac{1}{M}\sum_{m=1}^{M} I(T(x;\theta_m,\Omega_m)=k), \tag{1.13}$$

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

$$\hat{E}_{x,y}[\Psi(x,y)]=\hat{E}_{x,y}[\hat{P}(f_M(x)=y)-\hat{P}(f_M(x)=j^M_{max})] \tag{1.14}$$

and its variance as

$$\hat{V}_{x,y}[\Psi(x,y)]=\hat{E}_{x,y}[(\hat{P}(f_M(x)=y)-\hat{P}(f_M(x)=j^M_{max}))^2]-\hat{E}_{x,y}[\Psi(x,y)]^2. \tag{1.15}$$

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:

$$\begin{aligned} P(\Psi(x,y) < 0) &\leq P(|E_{x,y}[\Psi(x,y)] - \Psi(x,y)| \geq E_{x,y}[\Psi(x,y)]) \\ &\leq \frac{V_{x,y}[\Psi(x,y)]}{E_{x,y}[\Psi(x,y)]^2}, \end{aligned} \tag{1.16}$$

which can be estimated from equations (1.14) and (1.15). 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

$$\begin{aligned} E_{\theta_1,\theta_2}[\psi(x,y,\theta_1) \times \psi(x,y,\theta_2)] &= E_{\theta_1}[\psi(x,y,\theta_1)] \times E_{\theta_2}[\psi(x,y,\theta_2)] \\ &= \Psi(x,y) \times \Psi(x,y) \\ &= \Psi(x,y)^2. \end{aligned} \tag{1.17}$$

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

$$\begin{aligned} V_{x,y}[\Psi(x,y)] &= E_{x,y}[\Psi(x,y)^2] - E_{x,y}[\Psi(x,y)]^2 \\ &= E_{x,y}[E_{\theta_1,\theta_2}[\psi(x,y,\theta_1) \times \psi(x,y,\theta_2)]] && (1.18) \\ &\quad -E_{x,y}[E_{\theta_1}[\psi(x,y,\theta_1)]] \times E_{x,y}[E_{\theta_2}[\psi(x,y,\theta_2)]] \\ &= E_{\theta_1,\theta_2}[E_{x,y}[\psi(x,y,\theta_1) \times \psi(x,y,\theta_2)] && (1.19) \\ &\quad -E_{x,y}[\psi(x,y,\theta_1)] \times E_{x,y}[\psi(x,y,\theta_2)]] \\ &= E_{\theta_1,\theta_2}[Cov_{x,y}[\psi(x,y,\theta_1), \psi(x,y,\theta_2)]] \end{aligned}$$

$$
\begin{aligned}
&= E_{\theta_1,\theta_2}[\rho(\psi(x,y,\theta_1),\psi(x,y,\theta_2))] \times E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta_1)) \times \sigma(\psi(x,y,\theta_2))] \\
&= E_{\theta_1,\theta_2}[\rho(\psi(x,y,\theta_1),\psi(x,y,\theta_2))] \times E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta))]^2. \qquad (1.20)
\end{aligned}
$$

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

$$\bar{\rho}(\cdot) = \frac{V_{x,y}[\Psi(x,y)]}{E_{\theta_1,\theta_2}[\sigma(\psi(x,y,\theta))]^2}, \qquad (1.21)$$

where we can use (1.15) 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

$$
\begin{aligned}
E_{\theta_1,\theta_2}[\sigma(x,y,\theta)] = \frac{1}{M}\sum_{m=1}^{M} \sqrt{P(f_m(x)=y)+P(f_m(x)=j^m_{max})-} \\
\overline{(P(f_m(x)=y)-P(f_m(x)=j^m_{max}))^2}, \qquad (1.22)
\end{aligned}
$$

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

$$\hat{P}(f_m(x)=k) = \frac{1}{|S|}\sum_{(x_i,y)\in S} I(T(x_i;\theta_m,\Omega_m)=k), \qquad (1.23)$$

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

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.

1.7 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.

The numerical evaluation of random rotations in this chapter confirms that the random rotations reduce the correlation of the base learners and result in ensembles that exhibit competitive performance across a wide range of test data sets.

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. This and related techniques are discussed in Chapter 2 of this thesis.

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.

Tab. 1.6 Raw performance comparision 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).

	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

2 Regularizing axis-aligned ensembles via data rotations that favor simpler learners

There are no wrong answers only different perspectives. With that being said, some perspectives are certainly better than others.

— **Amanda Mosher**
(Author and psychotherapist)

2.1 Introduction

Feature rotations are ubiquitous in modern machine learning algorithms – from structured rotations, such as PCA, to random rotations and projections. For example, in computer vision, local image rotations are routinely used to obtain high-quality rotation-invariant features (e.g. Takacs et al., 2013). In the context of axis-aligned ensemble learning, rotations – and random projections, which can be decomposed into a random rotation and an axis-

The main findings of this chapter are based on Blaser and Fryzlewicz (2021), originally published in the Springer Journal Statistics and Computing.

aligned projection – can make the difference between a highly successful classifier and an average classifier (e.g. Durrant and Kaban, 2013).

Rodriguez et al. (2006) introduced rotation forests after demonstrating that repeated PCA-rotations of random subsets of the feature space significantly improved classification performance of random forests (Breiman, 1999) and other tree ensembles. In Chapter 1 we showed that rotation forests can be outperformed using unstructured random rotations of the feature space prior to inducing the base learners. While random rotations are used with classifiers designed for high-dimensional settings, Cannings and Samworth (2017) presented a random projection ensemble, in which the high-dimensional feature space is first projected into a lower-dimensional space before applying a classifier designed for low-dimensional settings.

An important insight from Chapter 1 and the literature described in Chapter 4 is that the vast majority of rotations are unhelpful in improving out-of-sample classifier performance. Instead, most of the benefit of these ensembles is derived from a small number of rotations that are particularly well-suited for the specific classification problem.

In this chapter we investigate the efficacy of rotations more closely and attempt to answer the question of how we can identify or construct rotations that explicitly improve classifier performance. We hypothesise that the most beneficial rotations are those that align significant segments of the decision boundary with one of the axes and thus result in simpler and more compact base learners: we call it *rotation to simplicity*. We also conjecture the converse to be true: those rotations that produce less complex base learners positively impact ensemble performance. Supporting evidence for this assertion is provided in Section 2.5.

The remainder of the chapter is organized as follows: in Section 2.2, we introduce the basic ensemble notation, as well as an extended loss function which takes into consideration the complexity of the base learners. This is similar to loss functions in linear regression that include penalties on the regression coefficients. In Section 2.3, we introduce a low-cost regularization technique, which explicitly favors rotations that are expected to produce simple base learners. Section 2.4 takes a step back and illustrates why certain rotations are better than others for axis-aligned learners and how these rotations differ from analytic methods, such as PCA. Next, we present performance results on a sample of well-known UCI data sets in Section 2.5 and conclude with our final thoughts.

2.2 Motivation

A decision tree divides the predictor space into disjoint regions G_j, where $1 \leq j \leq J$, with J denoting the total number of leaf nodes of the tree. Borrowing the notation from Hastie et al. (2009), the binary decision tree is represented as

$$T(x;\Omega) = \sum_{j=1}^{J} c_j \, I(x \in G_j), \tag{2.1}$$

where $\Omega = \{G_j, c_j\}_1^J$ are the optimisation- or tuning parameters and $I(\cdot)$ is an indicator function. Inputs x are mapped to a constant c_j, depending on which region G_j they are assigned to. A tree ensemble consisting of M trees can then be written as

$$E_M(x) = \sum_{m=1}^{M} T(x; \Omega_m). \tag{2.2}$$

In this chapter, we assume that trees are grown independently and that no co-dependence exists between the tuning parameters of different trees.

This restriction implicitly excludes boosted tree ensembles (Friedman, 2001). We will lift this restriction in Chapter 3. Our goal is then to optimise the tuning parameters Ω_m for each tree in such a way as to minimise a given loss function, $L(y_i, f(x_i))$, that is

$$\hat{\Omega}_m = \arg\min_{\Omega_m} \sum_{i=1}^{N} L(y_i, T(x_i; \Omega_m)). \tag{2.3}$$

It should be noted that the general tree-induction optimization problem in equation (2.3) is NP-complete (Hyafil and Rivest, 1976) even for two-class problems in low dimensions (Goodrich et al., 1995) and an axis-aligned, greedy tree induction algorithm such as CART (Breiman et al., 1984) is typically used to find a reasonable approximation.

At this point we depart from the standard tree ensemble setting in two aspects: (1) we add a penalty P to the loss function in order to penalize complex base learners and (2) we add rotations R_k to the input data such that groups of trees are constructed on different rotations of the input space. Hence, the loss function takes the form

$$L(y_i, f(x_i)) = \underbrace{V(y_i, f(R_k(x_i)))}_{accuracy} + \underbrace{P(R_k(x_i))}_{complexity}, \tag{2.4}$$

where the regularisation term $P(\cdot)$ penalises rotations that lead to more complex base learners. $V(y_i, f(x_i))$ is a typical loss function – such as squared-, hinge-, or logistic loss – which does not take model complexity into account (see e.g. James et al., 2013). Minimizing this combined loss function resembles constrained regression problems, such as Ridge- or Lasso-regressions (Tibshirani, 1996), but instead of constraining coefficients, we actively regularize the base learners. Lastly, the subscript k denotes the specific rotation;

we typically grow multiple trees per rotation, depending on the efficacy of the rotation: this is described in detail in Section 2.3.

With the addition of the regularisation term, we have made the problem even more challenging to solve. Since tree induction was already NP-complete to begin with, we discuss an algorithm in the following section, which strictly separates the weighting of favorable rotations that reduce model complexity from the tree induction optimization, thereby improving accuracy. Using this approach, we implicitly assume that simpler models do not lead to lower prediction accuracy, a hypothesis we show to be empirically valid in Section 2.5.

2.3 Regularization

In this section, we introduce our proposed algorithm for generating an ensemble that optimizes the use of available rotations.

Given a set of R feature rotations, we would like to build an ensemble consisting of M base learners. In order to accomplish this, the algorithm first builds tiny micro-forests of U unconstrained trees on each rotation, a low-cost operation because $U \ll M$ and $U < M/R$. Based on the statistical properties of these micro-forests, the full ensemble is constructed. Here we present the generic algorithm; in Section 2.5 we demonstrate several ways of leveraging the available statistics. For tree-based ensembles, the trees in the micro-forests can frequently be reused for the full ensemble, further reducing the amortized cost of building the micro-forests.

Algorithm 2 describes the regularized rotation procedure in detail. The integer inputs denote the desired total number of trees M in the complete

Algorithm 2 Regularized Rotation Ensemble (Pseudocode)

```
procedure REG_ROT($M, R, U$)        ▷ $M$: ensemble size, $R$: #rotations, $U$: trees per $\mu$-forest
    rotations ← obtain_rotations($R$)
    for $i \leftarrow 1$ to $R$ do
        rotations[$i$].forest ← create_unconstrained_$\mu$-forest($U$)
        rotations[$i$].complexity ← compute_complexity(rotations[$i$].forest)
    end for
    rotations ← sort(rotations, complexity)
    for $i \leftarrow 1$ to $R$ do
        rotations[$i$].numtrees ← compute_numtrees($M$, rotations)
        rotations[$i$].forest ← add_trees(rotations[$i$].forest, max(rotations[$i$].numtrees - $U, 0$))
    end for
    for $i \leftarrow 1$ to $R$ do
        for $j \leftarrow 1$ to rotations[$i$].numtrees do
            ensemble.forest ← extend_forest(ensemble.forest, rotations[$i$].forest[$j$])
        end for
    end for
    return ensemble
end procedure
```

ensemble, the number of available (or generated) rotations R, and the number of trees U created for each micro-forest.

In line 2 of Algorithm 2, the available rotations are stored in an array named *rotations*. It is important to include the identity rotation here to make sure the procedure returns high-quality results when the problem is already optimally rotated to begin with. If too few rotations are available, the procedure can generate random rotations in addition to the identity rotation (Anderson et al., 1987).

In lines 4-5, an unconstrained, unpruned micro-forest consisting of U trees is grown. The recommended default value of U is of the order of 10-20 trees. The purpose of these trees is merely to obtain a reliable estimate of the median complexity of a representative tree that will be grown on the particular rotation, with minimal interference from outliers.

Our main proposal in this chapter is to apply a complexity measure for base learners and use it to rank the obtained rotations from the best one, which corresponds to the least complex learners, to the worst one that corresponds to the most complex learners. In the case of tree ensembles, we suggest a complexity measure $C(\cdot)$ whereby trees with a smaller number of decision nodes ($\#nodes$) are considered less complex and, among trees with the same number of nodes, more shallow trees ($depth$) are considered less complex, that is

$$C(T(x;\Omega_m)) = \#nodes + depth/N, \tag{2.5}$$

where $\#nodes = 2J-1$ and $depth \leq J$ for binary decision trees, both depending on Ω_m. N is the number of data points and J the number of leaf nodes in the tree. It is clear that $1 \geq depth/N$ and, consequently, that $depth$ merely acts as a tie-breaker for trees of equal size. We further discuss tree complexity in Section 2.4.1. Up to this step, only model complexity was used to quantify rotations and no performance metrics were considered; this corresponds to the right-most section of formula (2.4).

The sorting procedure in line 7 of Algorithm 2 arranges the rotations into ascending order of complexity C. At this point, there are several ways of using this information. In section 2.3.1 we apply a parametric, non-increasing family of curves with a tuning parameter h and use the out-of-bag (OOB) errors of the micro-forests to determine the optimal parameter in a grid search. However, as we will show in Section 2.5, it is also possible to use the ranking on its own, without combining it with predictive performance. The key point here is that whatever procedure we use, it will determine the number of base learners that need to be created for each rotation. This is accomplished in line 9 of Algorithm 2.

Should additional trees (beyond the U already available trees on each rotation) be needed, these are generated and added to the rotation in line 10. Typically, these need to be added to the most favorable rotations.

Finally, the equal-weighted ensemble is constructed from the trees on the different rotations. It is important to note that while the individual trees are equal-weighted in the ensemble, more trees are used from favorable rotations and hence the rotations are not equal-weighted. Also note that $\sum_{i=1}^{R}$ rotations[i].numtrees $= M$.

2.3.1 Weighting of rotations

Given an *ordered* sequence of R rotations ($r = 1$ for the most favorable rotation and $r = R$ for the least attractive rotation) and a specified total number of base learners M in the ensemble, we need to determine how many base learners to train on each rotation. This corresponds to line 9 in Algorithm 2 above. We now discuss the details of this procedure.

Any sensible (percentage) weighting scheme will have the following three properties:

1. $w(r) \geq 0, \forall r$

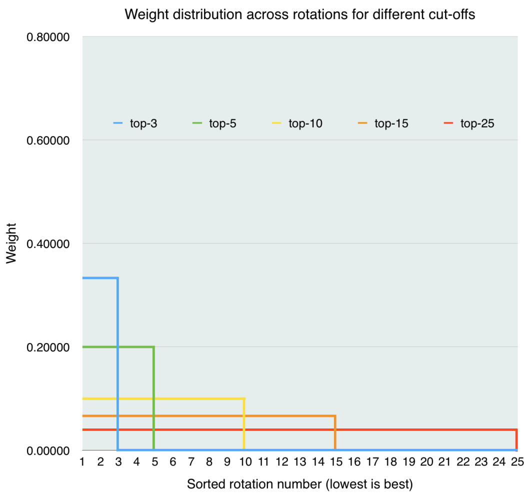
2. $w(r) \geq w(r+1)$

3. $\sum_{r=1}^{R} w(r) = 1$

We consider two parametric weighting schemes that meet these criteria:

- Select the first h rotations from the ordered list and generate a fraction of exactly $w(r) = 1/h$ of the required M base learners on each of these rotations;

- Use an exponential family of curves with decay parameter h to determine the percentage of base learners that should be trained on each rotation.



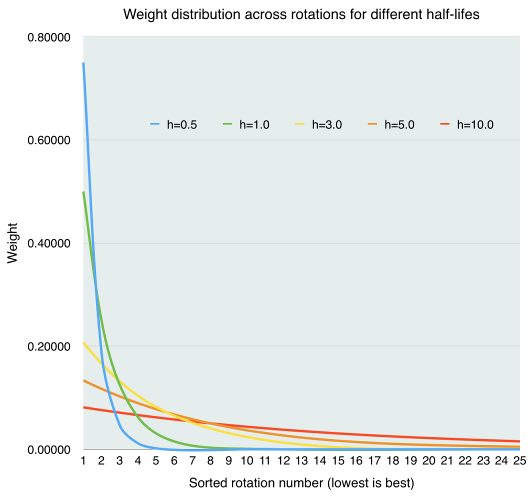


Fig. 2.1: Two parametric families of weight functions: top-h (left) and exponential (right).

The first scheme corresponds to selecting the h rotations that are expected to produce the lowest complexity base learners and equal-weight the trees on these rotations. The second scheme includes the possibility of including trees on more different rotations but at much smaller weights. In both cases, h acts as a tuning parameter that can be inferred from the data via a simple grid search, the details of which will be described at the end of this section.

In the first case, the weighting follows the formula

$$w_{cut}(R, h; r) = I(r \leq h)/h, \tag{2.6}$$

where h is an integer tuning parameter in $[1, R]$, representing a cut-off value and $I(\cdot)$ is the indicator function. Note that the sum of the weights is 1, as expected. For the second case, we use the following family of exponential curves:

$$w_{exp}(R, h; r) = \frac{2^{-r/h}(2^{1/h} - 1)}{(1 - 2^{-R/h})}, \tag{2.7}$$

where R is the total number of rotations and h is a positive, real tuning parameter. In both cases, r is the sorted (integer) rotation number, as described above. In both cases, small values of h result in large weights for the top rotations and small (or zero) weights for less favorable rotations. By contrast, large h eventually lead to the equal-weighting of rotations.

Figure 2.1 compares the two weighting schemes. A simple method for obtaining a good tuning parameter h is to use the out-of-bag error estimates of the micro-forests on each rotation and compute the sum product of these errors with the weight vectors using different values of h – effectively a grid search. Since the rotations are in complexity-sorted order and because the weighting schemes are non-increasing, the resulting weighting will differ significantly from a weighting based solely on out-of-bag predictive accuracy. In Section 2.5 we show that weightings based solely on out-of-bag predictive accuracy produce base learners that are more complex on average, without a corresponding out-of-sample performance gain. It should also be noted that in line 9 of Algorithm 2, the weights are multiplied with the ensemble size N and need to be rounded to integer values, since we cannot grow partial trees. In this process, it is possible due to rounding that the sum of the computed number of trees does not add up to N anymore. If this is the case, we automatically add any missing trees to the top rotation or analogously subtract additional trees starting from the worst rotation.

2.4 Defining useful rotations

The goal of this section is to provide an intuitive understanding of which rotations are useful in the context of axis-aligned learners. The discussion applies to higher-dimensional problems but is illustrated in two dimensions.

One visual indication betraying axis-aligned learners, such as decision tree ensembles, is their rugged ("stair-shaped") decision boundary. When a segment of the true decision boundary is not axis-aligned, such learners are forced to approximate the local boundary using a number of smaller steps. The greatest number of such steps is required when the true boundary occurs at a 45-degree angle to one of the axes.

A natural strategy to overcome this predicament is to rotate the space by 45-degrees, such that the decision boundary becomes axis-aligned. After the rotation, only a single hyperplane is necessary to represent the very segment that required many steps prior to rotation. Unfortunately, while rotating the feature space might improve classification locally, it may actually have a negative effect overall, as other segments of the decision boundary might have been well-aligned with the axes prior to rotation but are now poorly-aligned after rotation. For this reason, rotations need to be examined globally and jointly.

To better illustrate the argument, we artificially construct the 2-dimensional, two-class classification problems depicted in Figure 2.2. For simplicity of the

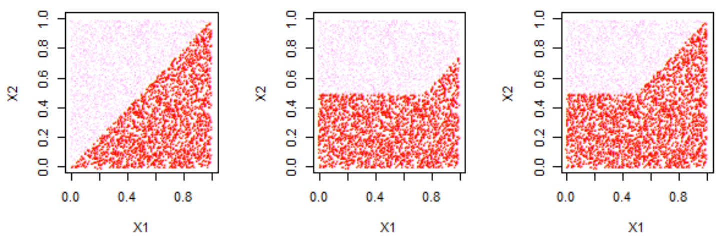


Fig. 2.2: Three illustrative classification problems with known decision boundaries: (a) $y = x$, (b) $y = 0.5 + \max(x - 0.75)$, (c) $y = 0.5 + \max(x - 0.5)$

argument, no class overlap is created but the conclusion will be unaffected. The problem on the left-hand side (a) corresponds to the situation where the decision boundary is at a 45-degree angle to both axes. For this problem, we expect a 45-degree (or equivalent) rotation to be optimal.

For the middle problem (b), the decision boundary is flat and axis-aligned but there is a small segment that protrudes at a 45-degree angle to the axes. A zero-degree rotation (or equivalent) seems ideal for the longer segment but a 45-degree rotation appears preferable for the smaller segment. Note that since we are running an ensemble of trees, it would be perfectly acceptable to combine one forest trained without rotation with another (perhaps smaller or down-weighted) forest on the rotated space. The question is: which approach produces a better-performing ensemble?

In the final classification problem on the right-hand side (c), the portion of the decision boundary at a 45-degree angle is slightly longer than the axis-aligned section. Here, rotation is likely preferred again. But is it better to rotate by 45-degrees to aid classification near the longer segment or perhaps just by 20-degrees, in such a way that the maximum slope of the decision boundary is reduced at the expense of constructing a problem that is completely unaligned to any axis? In order to answer these questions, we need to define a metric to quantify the value-add provided by a given rotation.

2.4.1 Tree Complexity

The number of steps required – and hence the average number of nodes required to form a decision tree – generally increases as the boundary becomes less aligned with the axis. This is because the tree construction is done recursively and a new level of the tree is built whenever the local granularity of the tree is insufficient to fully capture the details of the local decision boundary.

For this reason, we propose to use the expected median size of a decision tree as our metric of utility for a given rotation. Rotations that result in smaller, shallower trees on average are considered better rotations. Not only does the metric in Formula (2.5) assist in creating streamlined trees with fewer spurious splits, it also reduces the computational burden of actually generating and running the full forest. In addition, once we apply Metric (2.5) to all generated rotations, we are in a position to obtain a ranking of the relative usefulness of each rotation.

In order to compute a reliable and consistent (across rotations) estimate of the proposed metric, we generate a micro-forest for each rotation. It is necessary to create multiple trees to counteract the randomness that is injected in the tree-induction process. For each micro-forest we then compute the median number of nodes used. We use the median in order to actively ignore trees that are artificially inflated by poor (random) variable selections. These operations are computationally efficient when compared to generating a full-blown tree ensemble for each rotation and can generate a stable estimate of the true median. Based on our experiments, the complexity rankings computed on the basis a 10-20-tree forest is very similar to the complexity ranking computed on the basis of a full forest. Hence, the metric is highly predictive and useful.

2.4.2 Illustration

To demonstrate the usefulness of the proposed metric, we have generated 100 random rotations for each of the two dimensional classification problems listed in Figure 2.2 above. Figures 2.3, 2.4 and 2.5 illustrate cases (a), (b) and (c), respectively, ranked by tree complexity. Note that the sorting is entirely based on the tree complexity and, importantly, does not make use of the predictive performance of these trees. Despite this, it is interesting to see that the sort reflects our intuition: in Figures 2.3 and 2.4, those rotations for which one of the feature boundaries is aligned with one of the axis achieve the best scores, while diagonal boundaries achieve the worst scores. This allows us to find useful rotations without resorting to structured rotations (such as PCA) commonly used in other approaches.

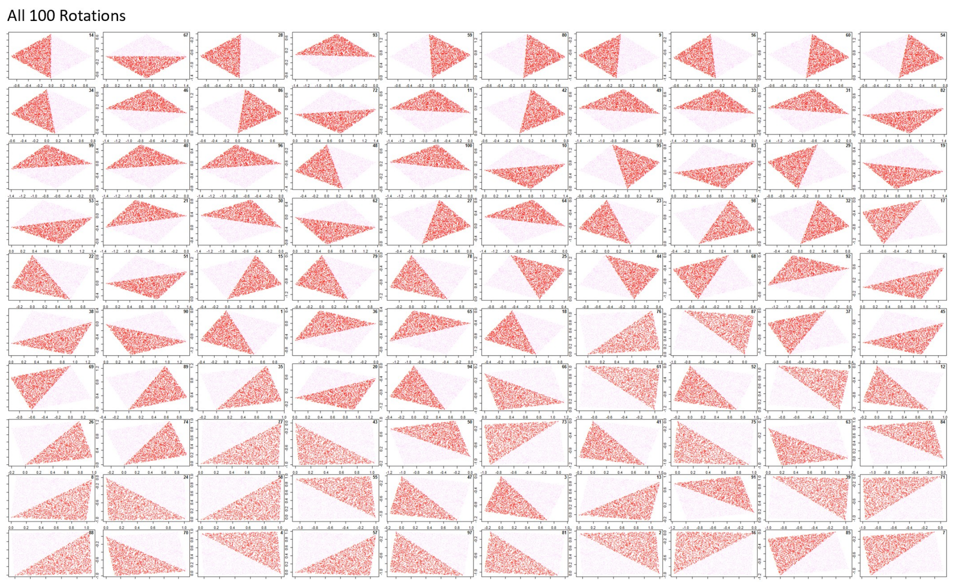


Fig. 2.3: Rotations of classification problem (a) in Figure 2.2, sorted by expected tree height, as described in the text. Top left is the best rotation, bottom right represents the worst rotation. The small number on the top right of each image is the unique rotation number.

However, if all of the top rotations were chosen solely on the basis of the largest segment of the decision boundary that is axis-aligned, important secondary segments might get neglected, ultimately leading to a worse overall prediction. Figure 2.5 demonstrates that this is not the case. In this case, the best five rotations again aligned the longer segment to one of the axis, as expected. However, the 6th rotation aligned the shorter segment to the y-axis. This illustrates the point that it may be useful to include multiple rotations in an ensemble, since different rotations can specialize on specific sub-features or decision boundary segments. These results are intuitive and demonstrate the usefulness of the tree-based ranking. What is also striking is that the best rotations do not at all resemble a PCA rotation. This is because the rotation is optimized for alignment of the decision boundary with the tree rather than for the variance of the covariates. This is what sets random rotations apart from rotation forests.

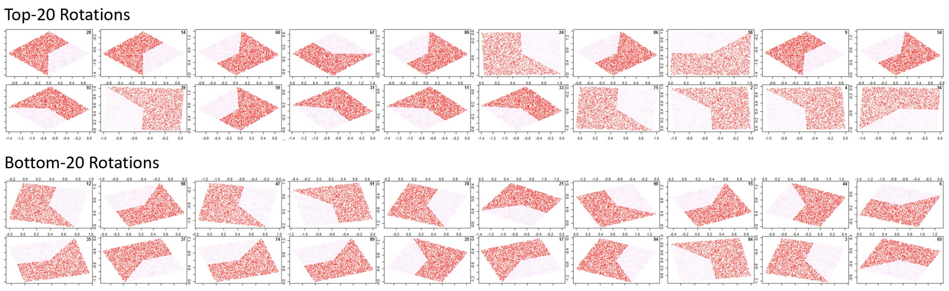


Fig. 2.4: Rotations of classification problem (b) in Figure 2.2, sorted by expected tree height, as described in the text. The top panel shows the twenty best ranked rotations from top left to bottom right, the bottom panel represents the twenty worst ranked rotations from bottom right to top left. The small number on the top right of each image is the unique rotation number.

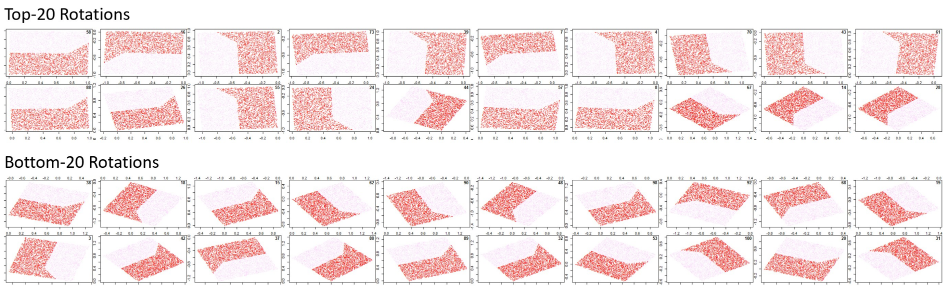


Fig. 2.5: Rotations of classification problem (c) in Figure 2.2, sorted by expected tree height, as described in the text. the top panel shows the twenty best ranked rotations from top left to bottom right, the bottom panel represents the worst ranked rotations from bottom right to top left. The small number on the top right of each image is the unique rotation number.

Up to this point, we examined some very simple two-dimensional toy problems with high signal-to-noise ratios (SNRs). In each case, both dimensions were highly informative and contained minimal noise. This setup is ideal for illustrating the method but is not representative of most real-world challenges. Therefore, an important question is how the method performs when we increase the dimensionality or decrease the SNR. To answer this question, we start again with the triangular base shape (a) but incrementally add uniform noise dimensions to the problem before applying the proposed method. In this setting, it is more difficult to visualize the results but we can still demonstrate alignment of the decision boundary with one of the axes by projecting the rotated problem onto the two-dimensional planes formed by the axes – the coordinate surfaces – before plotting. It is important to note that these are *different projections of the same rotation,* rather than different rotations.

Figure 2.6 demonstrates that the proposed approach is still successful in higher dimensions and with lower signal-to-noise ratios. In these figures, each row represents an exhaustive list of projections onto the coordinate surfaces for a single rotation in p dimensions. The first two dimensions are always the signal dimensions, while the remaining $p-2$ dimensions are random noise dimensions. For example, in the second row of Figure 2.6 we started with the two original signal dimensions plus one random noise dimension ($p=3$). We then generated 100 rotations and selected the one rotation that was ranked best according to the metric described in Section 2.3. The row shows the three two-dimensional projections of this best ranked rotation onto the (x,y), (x,z) and (z,y) planes, respectively. It is very apparent that the best rotation aligns the decision boundary with the third axis (the z-coordinate) in this case.

Even when the number of noise dimensions exceeds the number of signal dimensions, as is the case for $p=5$, the alignment of the decision boundary with one of the axes is still very consistent for the best rotation.

In contrast, Figure 2.7 shows that the worst ranked rotations are not aligned with any axis, regardless of the dimensionality of the problem and that there is a considerable overlap in the two classes at the decision boundary, making it extremely difficult to produce a successful classifier. These examples very clearly show the value of finding high-quality rotations.

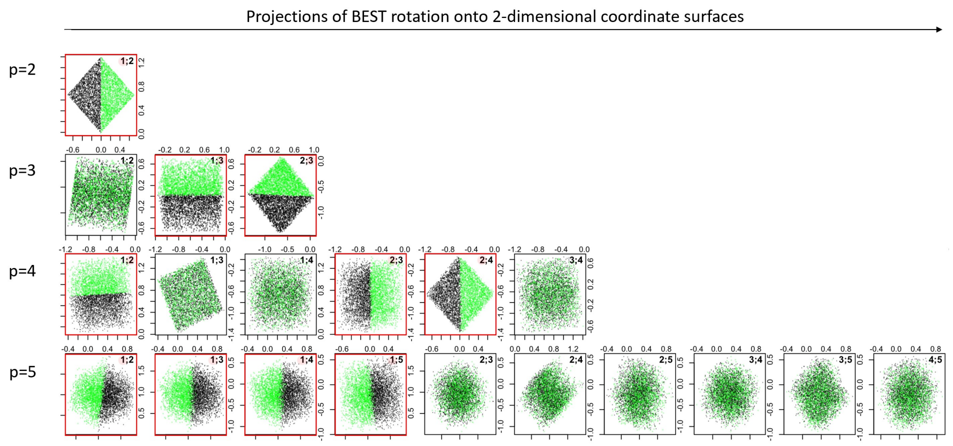


Fig. 2.6: Alignment of the decision boundary of the *best* ranked rotation in p dimensions with each axis. In p dimensions, there are exactly $p(p-1)/2$ coordinate surfaces, meaning two-dimensional planes formed by the p coordinate axes. In this figure, each row depicts the projections of the best rotation in p dimensions onto all available coordinate surfaces. The numbers on the top right of each sub-figure indicate the two axes used to form the specific coordinate surface. For $p = 2$, the signal-to-noise ratio (SNR) is high because only the two signal dimensions were used. For $p > 2$ a total of $p-2$ noise dimensions were added, decreasing the SNR accordingly. Highlighted projections indicate strong alignment with one of the axes.

2.5 Performance

In order to test our hypothesis that it is possible to rotate to simplicity without a corresponding performance penalty, we implemented the following weighting schemes:

(a) RRE: Random rotation ensemble, same number of trees on each rotation: M/R.

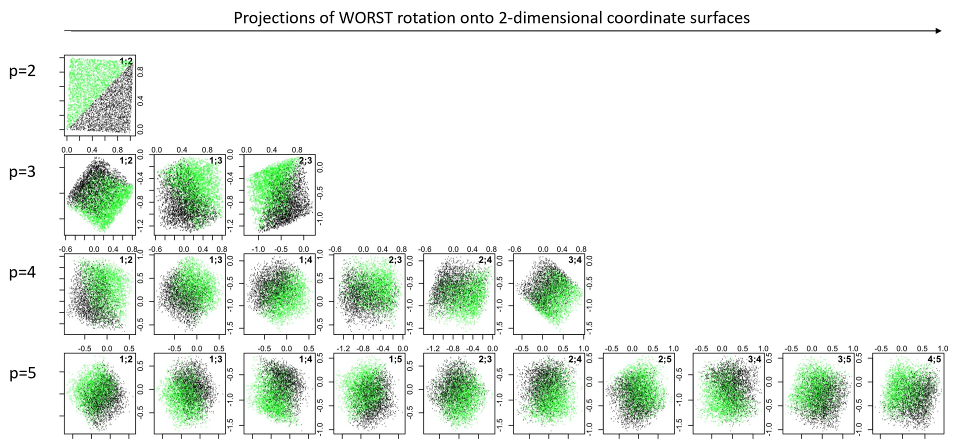


Fig. 2.7: Alignment of the decision boundary of the *worst* ranked rotation in p dimensions with each axis. In p dimensions, there are exactly $p(p-1)/2$ coordinate surfaces, meaning two-dimensional planes formed by the p coordinate axes. In this figure, each row depicts the projections of the worst rotation in p dimensions onto all available coordinate surfaces. The numbers on the top right of each sub-figure indicate the two axes used to form the specific coordinate surface. For $p = 2$, the signal-to-noise ratio (SNR) is high because only the two signal dimensions were used. For $p > 2$ a total of $p-2$ noise dimensions were added, decreasing the SNR accordingly. No alignment is apparent with any of the axes for the worst ranked rotation.

(b) CUT: Same number of trees on the top-h rotations in terms of complexity (h is chosen using grid search on OOB performance per section 2.3.1).

(c) EXP: Exponential weighting with half-life h in terms of complexity (h is chosen using grid search on OOB performance per section 2.3.1).

(d) BST: All N trees on the lowest complexity (best) rotation (equivalent to CUT with $h = 1$.

(e) NEW: Same number of trees on all rotations that are ranked higher than or equal to the identity rotation.

(f) LIN: linearly decreasing number of trees: k on lowest complexity rotation, $k-1$ on second lowest, $\ldots 1$ on highest complexity.

(g) OOB: linearly decreasing number of trees: k on lowest OOB error rotation, $k-1$ on second lowest, $\ldots 1$ on highest OOB error.

(h) JNT: linearly decreasing number of trees: k on lowest joint ranking of complexity and OOB error, $\ldots 1$ on highest joint ranking rotation: rank(rank(OOB error) + rank(complexity)).

For comparison, we also tested a standard Linear Discriminant Analysis (LDA), as well as three non-linear classifiers: a simple K-Nearest Neighbor classifier (KNN-5), a Support Vector Machine (SVM), and a Gaussian Process classifier (GPR). We have applied the competing methods (GPs and SVMs) with default parameters available from publicly available software implementations.

For KNN, we used the R implementation in the *class* package with k=5 and for LDA the implementation in *MASS*. For the SVM, we used the R implementation in the *e1071* package with default parameters, that is we used type C-classification with a radial basis function (RBF) kernel and a default gamma of 1/N, which was adjusted to reflect the number of data dimensions and added noise dimensions, where applicable. The cost parameter (or C-parameter in SVM parlance) was set to 1.0. For the GPR, we used the R implementation gausspr in the *kernlab* package. Here we too used the problem type classification with a RBF kernel (rbfdot) and took advantage of the built-in automatic sigma estimation (sigest). We did not attempt to manually- or otherwise tune the meta-parameters of these methods, unless a built-in auto-tuning feature was available, just like we did not tune any parameters in the proposed tree-based methods with the exception of the rotation selection that is the subject of this thesis chapter. The overarching goal was to compare methods with sensible default parameters across a

Tab. 2.1 Description of UCI Datasets

UCI Name	Dim (p)	Rows (N)
BREAST	10	699
ECOLI	8	336
GLASS	10	214
IONO	34	351
IRIS	4	150
LIVER	7	345
WINE	13	178
WAVE	21	5000

number of problem sets in order to determine how to best make use of rotations with axis-parallel learners.

The test procedure generated a random subset of 70% of the data for training purposes and all classifiers were tested on the remaining 30% of the data. This is consistent with the procedure described in Chapter 1. The process was repeated 100 times and averages are reported.

With the exception of the identity rotation, all rotations were generated uniformly at random from the Haar distribution. As our base case RRE, we implemented a random rotation ensemble, which does not differentiate between rotations. The only other weighting scheme that does not consider tree complexity at all is OOB, which only takes advantage of out-of-bag errors across the different rotations. Our expectation would be for OOB to outperform in terms of predictive accuracy but with high complexity ensembles. We would also expect BST to produce the lowest complexity ensemble but at the cost of lower predictive performance. In terms of methodology, we first generated 100 random rotations, including one identity rotation. These same rotations were then used by all weighting schemes before the entire process was repeated. In each case, we generated an ensemble with exactly $M = 5000$ trees in total. The dimensionality and

Tab. 2.2 Classification error on test data (lower is better). Each rotation weighting method RRE, CUT, EXP, BST, NEW, LIN, OOB, JNT, as described in the text, was applied to the data sets listed under UCI Name. Strikethrough represents a performance number that was more than one cross-sectional standard deviations above (worse than) the minimum.

UCI Name	LDA	SVM	GPR	KNN-5	RRE	CUT	EXP	BST	NEW	LIN	OOB	JNT
BREAST	~~0.0472~~	~~0.0386~~	~~0.0407~~	~~0.0410~~	0.0347	0.0348	0.0349	0.0354	0.0348	0.0349	0.0345	0.0347
ECOLI	0.1209	0.1265	~~0.1305~~	~~0.1365~~	0.1217	0.1233	0.1251	~~0.1445~~	~~0.1448~~	0.1208	0.1212	0.1207
GLASS	~~0.3805~~	~~0.3157~~	~~0.3213~~	~~0.3587~~	~~0.3001~~	0.2729	0.2400	0.2395	0.2378	~~0.2977~~	~~0.2994~~	~~0.3001~~
IONO	~~0.1381~~	0.0609	~~0.1317~~	~~0.1587~~	0.0514	0.0535	0.0544	0.0704	0.0706	0.0519	0.0510	0.0512
IRIS	0.0240	~~0.0407~~	~~0.0502~~	~~0.0416~~	~~0.0467~~	~~0.0451~~	~~0.0453~~	~~0.0456~~	~~0.0460~~	~~0.0467~~	~~0.0462~~	~~0.0464~~
LIVER	~~0.3318~~	0.3064	0.3162	~~0.3981~~	0.3116	0.3050	0.3135	0.3191	0.3089	0.3097	0.3079	0.3099
WAVE	0.1423	0.1383	0.1320	~~0.1815~~	0.1352	0.1380	0.1422	0.1435	0.1432	0.1352	0.1350	0.1351
WINE	0.0161	0.0206	0.0180	~~0.0433~~	~~0.0250~~	0.0176	0.0165	0.0172	0.0150	0.0222	~~0.0224~~	~~0.0232~~

number of data points for each data set is listed in table 2.1. The lowest-dimensional problem with 4 predictors is IRIS and the highest-dimensional problem with 34 predictors is IONO. We refer the reader to Dheeru and Taniskidou (2017) for a detailed description of the UCI data sets we used for testing. Before running the classification algorithms, we scaled all numeric predictors to $[0, 1]$.

Table 2.2 shows the names of the data sets, together with the classification error resulting from applying the different weighting schemes to the rotations. Interestingly, algorithm OOB did not perform quite as well as we had anticipated. For three of the data sets, the scheme performed more than one cross-sectional standard deviation above (worse than) the minimum error. In fact, this appears to be a common pattern among these methods, except for CUT and EXP described in Section 2.3.1, which are competitive on most of these data sets. One interesting exception was the IRIS data set, for which LDA outperformed all variants of the rotation-based ensembles and indeed all non-linear classifiers. This is an example of where the proposed method does not work as well as expected.

In Table 2.3 we can confirm that BST really does produce the most compact ensembles. However, unfortunately performance suffers accordingly. A good

Tab. 2.3 Tree complexity on test data (lower is better, only relevant for tree-based classifiers). Each rotation weighting method RRE, CUT, EXP, BST, NEW, LIN, OOB, JNT, as described in the text, was applied to the data sets listed under UCI Name. Strikethrough represents a complexity number that was more than one cross-sectional standard deviations above (worse than) the minimum.

UCI Name	RRE	CUT	EXP	BST	NEW	LIN	OOB	JNT
BREAST	~~54.69~~	52.08	51.68	51.11	52.94	53.66	~~54.42~~	~~54.63~~
ECOLI	~~76.82~~	~~71.67~~	69.08	44.52	44.56	~~75.23~~	~~76.68~~	~~76.59~~
GLASS	~~81.68~~	73.69	67.42	66.93	66.95	~~80.84~~	~~81.14~~	~~81.39~~
IONO	~~61.58~~	60.24	60.02	57.58	58.20	~~61.16~~	~~61.51~~	~~61.60~~
IRIS	~~20.98~~	16.52	15.86	15.36	18.67	19.44	~~20.30~~	~~20.98~~
LIVER	~~115.48~~	112.14	111.18	110.84	114.06	~~114.64~~	~~114.94~~	~~115.51~~
WAVE	~~1368.76~~	~~1332.53~~	1310.13	1303.26	1303.29	~~1359.99~~	~~1365.58~~	~~1368.30~~
WINE	~~36.21~~	32.52	31.02	30.88	31.21	~~35.42~~	~~35.42~~	~~36.15~~

compromise is EXP, which shows significant reductions in complexity without suffering from performance problems.

For the IRIS data set, EXP resulted in an ensemble that outperformed RRE despite a 24.4% decrease in complexity. Similarly, a 17.5% decrease in complexity was achieved in the GLASS data set. The smallest improvement of merely 2.5% decrease in complexity occured on the IONO data set, for which RRE actually outperformed EXP, although not in a statistically significant manner.

Tables 2.4 and 2.5 show the performance of a set of baseline classifiers (SVM, GPR, KNN-5) and the various rotation variants after adding noise dimensions to the data sets IRIS and IONO. It is evident that the performance of the rotation-based classifiers deteriorates relative to other classifiers as the signal-to-noise ratio decreases. This is a known limitation of the method, further described in the following section. At the same time, it can be observed that LDA performance is very problem dependent, while KNN and SVM classifiers actually became more competitive in a relative sense with decreasing SNR.

2.6 Limitations

It was empirically demonstrated in Tomita, Maggioni, et al. (2017) that in situations where the signal is contained in a subspace that is small relative to the dimensionality of the feature space, random rotation ensembles tend to underperform ordinary random forests. This is because such a setup renders most rotations unhelpful. By overweighting the most successful rotations, as we propose in this chapter, this effect is somewhat mitigated but not entirely eliminated.

Even in the illustrations in Figure 2.6 it is clear that the quality of the most successful rotations decreases marginally as the number of noise dimensions is increased. The alignment with the axes are not perfect and the noise around the decision boundaries increases visibly. Nonetheless, the rotated features lead to better (axis-aligned) classifiers than the those trained on the unrotated space.

The underlying issue is that rotations in the direction of uninformative noise dimensions do not improve predictions and when the number of noise dimensions is large relative to the signal dimensions, the likelihood of rotating in uninformative directions increases. Note that the same is not necessarily true when the SNR is decreased without increasing the dimensionality of the problem. In this case, random rotations and the ideas in this thesis *do not* underperform ordinary random forests in our experience.

One important consideration when introducing rotations into a classifier is that features need to be of comparable scale. We do not explicitly mention this in this chapter but a section on recommended scaling mechanisms can

Tab. 2.4 Classification error on test data (lower is better). Each rotation weighting method and control classifier was applied to the UCI data set IRIS. The first (NOISE) column indicates the number of noise dimensions added to the original data set, from which an upper bound to the signal-to-noise ratio can be estimated as SNR (dB) $\leq 10 \times \log(4/NOISE)$ by assuming that the original data set is noise free. In the table, SNR represents this upper bound.

NOISE	SNR	LDA	SVM	GPR	KNN-5	RRE	CUT	EXP	BST	NEW	LIN	OOB	JNT
1	6.02	0.0258	0.0578	0.0658	0.0507	0.0516	0.0489	0.0516	0.0560	0.0524	0.0507	0.0524	0.0516
2	3.01	0.0258	0.0533	0.0898	0.0480	0.0569	0.0551	0.0551	0.0587	0.0613	0.0569	0.0604	0.0578
4	0.00	0.0240	0.0827	0.1164	0.0871	0.0853	0.0693	0.0631	0.0640	0.0640	0.0853	0.0889	0.0853
8	-3.01	0.0267	0.1591	0.1680	0.1467	0.1316	0.1093	0.0960	0.0996	0.1013	0.1333	0.1289	0.1289
16	-6.02	0.0533	0.1591	0.1822	0.1200	0.1458	0.1396	0.1511	0.1671	0.1440	0.1440	0.1449	0.1422
32	-9.03	0.0569	0.1618	0.1724	0.1458	0.1591	0.1636	0.1733	0.1724	0.1618	0.1609	0.1644	0.1671
64	-12.04	0.1280	0.2222	0.2516	0.2249	0.2124	0.2116	0.2204	0.2364	0.2151	0.2151	0.2116	0.2133
128	-15.05	0.1618	0.2951	0.3484	0.2596	0.2640	0.2676	0.2684	0.2898	0.2729	0.2658	0.2791	0.2676

Tab. 2.5 Classification error on test data (lower is better). Each rotation weighting method and control classifier was applied to the UCI data set IONO. The first (NOISE) column indicates the number of noise dimensions added to the original data set, from which an upper bound to the signal-to-noise ratio can be estimated as SNR (dB) $\leq 10 \times \log(34/NOISE)$ by assuming that the original data set is noise free. In the table, SNR represents this upper bound.

NOISE	SNR	LDA	SVM	GPR	KNN-5	RRE	CUT	EXP	BST	NEW	LIN	OOB	JNT
1	15.31	0.1479	0.0675	0.1423	0.1725	0.0600	0.0604	0.0600	0.0721	0.0706	0.0596	0.0592	0.0592
2	12.30	0.1234	0.0551	0.1294	0.1509	0.0521	0.0558	0.0589	0.0702	0.0691	0.0528	0.0521	0.0517
4	9.29	0.1577	0.0725	0.1464	0.1675	0.0675	0.0683	0.0702	0.0868	0.00875	0.0679	0.0698	0.0683
8	6.28	0.1506	0.0668	0.1408	0.1706	0.0668	0.0660	0.0717	0.0808	0.0815	0.0664	0.0668	0.0679
16	3.27	0.1438	0.0634	0.1362	0.1672	0.0657	0.0615	0.0747	0.0789	0.0785	0.0649	0.0664	0.0642
32	0.26	0.1642	0.0792	0.1638	0.1864	0.0864	0.0808	0.0879	0.0906	0.0838	0.0860	0.0857	0.0842
64	-2.75	0.1951	0.1155	0.2257	0.1898	0.1521	0.1408	0.1385	0.1423	0.1442	0.1525	0.1498	0.1528
128	-5.76	0.2921	0.1479	0.2687	0.2083	0.2230	0.2242	0.2200	0.2208	0.2211	0.2238	0.2245	0.2242

be found in Chapter 1. We do not recommend using any rotation-based ensembles without prior scaling or ranking for practical problems.

2.7 Computational Considerations

When compared to random rotation ensembles, there is an additional computational cost for regularizing the ensemble. Given the desired total number of trees M, the algorithm requires the generation of micro-forests of size U for each of the R rotations. These micro-forests are essential for estimating the relative efficacy of each rotation. However, depending on the weighting scheme employed, only a subset of the rotations is actually included in the final model.

More specifically, in the initial step, $U \times R$ trees are constructed. However, if the weighting scheme only involves the top r rotations, then $(R-r) \times U$ trees subsequently get discarded. This, in turn, implies that $M - r \times U$ additional trees need to be induced within the r selected rotations to end up with M trees in total within the selected rotations. Expressed as a percentage, we know that $(R-r) \times U/(M - r \times U + R \times U)$ percent of the initially constructed trees get subsequently discarded, resulting in computational overhead when compared to random rotation ensembles, where all trees are used.

In order to obtain a bound on this expression, note that $R \times U <= M$. This is because it is not practical to generate more trees in the micro-forests than are needed in total. Hence, in the worst case $(R-r)/(2R-r)$ percent of the initially constructed trees get subsequently discarded, a quantity that is smaller than $1/2$ because r is in $[1, R]$ and R is in $[1, M]$. This expression is maximized when only the best rotation is selected $(r = 1)$ and minimized when all rotations are selected $(r = R)$. Therefore, in terms of computational overhead, the worst case is that nearly twice as many trees need to be constructed when the ensemble is regularized than for standard random rotation ensembles.

In practice, this bound is unrealistically high and the magnitude of the overhead can be influenced by selecting sensible parameters. For example, using $M = 5000$, $R = 50$ and $U = 10$ and utilizing the top $r = 10$ rotations, we achieve a computational overhead of merely $(50-10) \times 10/(5000 - 10 \times 10 + 50 \times 10) = 2/27$, or less than 7.41%. In addition, it should be noted that this overhead gets partially offset by the fact that only R rotations need to be generated with our method instead of M for random rotation ensembles. In the current example, that number is 50 instead of 5000.

Besides these rather modest effects, the computational complexity of our method is equivalent to that of random rotation ensembles, regardless of the number of training samples or data dimensions.

2.8 Software

As indicated in the introduction to Chapter 1, an R package called *random.rotation* is provided in the public domain on GitHub. The weighting- and regularisation methods described in this chapter are also part of this package. A detailed introduction to this software package is provided in Chapter 5.

3

Multi-rotation classifiers and rotation stacking

It is better to solve the right problem approximately than to solve the wrong problem exactly.

— **John Tukey**
Statistician

3.1 Background

In Chapter 1 on Random Rotation Ensembles, we concluded that random rotations can have a significant impact on the performance of ensembles of axis-aligned classifiers. To take advantage of this insight, it was proposed to rotate the feature space prior to inducing each base learner. From this work, we also learned that the best rotations are those that align the largest quasi-linear decision boundary segment with one of the axes.

In Chapter 2, it was conjectured that only a very small number of rotations actually aligned large segments of the decision boundary with one of the axes and were therefore effective. On the surface, the other rotations appeared counter-productive and a method was proposed to overweight effective rotations and underweight unsuccessful rotations. The results were mixed: On the one hand, we could confidently show that focusing on rotations that produced simpler base learners did not have a negative impact on performance, a fact that in and of itself is useful because it enabled us to produce smaller classifiers without decreasing performance. On the other hand, the performance improvements, while observable, were relatively modest in general.

In the present chapter, we leverage random rotations in two novel ways:

- In Section 3.2, multiple rotations are generated for each classifier. This is a notable departure from earlier chapters, where one or more classifiers were generated on each rotation. In fact, instead of generating a separate random rotation prior to inducing each base learner, we pre-generate all random rotations and then use the entire collection of these rotations as an input when inducing each base learner. The benefit of this approach is that classifiers can theoretically use the best available rotation at every *local* segment of the decision boundary, rather than only emphasizing those rotations that are best aligned in a global sense, as was the case in the procedures described in Chapters 1 and 2.

- In Section 3.3, we return to the standard setting with one or more classifiers per rotation. However, instead of using a global weighting scheme to combine these predictions, the outputs of the individual classifiers are combined using a meta- or stacking classifier. In contract to Section 3.2, it is not the rotations that are combined in a non-linear fashion but the actual predictions on these rotations, again achieving a combined predictions that is *local* in nature. Extra care must be taken in this case to avoid data snooping biases when training multiple levels of predictors. This is solved using a repeated, nested cross-validation.

We show evidence of performance improvements that can be achieved with these local approaches over their global counterparts and in a number of settings. At the same time, we demonstrate that the resulting classifiers are consistently more compact than those created in Chapters 1 and 2. These improvements appear stable across a range of test setups.

3.2 Multi-Rotation Classifiers

In Random Rotation Ensembles, each tree T in the ensemble was represented as follows:

$$T(x;\theta,\Omega) = \sum_{j=1}^{J} c_j I(R(x) \in G_j), \tag{3.1}$$

where each randomly rotated input $R(x)$ was mapped to a constant c_j, depending on which region G_j the input belonged to. J represented the total number of regions or, equivalently, terminal nodes of the tree. Ω represented the deterministic tuning parameters c_j and G_j, whereas θ represented the random parameters used to induce the tree, including the bootstrap sample and the random inputs to the split variable and threshold selection. It is evident that each tree was exactly associated with one rotation and there were multiple such trees in the ensemble.

In the present section, an array of rotations is first created and the input is augmented by its $K-1$ random rotations, which also forces the region G to become higher dimensional. The indicator function I still only tests if the augmented point lies in the extended region, that is

$$T(x;\theta,\Omega) = \sum_{j=1}^{J} c_j I(|x|R_1(x)|R_2(x)|...|R_K(x)| \in G_j). \tag{3.2}$$

The vertical lines in Formula 3.2 indicate that the columns of the original feature space are extended by all of the columns of each of the rotated versions of the feature space. This can also be thought of as a binary operation that merges the columns of two matrices and is then generalized to a n-ary operation. In the software implementation, this is accomplished using a standard *l*eft-join operation.

In the proposed setup, the trees are always induced on an extended set of features that includes the original unrotated features, plus $K-1$ randomly rotated versions of these very same features. Including the unrotated features affords the classifier the theoretical opportunity to never underperform a standard tree ensemble; in practice, the random nature of the induction process makes an unfavorable outcome possible nonetheless.

3.2.1 Intuition

At first glance, it might appear that not much is gained from loading the classifiers with multiple rotated versions of the same original features and thereby effectively increasing the dimensionality of the problem. However, there is a clear intuition behind it, particularly for axis-aligned learners: when multiple rotations of the problem are available at each variable split then the classifier can always select the rotation that orients that particular subproblem best with one of the boundaries. This approach is a lot more powerful than attempting to select one best rotation for the entire tree. It is also expected to produce smaller trees on average.

To illustrate these points, we revisit the introductory example from Chapter 1. Figure 3.1 illustrates that it is possible to achieve very complex oblique decision boundaries from a single tree simply by providing the classifier with

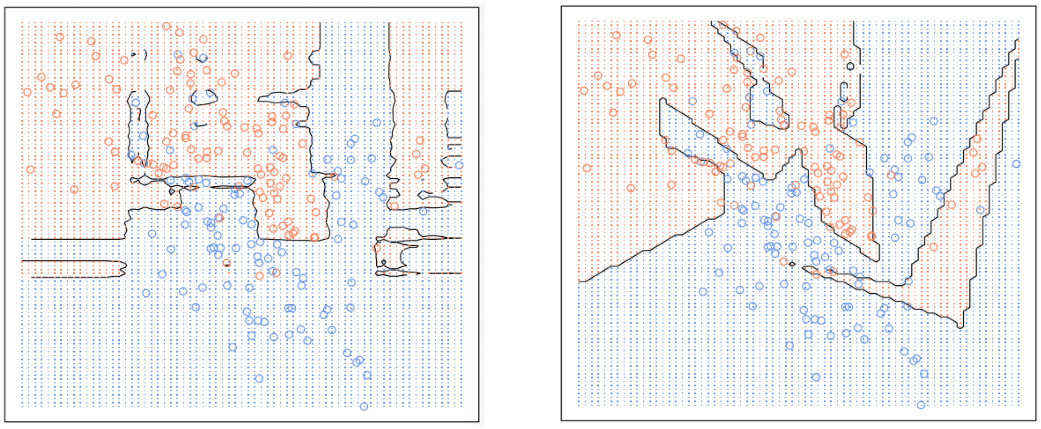

(a) Single tree, no feature rotations (b) Single tree, 20 feature rotations

Fig. 3.1: Comparison of the decision boundary for a single tree with only the original features and no rotations with a single tree built on top of 20 rotations of the original features

additional rotations of the original features. Indeed, the right-hand side of the figure does not have the characteristic shape of a traditional tree-based ensemble, much less of a *single* decision tree. In fact, it looks more like an entire random rotation ensemble. This illustrates the potential of the multi-rotation approach.

One potential downside of the proposed approach is that during the decision tree induction, many additional split variables need to be considered and evaluated at each split decision. This is because if K is the number of rotations passed to the classifier, there are K times as many features available to the classifier, when compared to the setting where each classifier is induced on a single rotation. Depending on how the randomness is controlled during the induction, it could therefore take a maximum of K times as many operations to induce each tree. Of course, this is offset by the fact that the resulting trees tend to be significantly smaller, as we will demonstrate, and that the split decisions are often randomized anyway, making this a rather theoretical upper bound.

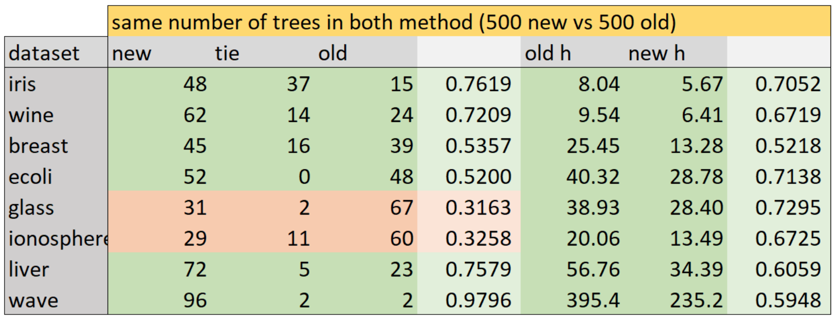
However, in the following section we do not account for these mitigating factors and simply assume that the computational work required to induce a Random Forest with 500 trees is comparable to a multi-rotation ensemble of $500/K$ trees. Surprisingly, the new method is competitive even under these unrealistically unfavorable assumptions.

3.2.2 Evaluation

Our own random.rotation R package described in Chapter 5 was used to compare the performance of a standard random forest with that of a multi-rotation classifier ensemble. As in previous chapters, all variables were scaled to $[0, 1]$ in order to make sure that rotations were not penalized if the scales of the variables were incompatible.

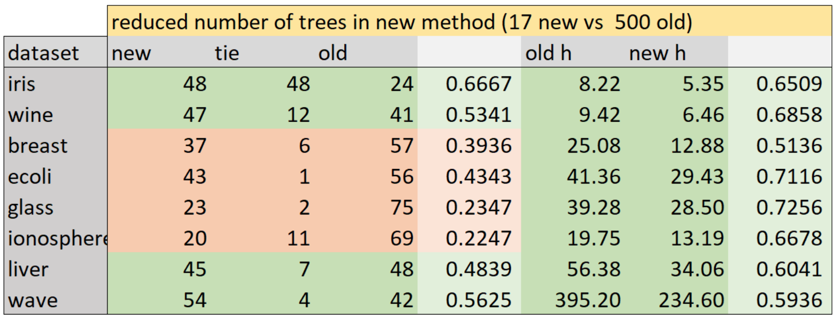
The tests were performed on eight standard UCI data sets: iris, wine, breast, ecoli, glass, ionosphere, liver and wave. In each case, half of the data set was selected at random as the training set and the other half was used as the test set. Initially, 500 trees were built for both methods and later the number of trees for the new method was reduced to $500/K$ in order to compare performance with a similar training effort. The tests were repeated 100 times to minimize the impact of poorly selected random training splits.

The reason the training efforts of the two methods is assumed to be comparable is because if the number of predictors is doubled, this also doubles the number of predictors that need to be considered and hence the number of split points that need to be evaluated. As explained in the previous section, this is a rather conservative assumption.



	same number of trees in both method (500 new vs 500 old)						
dataset	new	tie	old		old h	new h	
iris	48	37	15	0.7619	8.04	5.67	0.7052
wine	62	14	24	0.7209	9.54	6.41	0.6719
breast	45	16	39	0.5357	25.45	13.28	0.5218
ecoli	52	0	48	0.5200	40.32	28.78	0.7138
glass	31	2	67	0.3163	38.93	28.40	0.7295
ionosphere	29	11	60	0.3258	20.06	13.49	0.6725
liver	72	5	23	0.7579	56.76	34.39	0.6059
wave	96	2	2	0.9796	395.4	235.2	0.5948

(a) **500 vs 500 trees**



	reduced number of trees in new method (17 new vs 500 old)						
dataset	new	tie	old		old h	new h	
iris	48	48	24	0.6667	8.22	5.35	0.6509
wine	47	12	41	0.5341	9.42	6.46	0.6858
breast	37	6	57	0.3936	25.08	12.88	0.5136
ecoli	43	1	56	0.4343	41.36	29.43	0.7116
glass	23	2	75	0.2347	39.28	28.50	0.7256
ionosphere	20	11	69	0.2247	19.75	13.19	0.6678
liver	45	7	48	0.4839	56.38	34.06	0.6041
wave	54	4	42	0.5625	395.20	234.60	0.5936

(b) **17 vs 500 trees**

Tab. 3.1: The table compares the multi-rotation classifier with the unrotated classifier. In the top figure, the multi-rotation classifier contains the same number of trees as the unrotated classifier. In the bottom figure, the number of trees was reduced proportionally to the number of number of input features. Column heading 'new' is the number of times (out of 100) where the new method outperformed, 'tie' indicates the number of times they performed the same, while 'old' represents the number of times a plain random forest outperformed. Column header 'old h' shows the height of the median tree in the plain random forest, while 'new h' - the height of the median tree for the multi-rotation classifier.

We then tabulated the number of times the new method performed better than the standard random forest, the number of ties, and the number of times the random forest outperformed the new method. In addition, we kept track of the median tree size, here defined in terms of the height of the individual trees.

The results are interesting: in both cases, the multi-rotation classifier with multiple rotations per base learner considerably outperformed the existing method, while consistently generating smaller trees. The full results are listed in part (a) of Table 3.1

In the iris example, there were 48 cases where the new method outperformed, 37 cases where there was a tie and only 15 cases where the standard version outperformed. This means when there was no tie, the new method outperformed in 76.19 percent of cases. At the same time, the tree size was 5.67 on average, 30 percent lower than the average tree size of the standard method.

In the wine example, there were 62 cases where the new method outperformed, 14 cases where there was a tie and only 24 cases where the standard version outperformed. This means when there was no tie, the new method outperformed in 72 percent of cases. At the same time, the tree size was 6.41 on average for the new method and 9.54 for the standard method. In other words, the trees were smaller by one third in the new method on average. Equally interesting is the fact that the largest tree in the new method was smaller than even the first quartile of the standard method.

Only in one quarter of the test data sets did the unrotated version outperform the version based on multi-rotation classifiers. This occurred for data sets glass and ionosphere. These occasional negative results reinforce the point that random rotations should be offered as an optional add-on for tree-based methods, rather than as the only available option.

In the second test, the multi-rotation classifier ensembles were severely penalized by restricting the ensemble size to only $\lceil 500/30 \rceil = 17$ trees. This is a harsh penalty for three reasons: (1) we know from Chapters 1 and 2 that many rotations are not helpful for much of the decision boundary, (2) tree-sizes for multi-rotation classifiers were just shown to be significantly smaller than their unrotated counterparts, and (3) from experience, it is unreasonable to assume we can achieve full convergence with only 17 trees.

Despite these challenging conditions, the new method held up reasonably well in the comparison, still outperforming in half of the data sets and producing trees that were smaller on average in every problem instance. This is illustrated in part (b) of Figure 3.1. It is notable that the unrotated version with thirty times more trees and larger trees on average did not outperform in these tests.

3.2.3 Discussion

By extending the feature space via linear rotations and thereby taking the problem to a higher dimension, the effect of multi-rotation appears to be comparable to that of a kernel. It enables the classifier to find decision boundaries that are oblique to the original coordinate system and specialized to each local sub-part of the problem. On average, the biggest benefit in the form of a greater degree of freedom comes from the first rotation and the benefit slowly tapers off as additional rotations are added. Naturally, the exact marginal behavior depends on the natural fit of the generated random rotation to the problem at hand. The problem of finding the optimal rotations for a given data set is not tractable in general and random rotations are a convenient and useful mechanism to quickly converge to a set of transformations that can be efficiently leveraged by a classifier.

Just like for standard tree ensembles, the resulting classifiers exhibit a low bias because they are based on unpruned trees. However, the presented multi-rotation classifiers accomplishes this goal with fewer and much more compact trees on average.

It is also important to note that unlike for the methods described in Chapter 2, there is no need to select the best set of rotations *a priori* for multi-rotation classifiers. Instead, the induction algorithm of the classifier automatically selects the most suitable rotations at each decision point, for example when deciding on a variable split in a decision tree. Incorporating this decision into the classifier also extends the usefulness of the built-in importance metrics, making it possible to not only assess the importance of the individual predictors but also their rotations *ex post*. In this way, the classifier can optionally be retrained with only the rotations that were deemed most helpful.

Finally, because all rotations are pre-generated and passed in as a collection, the method is equally suitable to classifiers that have an inter-tree dependency, such as gradient boosting approaches.

3.3 Stacking Classifiers

In this section we discuss a second method for taking advantage of multiple random rotations: model stacking. The idea behind this approach is to generate a classifier on each random rotation, as we have done in the earlier chapters. However, instead of applying a global parametric weighting function, we use the predictions of each of these base (or Level-0) classifiers as an input into a newly introduced meta (or Level-1) classifier. This meta classifier determines for each region of the decision boundary, which rotations - or more precisely: which combination of rotated base classifiers - is best suited locally.

Let $Lx.y$ denote an index into the yth base classifier of a Level-x ensemble classifier. In this section, we restrict x and y as follows: $x \in \{0, 1\}$ and $y \in \{0, 1, \ldots, K\}$. Later, we will briefly touch on the idea of a classifier with $x > 1$, that is, a multi-stage classifier. In the following, if we omit y then we refer to the entire Level-x ensemble classifier, including all of its base classifiers. With this notation, a tree-based meta (Level-1) ensemble classifier T^{L1} can be expressed as follows in terms of the predictions of the individual tree-based base (Level-0) classifiers $T^{L0.0} ... T^{L0.K}$:

$$\begin{aligned} T^{L1}(x;\theta,\Omega) = \sum_{j=1}^{J} c_j I(\quad & |T^{L0.0}(x;\theta^{L0.0},\Omega^{L0.0}) \\ & |T^{L0.1}(R_1(x);\theta^{L0.1},\Omega^{L0.1}) \\ & |T^{L0.2}(R_2(x);\theta^{L0.2},\Omega^{L0.2}) \\ & |\ldots \\ & |T^{L0.K}(R_K(x);\theta^{L0.K},\Omega^{L0.K})| \\ & \in G_j). \end{aligned} \tag{3.3}$$

As in the previous section, we use the vertical lines to denote a column-based augmentation of the inputs.

At first sight, this Formula 3.3 looks very different from Formula 3.2 in the previous section. However, the two are actually closely related: instead of aggregating the raw, rotated predictors into a single input, the new formulation aggregates the predictions from the Level-0 classifiers, which are themselves trees. Also note that each Level-0 tree includes its own set of random and deterministic parameters θ and Ω, which is also the reason each tree specification uses a matching superscript $L0.0$ to $L0.K$ for these parameter sets.

The main challenge with a two-stage approach is to ensure that the Level-0 base classifiers are not trained on the same data points as the Level-1 stacking classifier. If we do not control for this, the Level-1 classifier ends up relying on the in-sample performance of the Level-0 classifiers. This predictably leads to a selection of the Level-0 classifiers with the best in-sample performance, rather than a true meta-classifier that takes advantage of local properties of the feature space and base predictions.

One way to accomplish this data separation is to randomly divide the data into three disjoint partitions, use the first partition to train the Level-0 classifiers, the second partition to train the Level-1 classifier and the third partition for out-of-sample validation of the Level-1 stacking classifier. However, this is a very inefficient use of data: the base classifiers are only exposed to one third of the full data set during training. To make matters worse, for data sets that are imbalanced, such a naive approach could mean that a randomly generated training partition does not contain any (or very few) instances of one of the class labels, especially in the absence of a carefully-implemented stratification strategy. A better approach is to use data folds and cross-validation. In this procedure, each data partition becomes the out-of-sample partition in one instance, leading to a much better utilization of the available data. In addition, by using more than three folds, the percentage of training data available to the base classifiers can be controlled with precision and

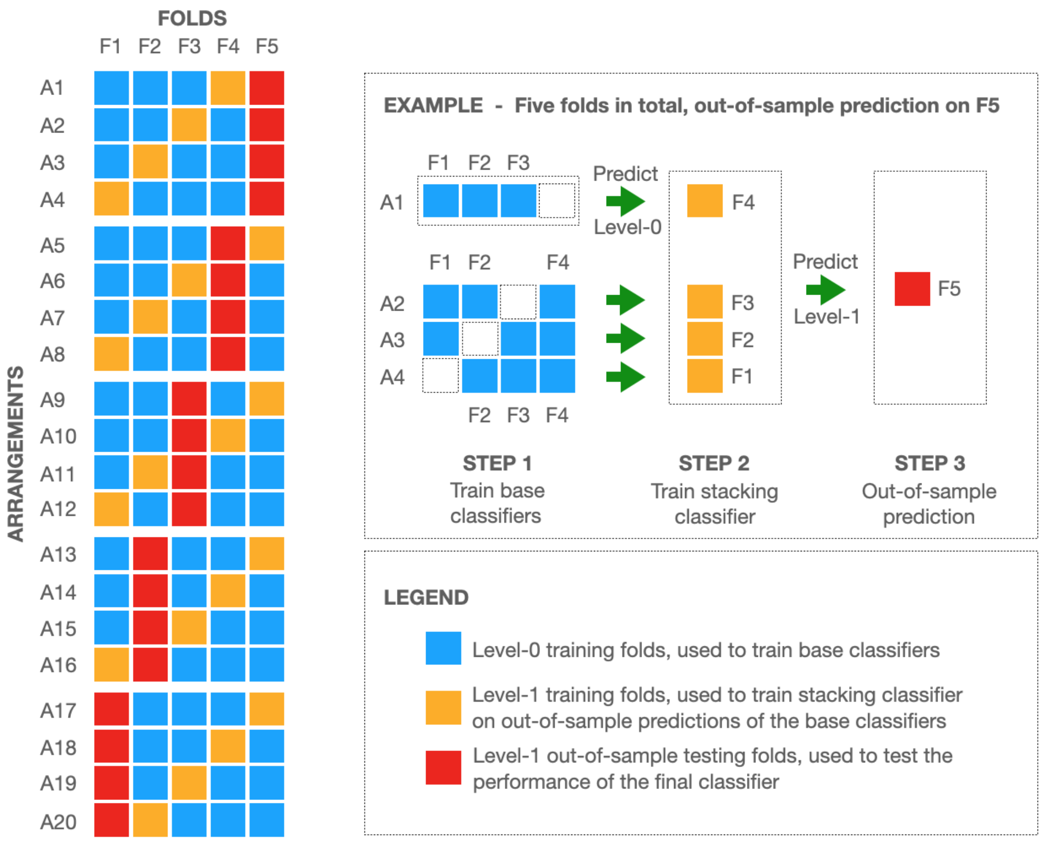

FOLDS
F1 F2 F3 F4 F5
ARRANGEMENTS
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 A16 A17 A18 A19 A20
EXAMPLE - Five folds in total, out-of-sample prediction on F5
Predict
Level-0
Predict
Level-1
STEP 1
Train base classifiers
STEP 2
Train stacking classifier
STEP 3
Out-of-sample prediction
LEGEND
Level-0 training folds, used to train base classifiers
Level-1 training folds, used to train stacking classifier on out-of-sample predictions of the base classifiers
Level-1 out-of-sample testing folds, used to test the performance of the final classifier

Fig. 3.2: High-level overview of the nested cross validation procedure required to properly tune and evaluate a stacking predictor. The depicted case relies on five cross validation folds F1-F5.

carefully traded off against the the cost of the corresponding computational overhead. In the next section, we will discuss this approach in detail.

3.3.1 Repeated Nested Cross-Validation

Traditional cross-validation divides the data into F Folds, then uses $F-1$ of these folds to create a model, and finally leverages the remaining (holdout) fold to test the model out of sample. This procedure results in a good single-step model. However, it suffers from the same data snooping issues described above in the case of a two-stage or multi-stage stacking model. For this reason, a nested cross-validation is necessary in this case to address these challenges.

Nested, in this context, means that we use $F-1$ folds to create a Level-1 stacking model but to create the Level-0 base predictors and any potential tuning of these models we only use $F-2$ folds, essentially performing a cross-validation within a cross validation. Figure 3.2 shows an illustration of this approach in the case where the data is divided into 5 folds.

In the general case, we have $F-2$ folds available to train the base classifiers. There are exactly $C(F,F-2) = C(F,2) = F(F-1)/2$ such arrangements. For each of these, we need to train a Level-0 base classifier and the out-of-sample predictions can be made on either of the two holdout folds. For this reason, the total number of possible fold arrangements is $F(F-1)$. In Figure 3.2, $F = 5$ and, therefore, the number of possible arrangements is $5 \times 4 = 20$.

It is important to note that in a first step, groups of $F-1$ arrangements are used to make out-of-sample predictions (orange) on the entire data set except for the holdout fold (red). All of these out-of-sample predictions are then combined to train the Level-1 stacking predictor and that, in turn, is then used to make an out-of-sample prediction on the holdout fold.

In general, $F-2$ folds (or in percentage terms $(F-2)/F$ percent of the data) are used to train the Level-0 classifiers and $F-1$ folds (or in percentage terms $(F-1)/F$ percent of the data) are used to train the Level-1 classifiers. With the number of folds, this percentage increases gradually. For example, with 3 folds only $(3-2)/3 = 1/3$ of the data is used to train the Level-0 classifiers and $(3-1)/3 = 2/3$ of the data to train the Level-1 classifiers. With 5 folds, these numbers already increase to $(5-2)/5 = 3/5$ and $(5-1)/5 = 4/5$ respectively. The two upward sloping green curves in Figure 3.3 provide an illustration of the data utilization for the Level-0 and Level-1 classifiers,

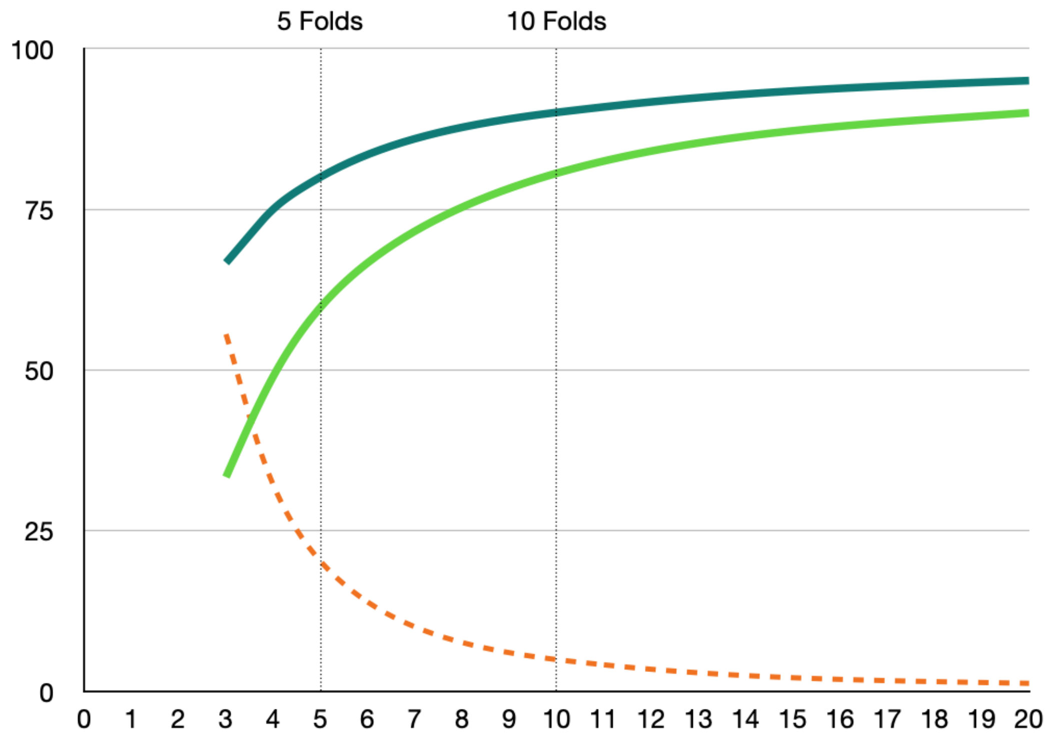


Fig. 3.3: The number of required function evaluations of each predictor increases with the square of the number of folds used. The declining dashed orange line shows the number of repeats that can theoretically be performed if the number of evaluations are to be kept constant at 500, while the number of folds is changed. The light green increasing line shows the percentage of the data that is used to construct each level-0 predictor as a function of the number of folds, while the dark green line shows the percentage of the data that is used to construct the level-1 (stacking) predictor.

respectively. If our goal is to use more than 50% of the data to train the base classifiers, we need to use at least 5 folds.

According to this analysis, it would be preferable to maximize the number of folds for optimal data utilization. However, there is a second effect that needs to considered: computational costs. In the general case, we need to build $F(F-1)$ versions of the base models plus F versions of the stacking model in order to perform an out-of-sample evaluation on the full data. In other words, we need to generate F^2 predictive models in total. This means the computational costs increase by the square of the number of folds.

To make matters worse, the entire process needs to be repeated multiple times in order to account for the variance introduced by partitioning the data at random. For example, some random partitions may produce better classifiers simply because the different classes are better distributed across the partitions - purely by chance. In order to isolate this effect, the experiment needs to be repeated. The downward sloping dashed line in Figure 3.3 shows how many times we can repeat the experiment with a given number of folds at a constant number of total classifiers that need to be built for a full model evaluation. The line drops off rather sharply from 20 possible repeats with 5 folds to merely 5 repeats with 10 folds with the same computational effort. Hence, if our goal is to repeat the experiment at least 5 times, we cannot use more than 10 folds.

From these calculations, it becomes clear that a sensible number of folds in terms of a reasonable trade-off between data utilization and computational effort is in the range of $5 \leq F \leq 10$. In the following section, we will use 5 folds with 20 repeats, emphasizing additional repeats over higher data utilization. This allows us to make a more accurate evaluation of the models and corresponds exactly to the setup in Figure 3.2.

Measuring the variance of the procedure across the different randomly generated fold groups is valuable but even more important, in the context of random rotations, is to determine the impact of a specific set of random rotations: what happens if we replace a given set of random rotations with a different set of random rotations? To answer this question, the experiment needs to be repeated with multiple sets of random rotations. However, we would like to do so with the same fold groups in each case, in order to cleanly delineate the variance resulting from the different fold assignments from that

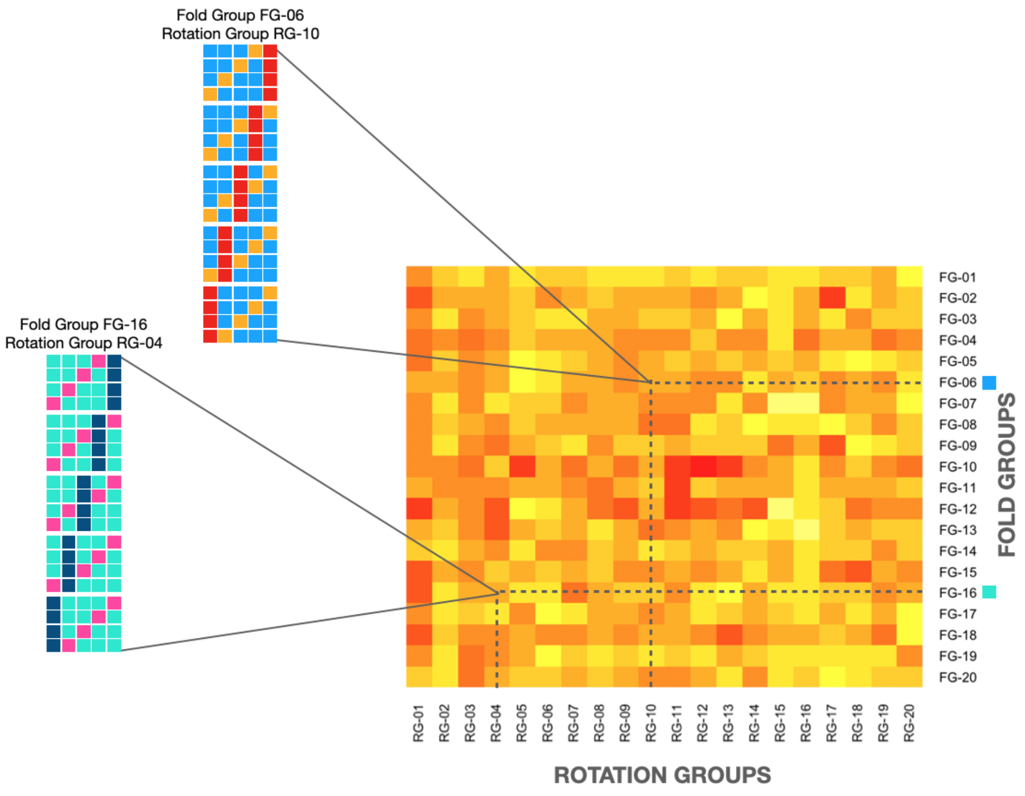


Fig. 3.4: The testing procedure uses 20 different fold groups and 20 different rotation groups. Each fold groups represents a random partition of the data into five disjoint folds. Each rotation group represents a set of multiple random rotations. Each row of the matrix uses the same fold group but different rotation groups and each column of the matrix uses the same rotation group but different fold groups. Darker squares in the heatmap correspond to higher out-of-sample errors (worse performance), lighter squares correspond to lower out-of-sample errors (better performance).

caused by the use of different rotations. A general procedure to accomplish this is presented in the next section.

3.3.2 A general evaluation procedure

We now describe a general evaluation procedure for random rotations, illustrated by a specific experiment.

In a first step, N groups of random rotations and M groups of folds are generated. Each of the N groups of random rotations contains exactly K rotations, while each of the M fold groups contains exactly F disjoint folds, whose union is always the entire data set. The pre-generation of these rotation- and fold-groups makes it possible to evaluate a given classifier on all $N \times M$ combinations, allowing us to perform an analysis of variance on the result matrix. For each entry in the test matrix, a nested cross-validation of the classifier is performed.

In Figure 3.4 the test setup is illustrated for the case of $N = M = K = 20$ and $F = 5$, the parameters we will continue to use for the remainder of this chapter. In this case, we have generated 20 different sets of rotations and 20 different fold groups for a total of $20 \times 20 = 400$ entries in the test matrix. We then applied a nested cross-validation on each of the 400 rotation group / fold group combinations. Since the nested cross-validation is repeated 400 times in total, it is a repeated nested cross-validation. The out-of-sample test error of each entry in the test matrix is illustrated in the form of a heat map, where lighter squares indicate smaller test errors (better results), while darker squares indicate greater test errors (worse results). Two examples of the underlying cross-validations are shown on the left-hand side of the figure; each pixel of the heat map corresponds to a separate cross-validation procedure. It should also be noted that each row of the heat map uses the same fold group (but different rotation groups), while each column of the heat map uses the same rotation group (but different fold groups).

Once the repeated nested cross-validation is complete, the result matrix can be analyzed: If the variance of the out-of-sample test errors is small across rotation groups then this means the method is not particularly sensitive to the choice of rotation. Furthermore, if the variance across rotation groups is small even compared to the variance across fold groups then the rotations do not add significant value to the classifier. In this specific case, the local methods described in this chapter do not provide a great advantage over the global methods described in earlier chapters. On the other hand, if the variance across rotation groups is high, this implies that a judicious selection of the rotation is helpful for that specific problem. And finally, if the variance across fold groups is very high then the classifier is unstable or the data set is too small to properly evaluate the classifier.

One particularly useful metric for comparing different classifiers is to infer a histogram of the test errors from the results matrix, across the different rotation- and fold-groups. An example of this process is shown in Figure 3.5, again with $N = M = K = 20$ and $F = 5$. Particular care must be taken to scale the heatmaps and histograms properly when comparing two competing classifiers.

3.3.3 Real-world examples of the evaluation procedure

We compared a stacked random rotation ensemble with a random forest base learner (S-RRE/RF) with a standard random forest (RF), a regular random rotation ensemble (RRE), a stacked random rotation ensemble with a XGBoost base learner (S-RRE/XGB) and a plain XGBoost model (XGB).

It should be noted, however, that the goal of this chapter was not to perform a comprehensive comparison of these methods but rather to highlight on some salient examples how stacking ensembles can be rigorously tested on real data sets using the analysis described in the previous section. The easiest way to accomplish this is by re-examining three data sets from Table 3.1 in Section 3.2. There, we found that rotations were somewhat helpful for the iris data set, not helpful for the glass data set and very helpful for the wave data set. We now revisit these examples using the newly developed tools.

Figure 3.6 shows the complete results of the procedure for the stacked random rotation classifier (S-RRE). On the right-hand side of the figure, the names of the fold groups (FG-01 to FG-20) are presented, along with an average error in this fold group across all rotations. On the bottom side, the names of the rotation groups (RG-01 to RG-20) are presented, along with an average error in this rotation group across all fold groups. These figures allow us to sort the heat map by rotation group and fold group from lowest to highest average error, providing us with an indication how much impact the various rotation groups have on the outcome and how much variance is due to the random data split in the various fold groups. These sorts are

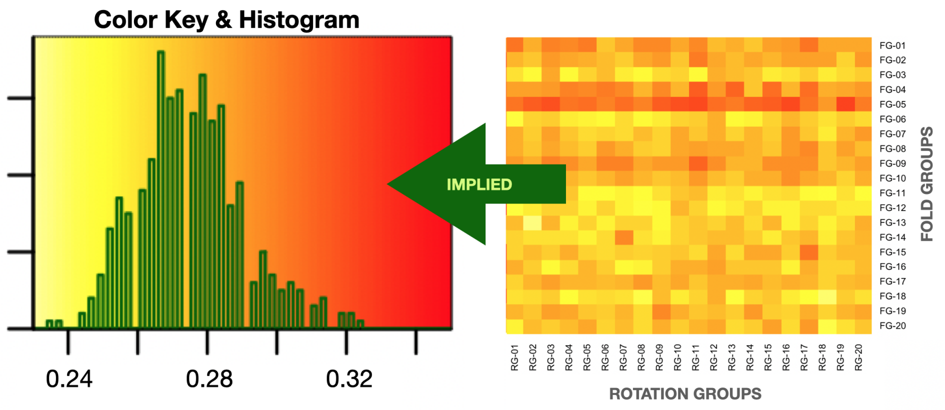


Fig. 3.5: A total of 400 different out-of-sample tests are performed for each predictor: one for each combination of the 20 fold groups and the 20 rotation groups. In addition to evaluating the performance of the predictor on the different rotation groups, the procedure also implies a histogram of the out-of-sample prediction errors. In this example, only very few tests resulted in an error smaller than 0.24 (the very lightest squares in the heatmap and the very left bars of the histogram) and only a few test resulted in an error greater than 0.32 (the very darkest squares in the heatmap and the very right bars of the histogram). This implied histogram can be subsequently be used to directly compare the performance of different predictors on the same data set.

provided in Figures 3.7 and 3.8, respectively. Naturally, the histograms do not change from simply sorting the heat maps.

One interesting outcome from looking at Figure 3.7 is that we can actually use this procedure as a method to find good rotations. For example, rotation group RG-16 resulted in significantly lower out-of-sample errors than RG-01, independently of the fold group (0.0307 vs 0.0487). This illustrates that rotations are useful for this data set and that we have a clear path to find good rotations. The process could be extended by creating a classifier that, for example, only includes the top 5 rotation groups (100 rotations in total).

Next, we look at the same dataset, the same rotation groups and the same fold groups but this time in the context of a plain random forest. This configuration is depicted in Figure 3.9. Rotation groups should not have an impact on the performance of a plain random forest because no rotations

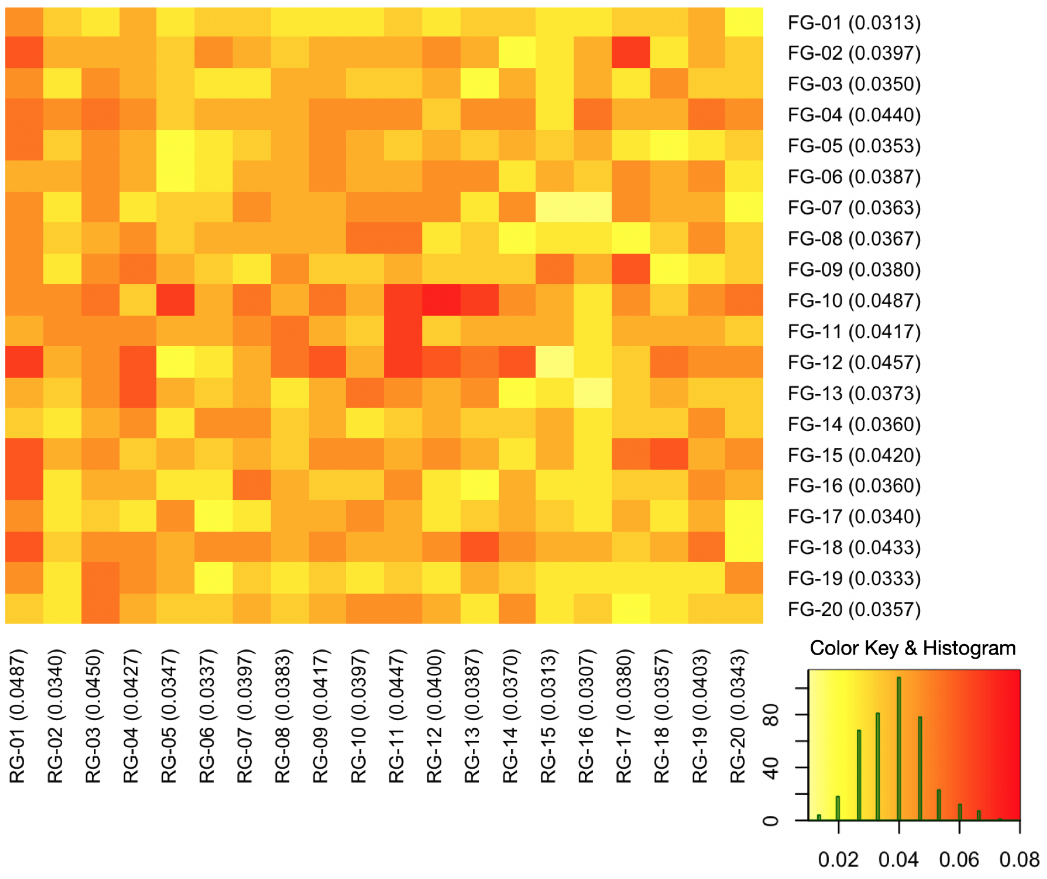


Fig. 3.6: UCI iris dataset, S-RRE

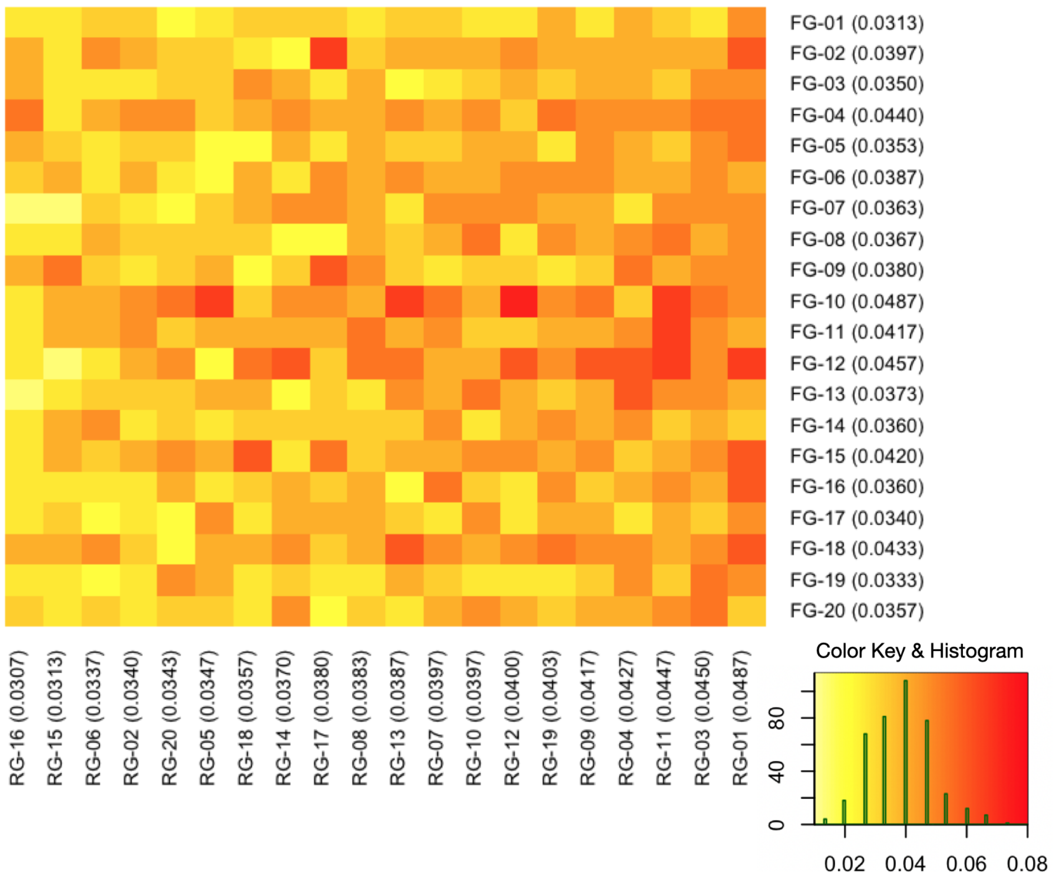


Fig. 3.7: UCI iris dataset, S-RRE, sorted from best to worst rotation group

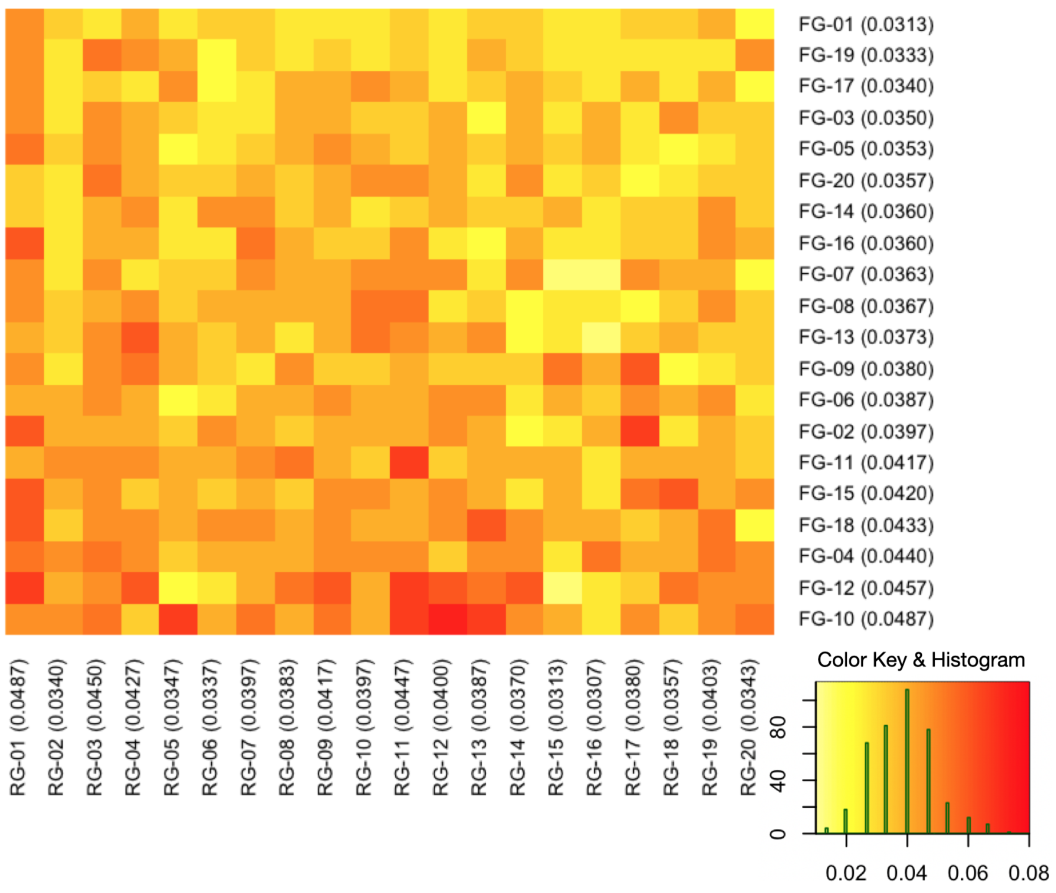


Fig. 3.8: UCI iris dataset, S-RRE, sorted from best to worst fold group

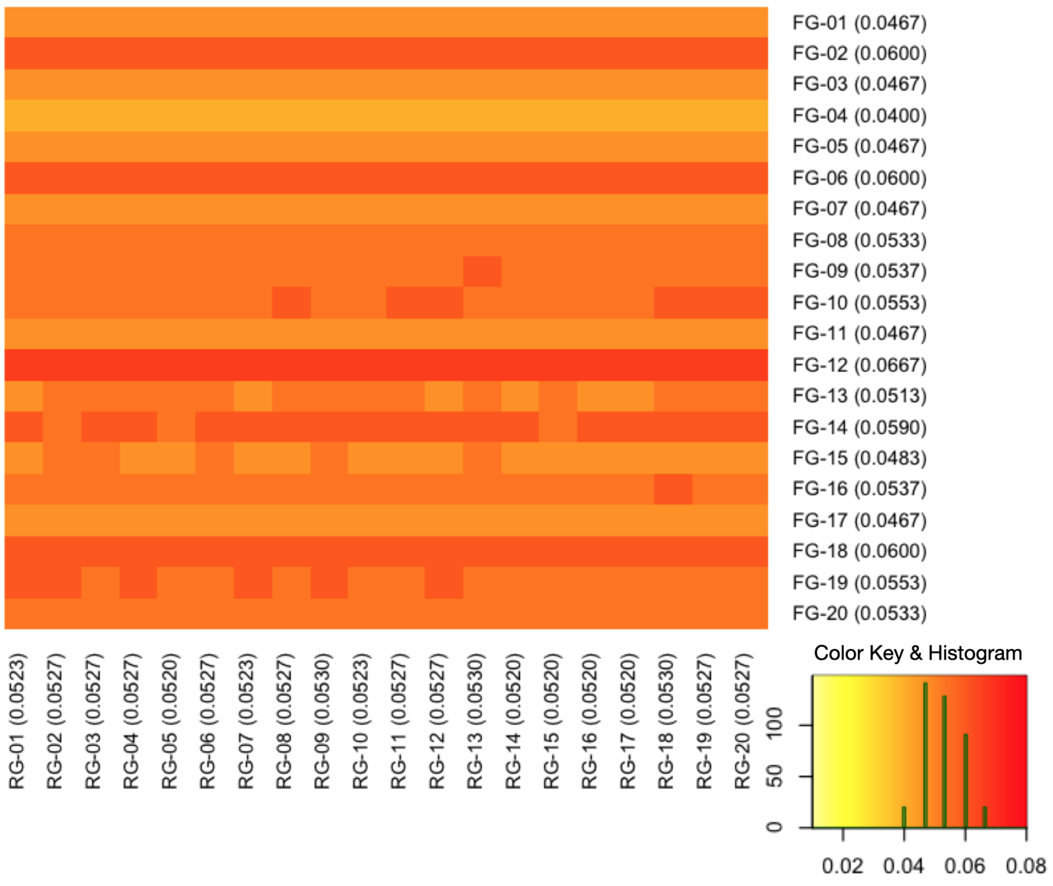


Fig. 3.9: UCI iris dataset, RF

are involved. For this reason, the figure shows more or less horizontal lines of equal color (and error figures). The minor visible artifacts stem from the fact that there is also randomness in the creation of these random forests, for example in the form of random feature selections at each tree branch. However, it is clear from the Figure that across the entire ensemble the impact of this internal randomization is significantly lower than the impact of the random data partitions on the overall out-of-sample error.

One important point to repeat is that we should always keep the scale of the Figures identical across methods to ensure that the color gradients of the different heat maps match exactly across methods. This allows us to then compare the methods directly, as we have done in Figure 3.10. In this case, both figures are sorted from best (top) to worst (bottom) fold groups.

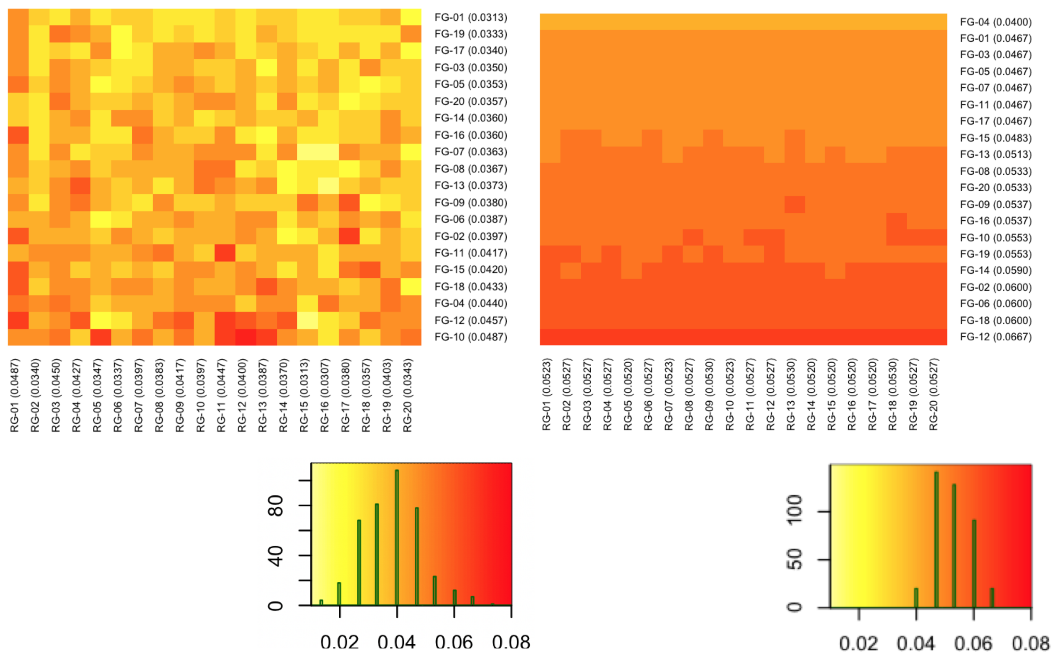


Fig. 3.10: UCI iris dataset, S-RRE (left) vs RF (right), both sorted from best to worst fold group

Just looking at the colors it is clear that S-RRE significantly outperforms RF across a wide range of parameters for this data set. The difference becomes even more obvious when we examine the resulting histograms. Most of the bars for the S-RRE classifier are in the yellow to orange section of the spectrum, whereas the RF classifier has most of the density in the orange and red section. Of course, this also corresponds well with the computed means on these data sets.

A comparison between the S-RRE classifier and plain XG Boost in Figure 3.11 shows a similar result. Here it is interesting to see that we do not observe any artifacts in this figure for the XGB classifier. This has to do with the specific parameters used and the more stable implementation of the boosting algorithm deployed.

We now move on to the glass dataset, which has previously provided us with some challenges for rotations. Figure 3.12 compares the S-RRE classifier

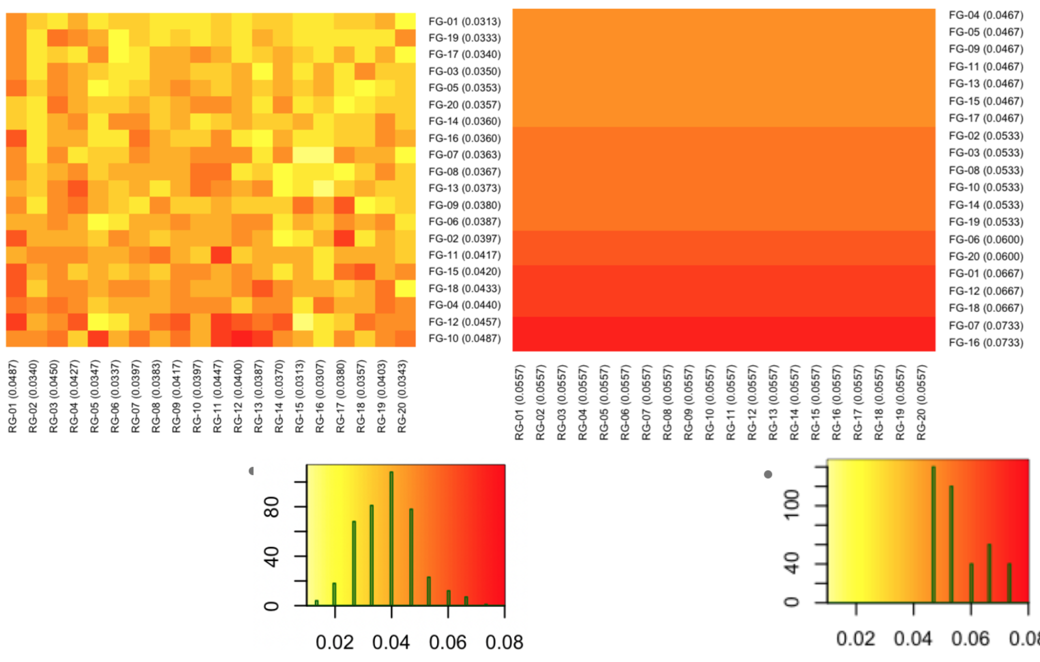


Fig. 3.11: UCI iris dataset, S-RRE (left) vs XGBoost (right), both sorted from best to worst fold group

with a plain random forest. In this case, the random forest significantly outperforms our new stacking classifier. Not only that but we cannot find very large differences across the rotation groups and there is no rotation group that outperforms the plain random forest. It thus becomes obvious that rotations are not an effective way to improve predictive performance for this data set.

For comparison, we also provide a run using plain XG Boost in Figure 3.13. Performance of this classifier is again independent of the rotation group and we see that the results are somewhere in between S-RRE and RF.

For the wave dataset we only show the final histograms of each method in Figure 3.14. In every instance, the rotation-based classifiers outperformed the unrotated classifiers, regardless if the underlying base learner was based on a random forest or an XGBoost classifier.

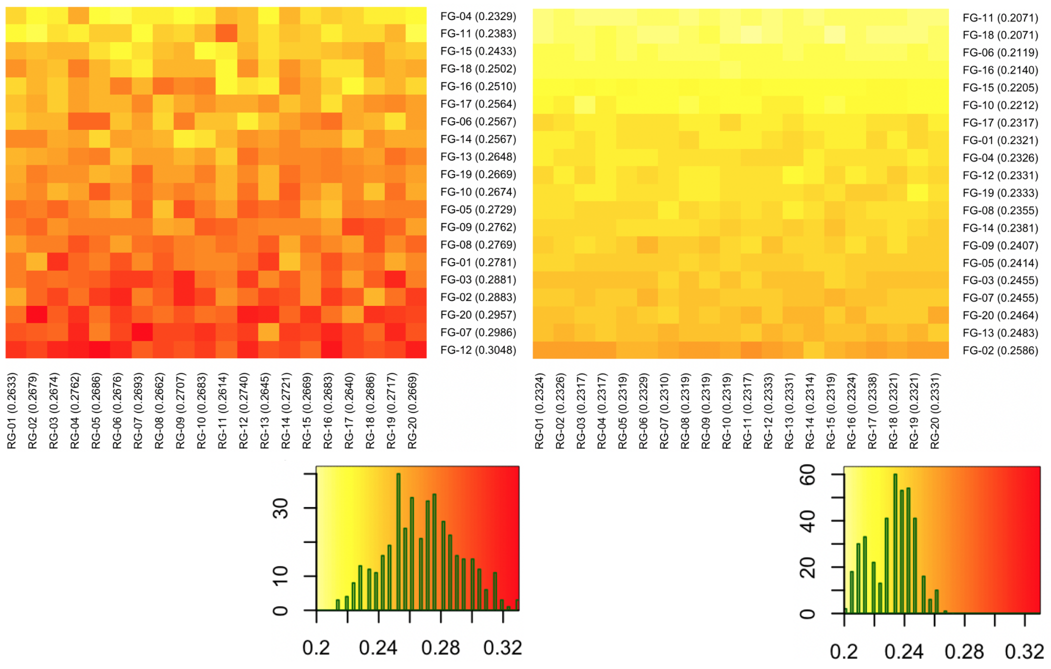


Fig. 3.12: UCI glass dataset, S-RRE (left) vs RF (right), both sorted from best to worst fold group

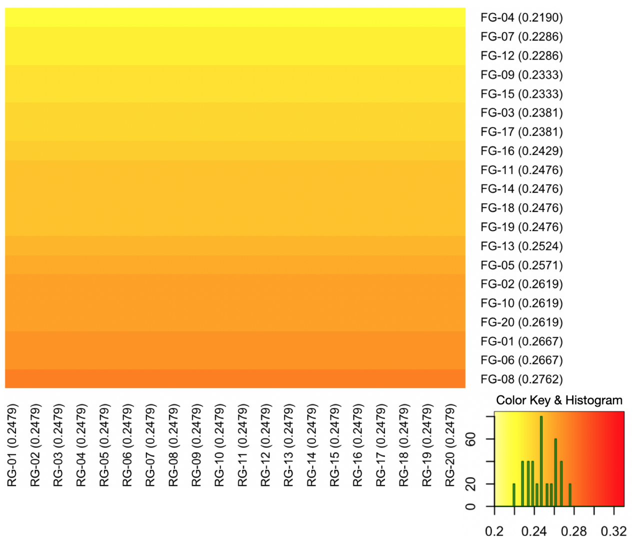


Fig. 3.13: UCI glass dataset, XGB, sorted from best to worst fold group

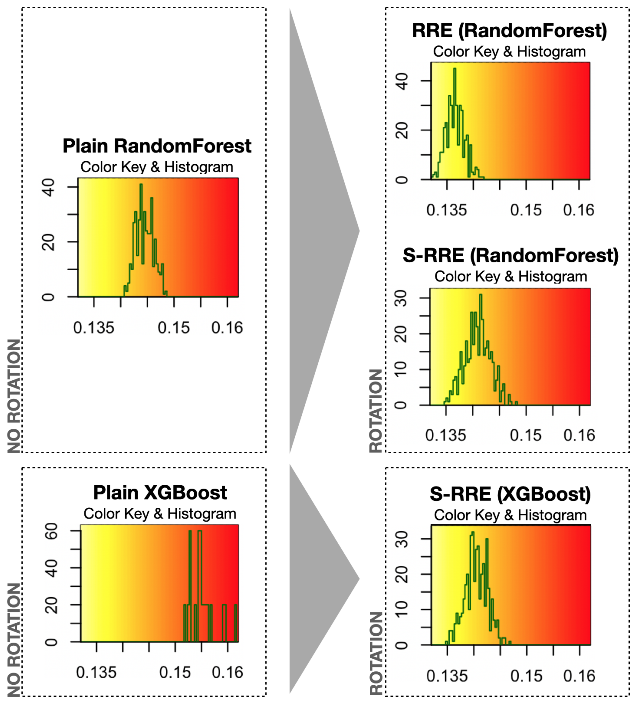


Fig. 3.14: UCI wave dataset, comparison of histograms of 5 different methods. The classifiers on the left-hand side do not use random rotations, while the classifiers on the right-hand side all use random rotations. Adding random rotations appears very beneficial for this problem instance and the selected base learners.

3.4 Observations and future research

In the present chapter we provided an overview of some previously unexplored and unpublished areas of random rotations. We described a general technique based on a repeated, nested cross-validation for determining the efficacy of random rotations for a particular problem instance, along with a clear path to select groups of helpful rotations.

We now discuss some high-level observations, connections, extensions and experimental musings that may lead to future research.

3.4.1 Grid searches

The first observation is that the repeated, nested cross-validation is not only useful to create meta-predictors that do not exhibit a data snooping bias but can simultaneously be used to perform grid searches of the hyper-parameters of the base classifiers without introducing data leakage.

Referring back to Figure 3.2, when the Level-0 training folds (blue) are used to predict the Level-1 training fold (orange), it is helpful to first perform a grid search in which out-of-bag data is leveraged to determine a good parameter set. This process can be used to predict each of the Level-1 training folds and again during the training of the Level-1 classifier.

Obviously, this process is computationally more demanding but produces better predictors. Perhaps less obviously, it also leads to base classifiers that use different tuning parameter settings. However, this is not an issue if we take a modeling pipeline view of the process: the parameter search is simply part of the process and is repeated in the same way for each fold and again to produce the final model. In a way, the search is part of our model and consistently used across the pipeline.

3.4.2 Multi-stage classifiers

The second observation is that the repeated, nested cross-validation can easily be extended to more than two stages: in each stage we perform out-of-sample predictions that are used as inputs for the next stage. Such an architecture starts to resemble that of a deep neural network, especially block-based and modular neural networks. There are two main differences between these approaches: (a) neural networks are typically trained via backpropagation, whereas such a sequence of trees is trained in a forward-only fashion and (b) the initial layers of neural networks are often used to transform and restructure the input data and only later layers assemble the actual prediction, whereas a sequence of decision trees does not exhibit such a specialization.

3.4.3 Combining linear with nonlinear transforms

The third observation is also related to neural networks: it is very interesting to note that neural networks apply a nonlinear activation function with a linear weighting scheme (and do so repeatedly, layer for layer), whereas random rotation ensembles combine nonlinear decision trees with a linear rotation operation. The interplay between these two operations is essential for good performance in both approaches.

In fact, it is a very common theme in statistics and machine learning to resort to carefully chosen combinations of linear and nonlinear approaches. For example, even in the case of linear regression, it is now common to include nonlinear penalties to an otherwise linear model (e.g. Tikhonov / Ridge regularization, Lasso regularization, Elastic Net, etc). Random rotation adds a linear perturbance to an otherwise nonlinear (but axis-aligned) model.

The interplay between linear and nonlinear approaches, as well as the similarities between tree-based approaches and neural networks would make for interesting future research areas.

4 Impact and Discussion

Artificial intelligence is the science of making machines do things that would require intelligence if done by men.

— **Marvin Minsky**
(Computer scientist and AI researcher)

The core material of Chapter 1 was first introduced as a working paper in 2015 and eventually led to the publication of Blaser and Fryzlewicz (2016).

Since then, a full five years have passed and the pace of new research has been nothing short of astonishing.

During this time, our paper was cited numerous times. At the same time, at least five different publicly available software implementations of random rotation ensembles - including a module for the highly popular scikit-learn machine learning library - have been created by talented third party developers.

In this chapter, we will review the impact these original ideas on random rotations have had on the literature, examine the main themes of recent research and provide an outlook for future research.

At a high level, there are five main themes that can be gleaned from scientific literature in relation to random rotations and we will examine each of them in turn.

Probably the lowest hanging fruit are the numerous applications of random rotations in various fields from neuroscience and heart medicine to naval propulsion systems and intrusion detection. It is rewarding to see that random rotations have had a positive impact on so many different fields.

There are also papers that directly use the ideas and code from the paper to generate uniformly random rotations in other machine learning contexts, such as clustering or even robotic motion planning, rather than strictly as part of a random rotation ensemble. Some of these applications are quite ingenious.

The second category are direct extensions of random rotation ensembles, for example making them applicable to boosting trees, adding additional sources of randomness or using a structured approach in an attempt to find the most effective rotations. This last topic is also the subject of our second paper in this series, which is covered in Chapter 2 of this thesis. The recently emerging field of privacy-preserving approaches is somewhat orthogonal to random rotations but can still be viewed or at least implemented as an extension and we therefore optimistically subsume them in this category as well.

The third category are variations on oblique trees and forests thereof. Random rotations provide a natural way of generating decision boundaries that are not axis-aligned but there are a number of other ways to accomplish this and the authors of these papers noted the connection.

The fourth category are ensemble aggregation and weighting schemes designed to increase the diversity of the ensemble and the quality of the prediction.

And finally, there are a number of approaches and papers based on random projections in machine learning. Random projections can be viewed as a random rotation followed by an axis-aligned projection. For this reason, the two approaches are actually very closely related. Random rotations cleanly separate the alignment of the feature space from the actual classifier, whereas random projections serve as a computationally efficient and theoretically sound dimension reduction technique prior to applying the actual classifier.

In addition, there are also summary articles that mention random rotations, sometimes among many other options, and we will mention a few of these in this chapter too.

From a qualitative standpoint it can be said that some of the citing papers demonstrated an explicit theoretical or empirical benefit in using random rotations. None of the papers cited the approach as a negative example or bad performer and the worst reported outcome was that no statistically significant improvement was observed on the data set in question. In the best case, statistically significant improvements were demonstrated. In summary, it is likely beneficial to users of axis-aligned classifiers, such as tree-based models, to at least be provided with the option of randomly rotating the feature space.

We now examine each of these categories in more detail.

4.1 Applications of random rotations

There are a number of papers that were inspired by random rotations to improve upon a particular, often industry-specific application of machine learning. In some cases, the paper directly leads to better classification, while in other cases random rotations are simply listed as additional, optional enhancements or variations.

Desai et al. (2016) use ensemble classifiers to detect fatal cardiac abnormalities and leverage elements of our approach to randomize one of their model parameters. A different medical application is discussed in Paramanik et al. (2021), where decision forests are used for Parkinson’s detection. In this case, random rotations are only mentioned as a variation of a random forest.

Cipollini et al. (2018b) and Cipollini et al. (2018a) successfully apply supervised learning to naval propulsion systems and describe random rotation ensembles as a "popular state-of-the-art and widely adapted method" and note that the approach can be adapted to both regression and classification problems. The method is also used in Oneto, Cordaddu, et al. (2017), where the crash stop maneuvering performance of ships – one of the key indicators of vessel safety – is predicted in the preliminary design stages, rather than when the vessel is already built. This paper uses our algorithm and shows that it outperforms regular random forests in their context. It also shows that their own extension, which leverages random rotations plus their own tuning algorithm for the hyperparameters, outperforms both versions.

One particularly interesting application of random rotations is discussed in Andrews et al. (2017), where the authors use X-ray images to provide non-intrusive inspection for freight shipping containers. They directly leverage our work to introduce a randomly rotated version of their algorithm and show that the detection performance is improved when the feature values are randomly rotated, along with an additional enhancement.

Poona et al. (2016) use tree-based ensembles for the classification of hyper-spectral data. They show that univariate orthogonal splits lead to "staircase or box-like decision boundaries" and discuss approaches to avoid this. A different paper addressing hyperspectral image classification is Zhang and Cao (2020), which explicitly introduces space transformations (including rotations) to increase diversity and improve accuracy. They also discuss accuracy-guided pruning of the trees in the ensemble.

Ani et al. (2017) use a rotation forest ensemble for medical diagnosis in decision support systems. The authors attempt to use structured rotations and show that rotations based on LDA outperform the PCA-based rotations in the original rotation forest paper for their application.

Lefevre et al. (2018) is an example of a paper that does not use random rotation ensembles but leverages our code to produce random rotations and cites our work to show that a competing paper is producing biased results due to non-uniform rotations. The paper introduces a spectral clustering framework in neuroscience (for individual and group parcellation of cortical surfaces in lobes).

Another application is covered in Farayola et al. (2018), where the goal is to extract maximum power from photovoltaic (PV) systems using machine learning.

Further applications are discussed in Pohjalainen (2017), where service contract churn is predicted with decision tree models and Oneto, Siri, et al. (2017), where university student dropout is predicted. In the first case, the paper states that "The fact that decision trees are limited to splitting the space into rectangular areas can be a hindrance, especially in the case when the true decision boundary does not follow such a shape." They cite random rotations as a remedy.

Foundjem (2017) tackles a problem in virtualized cloud computing: predicting process migration failures (across physical machines). The paper does not actually implement random rotations but rather uses our work as a justification for using decision forests in the first place.

The estimation of thermal resistance from aerial thermal imagery is covered in Alshatshati and Hallinan (2017), which estimates building energy efficiency (R-value) from thermal images. The paper cites our work but does not discuss it in detail.

In BrainSpace – a Python/Matlab toolbox to detect and visualize cortical gradients – the authors of de Wael et al. (2020) reused our code for sampling from the set of rotations uniformly at random.

4.2 Extensions of random rotation ensembles

The authors of Lulli et al. (2019) and Lulli et al. (2017) have created a distributed (scalable) random forest implementation called ReForeSt. They explicitly state that "a recently proposed improved random forest formulation called random rotation ensembles can be used in conjunction with model selection to automatically tune the random forest hyperparameters". They demonstrate convincingly that their implementation, even when compared to state-of-the-art alternatives, such as MLlib, is "less computationally intensive, more memory efficient, and more effective". The authors include an entire section on random rotations, including an algorithm in pseudocode. They are also able to confirm the effectiveness of this approach and provide figures to demonstrate this. In fact, in half of their data sets (which differ from the ones we tested), they showed an improvement when using random rotation when compared to using a plain random forest without rotations. This again suggests that users should be given a choice to rotate.

A cluster of Russian papers including Kitov (2016), Mikhailovich (2018), and Sergeyevich (2017) discuss random rotations in the context of Boosting. The reason this is a worthwhile endeavour is that Boosting introduces a dependency between the different decision trees (and hence the different rotations), making the implementation more technically challenging. In Chapter 3, we show a remedy for this but the original approach discussed in Chapter 1 is less practical in this sense. The papers confirm that random rotations are effective in this context and, in the case of the first paper the conclusion was as follows: "It was found that the advantage of the proposed approach (in the case of Boosting) is greatest with large training sample size, small feature dimensions and with lower noise levels. The method provides clear advantages for classes that are linearly or piecewise linearly separable, as well as cases where symmetry is present" and "The method is less preferred for cases where classes are separated by shapes that have as the axis of symmetry one of the axes of the original feature space".

In Cyarnowski and Jedryejowicz (2018) the authors apply various techniques, including stacking, rotation and agent population learning techniques and explicitly state that "the rotation-based techniques are used to increase the heterogeneity of the stacking ensembles".

Taxi fare prediction in New York City is covered in the term paper Antoniades et al. (2016) from Stanford University. The student authors cite our work, noting that "with the hypothesis that the avenue or street could explain some of the effect of the location, transforming the coordinates so that the splits in the random forest algorithm made aligned and perpendicularly to the avenues and streets, could potentially yield better predictions". In a limited way, this citation indicates that the publication of our paper may have inspired others to consider looking at a problem from different viewpoints.

In the French paper Genuer and Poggi (2017) the authors examine the concepts of feature importance and variable selection in traditional and oblique decision trees. Random rotations are shown as an approach to overcome the intrinsically axis-parallel nature of tree-based methods. The authors later also published a very insightful book on the topic (Genuer and Poggi, 2020).

Joly (2016) mentions random rotations along with other oblique methods as an "input-based randomization" rather than an "output-based randomization" where the outputs of the model are perturbed, for example with Gaussian noise.

Kazllarof et al. (2019) applies rotation to multiclass classification problems but with emphasis on structured rotations.

An application of random transformations applied to ensemble clustering is discussed in G. Rodrigues et al. (2020). This work was later also made available in thesis format. It combines different clustering algorithms to reduce noise and sensitivety to outliers. The paper extends our work by also considering Mahalanobis distance (distorting the dataset by multiplying the data matrix by a random covariance matrix) and density proximity (attracts or repulses data points using cluster centroids as references) in addition to random rotation matrices. Random rotations make it into the top 5 low-dimensional and high-dimensional empirical test results but the injection of randomness based on their Mahalanobis distance model show the best overall performance.

Yang et al. (2019) is a business research paper that addresses the measurement error problem by combining machine learning with econometric analysis. They mention our work as "future research directions worth pursuing" and also hint at some of the work described in Chapter 2.

Robotic motion planning is the subject of Wuelker et al. (2019). In the words of the authors the goal is to "develop an alphabet of basic motions from which discrete words that capture the essence of a continous motion/action are constructed. Throughout the literature, discretization of motions has attracted significant interest, where the Euclidean group is one of the popular ones. In particular, uniform sampling of rotations, either random or deterministic, has a wide range of applications". The paper cites seven such papers, including our own.

Hang et al. (2019), Cai et al. (2020) and Hang (2019) extend random rotations (via rotation matrix R) with random stretchings (diagonal stretching matrix S) and random translations (translation vector b) to a complete transformation vector (R, S, b). The complete histogram transform, as it is called in the paper, is $H(x) = R \cdot S \cdot x + b$ in matrix notation. The first paper contains different versions of partitions: naive subdivision of space (NHTE), kernel histogram transform (KHTE), and the adaptive kernel histogram transform ensemble (AKHTE), which includes data-dependent partitions with more splits where more data points are available. The other papers apply the same approach to regression and density estimation.

In Zhou et al. (2018) the authors combined random forests with partial least squares (and the Buckley-James estimator, which extends least-squares to cover the case of censored dependent variables). First they take random subspaces (similar to rotation forests) and then they censor the survival times using Buckley-James and then apply partial least squares. This paper incorrectly groups our work into the "rotation forest based approach" and then claims that "prediction accuracy and additional data diversity are obtained via reconstructing covariates through keeping all principal components extracted from a group of bagged variable subsets". While this is true for rotation forests, it is not at all the case for random rotation ensembles. This, in turn, leads to the misleading conclusion that "In these PCA rotation-based approaches, the objective of orthogonal linear combinations are constructed so as to maximize the variance of the linear combinations of covariates. However, in high dimensional survival data, this optimization may become problematic for prediction since the obtained principal components exploit only the covariate matrix". Again, the statement may be true for rotation forests but is clearly incorrect for random rotation ensembles.

A slightly different application of partial least squares is presented in Li et al. (2020).

Another creative use of random rotation ensembles is described in Ghose and Ravindrani (2020), a paper that attempts to increase interpretability by building smaller models (smaller trees, etc). They use transforms to get around the problem of boundaries that are not axis aligned.

As indicated in the introduction, a number of approaches have recently been proposed that focus on data privacy during the analysis. For example, Differential Privacy (DP) is a system for publicly sharing information about a dataset by describing the patterns of groups within the dataset while withholding information about individuals in the dataset.

Two such papers are Oneto, Cipollini, et al. (2018) and Oneto, Donini, et al. (2020), which focus on Differential Privacy algorithms and demonstrates that non-private algorithms tend to generalize better but at the cost of privacy. They implement and analyze versions of different decision tree ensembles, including random forests and random rotation ensembles, that are differentially private. Their findings show that on average the generalization performance of the different methods are quite similar but with random forests and random rotation ensembles slightly outperforming. In one dataset (101 dimensions, 1212 data points), random rotation ensembles only exhibit 1/3 of the generalization error of the other methods.

In Oneto, Siri, et al. (2017) student dropouts are predicted under privacy constraints and compared to non-private predictors. They explicitly state that "the purpose is to understand how much the privacy constraints affect our ability of building an effective data driven model". The empirical results show that "comparing the performance of the three algorithms, it can be easily seen that, in the non-private framework, random rotation ensembles show a slightly greater accuracy than random forests" and it is further demonstrated that the privacy-enhanced version is outperformed by both of these methods, as expected.

Applications in software fault prediction and South African forest fires can be found in Moosavi and Mohebbi (2020) and Buthelezi (2020), respectively.

Finally, out own paper Blaser and Fryzlewicz (2021), which forms the basis of Chapter 2 in this thesis, is also an extension of the original paper on random rotation ensembles.

4.3 Oblique trees and forests

Rainforth and Wood (2015) builds oblique hyperplanes at each tree node, taking advantage of local correlation of the features. The paper leverages our procedure for sampling rotations uniformly at random but does so at each split, rather than only once for the entire tree. In Chapter 3 we independently came to the conclusion that there are advantages to this approach.

The authors acknowledge the limitations of axis-aligned decision tree splits and note that oblique approaches - specifically including random rotations - can "improve the ensemble diversity, alleviate sensitivity to rotation, and improve performance for many datasets".

In Tomita, Maggioni, et al. (2017), the authors build robust oblique forests and mention our paper in their literature review, stating "while all of these recent approaches deal with rotational invariance, they fail to address several important issues that random forests can natively handle: (1) scale invariance, (2) efficiency, (3) data corruption, (4) sparsity". They also contest that "however, random rotations of the feature space imply that in general splits will not be sparse (i.e. oblique splits are linear combinations of all features rather than a subset of features)" and "therefore, we conjecture that random rotation ensembles will perform increasingly poorly as the ratio of the number of irrelevant features to the number of relevant features becomes larger" and "furthermore, we suspect that linearly combining features will lead to higher sensitivity to data corruption". Unfortunately, no proof or empirical study is performed to confirm these conjectures. In our own studies, which were part of Chapter 2, we have shown that that it is true that the efficacy of the approach reduces then the signal-to-noise ratio becomes very low and that it makes sense to focus on rotations that are helpful.

An interesting paper Balestriero (2017) targets the "synergistic melting of neural networks and decision trees". The authors deploy an independent multilayer perceptron at each layer of the decision tree. This leads to arbitrary nonlinear decision functions at each layer. They cite our paper, together with random projection trees and PCA trees as a related approach to address the fact that decision trees are not rotation invariant.

Majumder (2020) discusses ensembles of oblique decision trees in great detail in the form of a PhD thesis. The numerical tests again show that random rotations are a worthwhile addition to decision tree learners. One interesting finding is that "for shallower trees, oblique tree-based algorithms do not consistently outperform standard tree learners. The expectation that oblique splits are more representative and expressive and can produce good performance with shallower depths is not apparent in the bagging context".

Guided random forests (Gupta et al., 2019) extend the idea of building oblique decision trees with localized partitioning. In a first step, data points are randomly sub-spaced, then a random (oblique) hyperplane is introduced as a split and the partition with the highest impurity is used to apply the split. The paper claims that "this process of tree construction eventually bridges the gap between boosting and decision trees, where every tree represents a high variance instance". It looks like a promising approach for future research.

Decision splits in trees are either found via exhaustive search or randomly. In Wang et al. (2020), the authors use PCA to make the split decisions. They claim that this is faster than exhaustive search and more accurate than random splits. The approach sounds similar to rotation forests (which they also cite in their paper) but rotation forests don't use PCA to make the split decisions but rather to find useful subspaces, so the application of the structured rotation is different.

4.4 Aggregation and weighting schemes

Verma (2019) uses error correcting output codes to improve the adversarial robustness of deep neural networks. They state that "promoting diversity across the constituent learners is crucial and is generally a priority in ensemble-based methods" and indicate that randomly rotating the feature space is one way to accomplish this. They also suggest training each ensemble member (in their case a deep neural network) on different rotations or with distinct architectures.

Armano and Tamponi (2018) propose to use different ensemble components on different regions of the sample space ("local trees"). They state "in the proposed algorithm, the diversity between the components of the ensemble is increased by forcing them to become experts on different regions of the sample space". Random rotations are mentioned as a "feature manipulation" technique, along with 5 others.

A different local approach is presented in Kannao and Guha (2015). The authors state that while "other ensemble frameworks have used fixed weights to determine the influence of each of the component classifiers on the ensemble decision" it is the case that "in practice base classifiers usually have expertise in local regions of the feature space". We confirm this hypothesis in Chapter 3, where we show that it is useful to have access to different rotations of the feature space at the same time in order to leverage the best rotation for each local region.

Dynamic ensemble selection (DES) is described in Narassiguin (2018) as an approach where "different models have different areas of expertise in the instance space". They propose two novel DES approaches: ST-DES (for decision trees): prunes trees using internal supervised tree-based metric (rather than Euclidean distance, which suffers from the curse of dimensionality and PCC-DES (multi-label learning). A nice figure on page 15 shows how random rotations in the case of a spiral decision boundary provides a visibly smoother boundary and "rotating the training set can sometimes give better generalization abilities to the ensemble learning process".

T. Nguyen, M. Nguyen, et al. (2018) introduces a framework for combining multiple classifiers in an ensemble system. In this approach, uncertainty in the predictions of the base classifiers on training observations is captured as an interval-based representation. Out-of-sample, the distance of each class prototype to the input is determined and then the label with the shortest distance is selected. In their paper, the authors differentiate between three different research areas on ensemble methods: (1) design of new ensemble systems (they consider random rotations to be in this category), (2) enhancements to existing ensemble methods (for example boosting, bagging, and random subspaces) and (3) study on properties of the ensemble (for example, different measures of diversity, margin, generalization or error bounds).

Barabec and Machlica (2018) applies decision forests for network intrusion detection. This problem is particularly tricky because the amount of data is vast, while specific patterns of network attacks are rare. For this reason, the most important decision class is heavily underrepresented in the test data. They use a Bayesian-based aggregation of decision trees in an ensemble for classification with samples significantly skewed toward one of the classes. A different application of tree-based ensembles for malware detection is discussed in Euh et al. (2020), where a set of low-dimensional features is proposed and evaluated with a number of different tree-based ensembles. The authors also implement Python-based models in the scikit-learn framework.

The authors of Junior et al. (2018) present heuristics for a multiagent reinforcement learning system. The paper cites our work in relation to the difficulty in combining multiple learners.

4.5 Random projections

As indicated in the introduction, random projections are closely related to random rotations. One big advantage of random projections is that they can be generated very time and space efficiently when compared to random rotations and can be combined easily into a basis for an aggregate classifier. This is related to the ideas in Chapter 3 of this thesis.

For example, Cannings and Samworth (2017) and Cannings (2020) use groups of random projections and apply an arbitrary base classifier on these groups, selecting the projections that yield the smallest estimate of the test error. The latter paper includes a very interesting theoretical background on the use of low-dimensional projections for the analysis of high-dimensional distributions and the different approaches of using random projections (ensemble methods vs sketching and hashing). More detail on how this paper relates to our own approach is provided in the next section.

Another example of this is Tomita, Browne, et al. (2020), which uses sparse random projections. They state that "many recent extensions to decision forests are based on axis-oblique splits. Unfortunately, these extensions forfeit one or more of the favorable properties of decision forests based on axis-aligned splits, such as robustness to many noise dimensions, interpretability, or computational efficiency". Unfortunately, they do not explicitly state which of the above mentioned properties is forfeited by each method.

An interesting classifier is described in Tian and Feng (2021), which aggregates many weak learners, whereby each learner is a base classifier trained in a subspace optimally selected from a collection of random subspaces. They propose algorithm called Random subspace ensemble classification (RaSEn) and have created a very useful R package of the same name. The authors note that in high-dimensional settings, a large number of random subspaces need to be generated but this effect can be reduced with the introduction of an iterative version of their algorithm.

T. Nguyen, Dang, et al. (2019) introduces a weighted multiple classifier framework based on random projection. They generate random projections to lower dimensional spaces they call "training sets" and then apply decision trees on each set and combine the results linearly using least-squares.

In Tasoulis and Vrahatis (2018) multiple random projections are used to visualize high-dimensional single-cell RNA-sequencing data. For this application, projections make much more sense than rotations, which retain the original dimensionality of the data set.

It should also be noted that the efficiency of random projections as a preprocessing step for classification comes at a cost because information is lost as part of the dimension reduction, independently of which classifier is used. More specifically, the dimension reduction is not coordinated with the classification and may even interfere with the classification in the worst case. In addition, most random projections are not helpful to the classification, leading to the same trade-offs we have demonstrated for random rotations.

Examples of summary articles include Sagi and Rokach (2018), Antoniades et al. (2017), Pretorius (2016) and Gonzales et al. (2020)

One notable example of a highly innovative approach that cites our work but does not fit into any of the above categories is presented in Y. Liu and Yin (2019). In this paper, the authors introduce the Delaunay triangulation learner (DTL), which partitions the feature space into a series of p-dimensional simplices and then proceed by finding a linear model within each simplex. The result is a very interesting approximator of low-dimensional smooth functionals that is nonparametric, differentiable and geometrically optimal. The paper also includes some insightful comparative images of Delaunay triangulation versus tree-based learners and MARS.

4.5.1 Comparing rotations with projections

A shortened version of this section was published in the Journal of the Royal Statistical Society, Statistical Methodology, Series B in 2017 as an invited discussion of a third party paper. We reprint it here as a high-level comparison our method of Chapter 1 with the work in Cannings and Samworth (2017).

In our own work on random rotation ensembles we focused on the random rotation of the feature space prior to applying a classifier designed for use in high-dimensional settings. In contrast, the authors of Cannings and Samworth (2017) propose to first project the high-dimensional feature space into a lower dimensional space prior to applying a classifier designed for use in low-dimensional settings. The two strategies are closely related.

In their paper, the authors propose that projections are performed randomly under the Haar measure in the general case, while a particular embodiment with axis-aligned random projections – really random feature selections – is mentioned as well. In our opinion, combining random rotations with axis-aligned projections effectively results in random projection ensembles. Indeed, such a hierarchical decomposition into random rotations (as a feature-preserving linear operation to change the viewpoint) and axis-aligned projections (as a feature selection operation that is delegated to the classifier) might prove to be valuable for future research, as it streamlines mathematical analysis. In particular, the decomposition might address the question, if the benefit of a particular random projection arises from an advantageous viewpoint at the problem due to the rotation or from an effective dimension reduction due to the feature selection, as the two operations can be analysed separately.

Since we advocated the use of classifiers that perform axis-aligned projections in our own papers – such as random forests or other tree-based ensemble classifiers – we would expect random rotation ensembles and random projection ensembles to perform similarly well in practice.

Other attempts have been made in the past to use random projections for dimension reduction in classification problems but the present paper sets itself apart through a clean and well-structured approach. The derived algorithm is clearly original, though it should be noted that neither the idea of dimension reduction prior to analysis, nor random projections applied to classification problems are entirely novel concepts.

However, there are two key areas where the authors provide interesting additional insights: (1) the selection of projections using a two-stage procedure and (2) the data-driven selection of the optimal voting threshold. We would like to discuss these now.

The authors note that most of the random projections are not very helpful in classification because they do not assist in the separation of the classes. In our work on random rotations we noticed a similar pattern: some rotations are extremely helpful but the vast majority — depending on the problem setting - do not improve classification. Hence, a natural question to ask is how we can identify (or explicitly generate) only the most helpful rotations or projections.

The most obvious way to address this issue is by performing a large number of candidate rotations or projections and retaining only the S most successful candidates. In our paper we mentioned dropping the least successful decile of rotations but other approaches are possible, as discussed in Chapters 2 and 3 of this thesis. We adopted a conservative approach to avoid needing to generate a much larger number of rotations but at the cost of retaining fewer good candidates. In contrast, it is possible to use analytical methods, such as PCA to determine successful rotations. The authors of Rodriguez et al. (2006) used this approach but for random subsets of the features.

The authors of the discussed paper use a two step approach: at each stage, B2 projections are generated but only the most successful of these projections is retained. This algorithm is repeated B1 times. Since the learners in the paper are not additive, this two-stage procedure is very similar to producing B1 $\times$ B2 projections and only retaining the best B1 candidates. However, in a more general setting, the two-stage procedure could be valuable because the selection of each additional projection could depend on the marginal performance improvement of the ensemble, conditional on the previously selected projections.

An uncomfortable reality with random rotations (and hence random projections) is that different rotations often prove useful at different subsets of the data. For non-linear decision boundaries this is quite evident. Hence, it might be possible to produce an ensemble classifier that divides the feature space into smaller blocks, each of which is rotated independently. This approach is explored in Chapter 3 of this thesis.

The authors also suggest a data-driven selection of the optimal voting threshold within the ensemble. In our view, this is an excellent contribution, although it is not straight-forward to generalise to multi-class problems.

5 Software

The purpose of computing is insight, not numbers.

— **Richard Hamming**

(Mathematician)

The R-Package randomrotation/random.rotation was created to make the ideas in this thesis accessible and reproducible by others and to make the tools available in the public domain via GitHub. No warranties are expressed or implied and the package should not be used for any commercial purposes.

This chapter provides a brief introduction to the package in the form of a usage example.

Note: to install the package, please follow these steps:

```
> library("devtools")
> install_github("randomrotation/random.rotation")
```

Note: The example uses the IRIS data set from the UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets.php

```
# EXAMPLE

library(random.rotation)   # load the R package
library(randomForest)      # random forest classifier

set.seed(42)     # reproducible results
```

```
NUM_ROTA <- 20   # number of rotations to create
NUM_TREE <- 20   # number of trees per rotation
```

First, the data set is read and pre-processed. Type conversions are performed automatically for string columns with many numeric entries. Columns with too many unique string- or factor values are removed, as these are essentially keys into the data. Columns with extreme duplication are also removed, as these tend not to add much predictive value. No scaling or rotation is performed here.

```
pre <- pre_sample_preprocessing(iris, "Species")
X_pre <- pre[[1]]  # pre-processed X matrix
Y_pre <- pre[[2]]  # pre-processed Y vector
```

Next, we pre-create an array of NUM_ROTA - 1 random rotation matrices plus the identity rotation. It would also be possible to generate these one-by-one inside of the test loop but creating them here keeps the code more flexible.

```
mArray <- create_random_rotation_matrix_array(NUM_ROTA-1,
                                    length(numeric_cols(X_pre)))
```

We proceed by dividing the input data into random, disjoint training (70%) and testing (30%) data rows

```
r_train <- generate_training_row_indices(nrow(X_pre), 0.7)
r_test  <- generate_testing_row_indices(nrow(X_pre), r_train)
```

Then the already pre-processed input data is further processed by scaling numeric columns to [0,1] and imputing missing numeric values to the median of the in-sample values. Both of these operations are performed with only in-sample inputs to ensure no out-of-sample data can have an impact.

```
X <- post_sample_preprocessing(X_pre, Y_pre, r_train, r_test)
Y <- Y_pre
```

Working now on scaled numeric values, we can finally apply the rotations. As discussed in the paper, there are many ways to scale the data but it is definitely not advisable to perform rotations on unscaled input data. We create an array of transformed predictors, one entry for each rotation.

```
tArray <- df_apply_random_rotations(X, mArray)
```

Now that we have the rotated predictors nicely lined up, it is time to apply a classifier to each set of rotated predictors. In this case, we take advantage of the excellent randomForest R package to create an array of classifiers. Obviously, only training data is used.

```
cArray <- lapply(1:NUM_ROTA,
    function(i) randomForest(x=tArray[[i]][r_train,],
                             y=Y[r_train],
                             ntree=NUM_TREE))
```

Here we get to the meat of the paper: our goal is to find those rotations that produced classifiers with the lowest complexity. On page 3 of the paper in formula (5), we define complexity in terms of the median number of nodes of the trees in the forest plus a tie-breaker based on the depth of the trees. Obviously, other measures are possible here but let's stick with this one.

```
nodes <- sapply(1:NUM_ROTA,
    function(i) median(cArray[[i]]$forest$ndbigtree))

depth <- sapply(1:NUM_ROTA,
    function(i) mean(sapply(1:NUM_TREE,
        function(x) tree_height(cArray[[i]]$forest$treemap[,,x]))))

xArray <- nodes + depth/length(r_train)  # complexity
```

At this point, we know the average complexity of the trees in each of the NUM_ROTA rotations. Instead of equal-weighting all predictors across the different rotations, we want to over-weight those predictors that benefit from favorable rotations. For this reason, we will sort all rotations by complexity. In addition, we look at the out-of-bag error but only to help us decide how many of the least complex rotations we should consider and how much weight we should put on each rotation. In either case, rotations with lower complexity scores *always* carry a weight that is at least as high as rotations with higher complexity scores.

```
oArray <- as.numeric(sapply(1:NUM_ROTA,
    function(i) cArray[[i]]$err.rate[NUM_TREE,1]))
```

With the out-of-bag error known, we can now proceed with actually computing the weights of each rotation. In the present example, we use the exponential weighting function described on pages 4-5 of the paper. In a first step, the tuning parameter h is optimized, as described in the previous comment above. Then the weight function is applied to obtain a weight for the predictors of each rotation.

```
h <- compute_tuning_parameter(xArray, oArray, weights_exp,
                              min_step=0.1, step_size=0.1)

wArray <- weights_exp(NUM_ROTA, h)
```

In the paper we describe the possibility of adding more trees to each rotation, depending on the computed weights and the total number of desired trees in the ensemble. For the sake of clarity, we omit this step in this example but the number of required trees on each rotation can be computed using the function num_trees_per_rot in the package.

And finally, let's investigate the out-of-sample performance of the ensemble by applying the now weighted ensemble to the test data. First we obtain a vector of out-of-sample predictions and corresponding errors and then we tabulate the weighted votes from different rotations to obtain a merged prediction.

```
pArray <- lapply(1:NUM_ROTA,
    function(i) predict(cArray[[i]], newdata=tArray[[i]][r_test,]))

v1 <- Reduce('+', lapply(1:NUM_ROTA,
    function(i) sapply(levels(Y), function(x)
        wArray[i]*(pArray[[i]]==x))))

v2 <- levels(Y)[(apply(v1,1,max)==v1) %*% (1:length(levels(Y)))]
```

The resulting ensemble prediction error is then given by

```
sum(Y[r_test]!=v2)/length(r_test)
```

Bibliography

Abbott, D. (2012). “Why Ensembles Win Data Mining Competitions”. In: *Predictive Analytics Centre of Excellence Tech Talks*. University of California, San Diego (cit. on p. 1).

Alshatshati, S. and Hallinan, K. (2017). “Data Mining Approach for Estimating Residential Attic Thermal Resistance From Aerial Thermal Imagery, Utility Data, and Housing Data”. In: *Energy Sustainability (The American society of mechanical engineers MSME)* (cit. on p. 102).

Anderson, T. W., Olkin, I., and Underhill, L. G. (1987). “Generation of random orthogonal matrices”. In: *SIAM Journal on Scientific and Statistical Computing* 8, pp. 625–629 (cit. on pp. 11, 44).

Andrews, J., Jaccard, N., and Rogers, T. (2017). “Representation-learning for anomaly detection in complex x-ray cargo imagery”. In: *Proceedings of SPIE, Anomaly Detection and Imaging with X-Rays (ADIX)* (cit. on p. 100).

Ani, R., Jithu, J., Manu, W., and Deepa, O. (2017). “Modified Rotation Forest Ensemble Classifier for Medical Diagnosis in Decision Support Systems”. In: *Conference paper for Progress in Advanced Computing and Intelligent Engineering (Springer)* (cit. on p. 101).

Antoniades, A., Fadavi, D., and Amon, A. (2016). “Fare and duration prediction. A study of New York city taxi rides”. In: *Unpublished student paper* (cit. on p. 104).

– (2017). “Decision Tree and Decision Forest Algorithms: On Improving Accuracy, Efficiency and Knowledge Discovery”. In: *PhD thesis* (cit. on p. 114).

Armano, G. and Tamponi, E. (2018). “Building forests of local trees”. In: *Pattern Recognition (Elsevier)* (cit. on p. 110).

Bache, K. and Lichman, M. (2013). *UCI machine learning repository*. University of California, Irvine, School of Information and Computer Science (cit. on p. 19).

Badirli, S., Liu, X., Xing, Z., and Keerthi, S. (2020). “Gradient Boosting Neural Networks: GrowNet”. In: *arXiv* (cit. on p. 6).

Balestriero, R. (2017). “Neural decision trees”. In: *arXiv* (cit. on p. 109).

Barabec, J. and Machlica, L. (2018). “Decision-forest voting scheme for classification of rare classes in network intrusion detection”. In: *Conference paper for the 2018 IEEE International Conference on Systems, Man and Cybernetics* (cit. on p. 112).

Blaser, R. and Fryzlewicz, P. (2016). “Random rotation ensembles”. In: *Journal of Machine Learning Research* 17, pp. 1–26 (cit. on pp. 1, 6, 97).

– (2021). “Regularizing axis-aligned ensembles via data rotations that favor simpler learners”. In: *Statistics and Computing (Springer)* 31 (cit. on pp. 39, 108).

Breiman, L. (1996). “Bagging Predictors”. In: *Machine Learning* 24, pp. 123–140 (cit. on p. 2).

– (1999). *Random Forests – Random Features*. Tech. rep. University of California at Berkeley, Berkeley, California (cit. on pp. 3, 40).

– (2000). “Randomizing Outputs to Increase Prediction Accuracy”. In: *Machine Learning* 40, pp. 229–242 (cit. on p. 2).

– (2001). “Random forests”. In: *Machine Learning* 45, pp. 5–32 (cit. on pp. 3, 18, 32).

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth (cit. on p. 42).

Buthelezi, M. (2020). “The use of machine learning algorithms to assess the impacts of droughts on commercial forests in KwaZulu-Natal, South Africa”. In: *Unpublished draft* (cit. on p. 107).

Cai, Y., Hang, H., and Lin, Z. (2020). “Boosted Histogram Transform for Regression”. In: *Proceedings of the 37th International Conference on Machine Learning* (cit. on p. 105).

Cannings, T. (2020). “Random projections: Data perturbation for classification problems”. In: *WIREs Computational Statistics (Wiley)* (cit. on p. 112).

Cannings, T. and Samworth, R. (2017). “Random-projection ensemble classification”. In: *Journal of the Royal Statistical Society: Series B* 79, pp. 1–38 (cit. on pp. 40, 112, 115).

Caruana, R. and Niculescu-Mizil, A. (2006). “An Empirical Comparison of Supervised Learning Algorithms”. In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 161–168 (cit. on p. 1).

Chen, T. and Guestrin, C. (2016). “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, pp. 785–794 (cit. on p. 6).

Cipollini, F., Oneto, L., Coraddu, A., and Murphy, A. (2018a). “Condition-based maintenance of naval propulsion systems: Data analysis with minimal feedback”. In: *Reliability Engineering and System Safety (Elsevier)* (cit. on p. 100).

– (2018b). “Condition-based maintenance of naval propulsion systems with supervised data analysis”. In: *Ocean Engineering (Elsevier)* (cit. on p. 100).

Cutler, A. and Zhao, G. (2001). “PERT-perfect random tree ensembles”. In: *Computing Science and Statistics* 33, pp. 490–497 (cit. on p. 3).

Cyarnowski, I. and Jedryejowicz, P. (2018). “An approach to data reduction for learning from big datasets: integrating stacking, rotation , and agent population learning techniques”. In: *Complexity* (cit. on p. 103).

De Bock, K. W. and Van den Poel, D. (2011). “An empirical evaluation of rotation-based ensemble classifiers for customer churn prediction”. In: *Expert Systems with Applications* 38, pp. 12293–12301 (cit. on p. 4).

de Wael, V., Benkarim, O., and Paquola, C. (2020). “BrainSpace: a toolbox for the analysis of macroscale gradients in neuroimaging and connectomics datasets”. In: *Communications Biology (Nature)* 3 (cit. on p. 102).

Delgado, M., Cernadas, E., and Barro, S. (2014). “Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?” In: *Journal of Machine Learning Research* 15 (cit. on p. 6).

Desai, U., Martis, R., and Acharya, U. (2016). “Diagnosis of multiclass tachycardia beats using recurrence quantification analysis and ensemble classifiers”. In: *Journal of Mechanics* (cit. on p. 100).

Dheeru, D. and Taniskidou, E. K. (2017). *UCI Machine Learning Repository* (cit. on p. 59).

Diaconis, P. and Shahshahani, M. (1987). “The Subgroup Algorithm for Generating Uniform Random Variables”. In: *Probability in the Engineering and Informational Sciences* 1, pp. 15–32 (cit. on pp. 10, 11).

Durrant, R. and Kaban, A. (2013). “Random projections as regularizers: learning a linear discriminant ensemble from fewer observations than dimensions”. In: *JMLR: Workshop and Conference Proceedings* 29, pp. 17–32 (cit. on p. 40).

Elghazel, H., Aussem, A., and Perraud, F. (2011). “Trading-Off Diversity and Accuracy for Optimal Ensemble Tree Selection in Random Forests”. In: *Ensembles in Machine Learning Applications*. Studies in Computational Intelligence. Springer Berlin Heidelberg, pp. 169–179 (cit. on p. 2).

Euh, S., Lee, H., Kim, D., and Hwang, D. (2020). “Comparative analysis of low-dimensional features and tree-based ensembles for malware detection systems”. In: *IEEE Access* (cit. on p. 112).

Fan, W., McCloskey, J., and Yu, P. S. (2006). “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. New York, NY, USA: ACM, pp. 136–146 (cit. on p. 3).

Farayola, A., Sun, Y., and Ali, A. (2018). “Optimization of PV Systems Using Linear Interactions Regression MPPT Techniques”. In: *2018 IEEE PES/IAS PowerAfrica* (cit. on p. 101).

Fawagreh, K., Gaber, M. M., and Elyan, E. (Sept. 2014). “Random forests: from early developments to recent advancements”. In: *Systems Science & Control Engineering* 2.1, pp. 602–609 (cit. on p. 3).

Foundjem, A. (2017). “Towards Improving the Reliability of Live Migration Operations in Openstack Clouds”. In: *MSc thesis* (cit. on p. 102).

Freund, Y. and Schapire, R. (1996). “Experiments with a New Boosting Algorithm”. In: *Proceedings of the Thirteenth International Conference on Machine Learning*. Bari, Italy: Morgan Kaufmann Publishers Inc., pp. 148–156 (cit. on pp. 2, 18).

Friedman, J. (2001). “Greedy function approximation: a gradient boosting machine”. In: *The Annals of Statistics* 5, pp. 1189–1232 (cit. on p. 42).

Galassi, M. (Jan. 2009). *GNU scientific library reference manual - third edition*. Network Theory Ltd. (cit. on p. 12).

García-Pedrajas, N., García-Osorio, C., and Fyfe, C. (2007). “Nonlinear Boosting Projections for Ensemble Construction”. In: *Journal of Machine Learning Research* 8, pp. 1–33 (cit. on p. 4).

Genuer, R. and Poggi, J. (2017). “Arbres CART et Forêts aléatoires, Importance et sélection de variables”. In: *HAL open access archive* (cit. on p. 104).

– (2020). *Random Forests with R (Use R!)* Springer (cit. on p. 104).

Geurts, P., Ernst, D., and Wehenkel, L. (2006). “Extremely randomized trees”. In: *Machine Learning* 63, pp. 3–42 (cit. on pp. 3, 18).

Ghose, A. and Ravindrani, B. (2020). “Interpretability with accurate small models”. In: *Frontiers in Artificial Intelligence* (cit. on p. 106).

Goldbloom, A. (2016). *What algorithms are most successful on Kaggle?* `https://www.kaggle.com/antgoldbloom/what-algorithms-are-most-successful-on-kaggle` (cit. on p. 6).

Gonzales, S., Garcia, S., Ser, J. D., and Rokach, L. (2020). “A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities”. In: *Information Fusion (Elsevier)* 64 (cit. on p. 114).

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press (cit. on p. 7).

Goodrich, M., Mirelli, V., Orletsky, M., and Salowe, J. (1995). “Decision tree construction in fixed dimensions: being global is hard but local greed is good”. In: *Technical Report TR95-1* (cit. on p. 42).

Guennebaud, G., Jacob, B., et al. (2010). *Eigen: linear algebra template library* (cit. on p. 13).

Gupta, P., Jindal, A., and Sengupta, D. (2019). “Guided Random Forest and its application to data approximation”. In: *arXiv* (cit. on p. 109).

Hang, H. (2019). “Histogram Transform Ensembles for Density Estimation”. In: *arXiv* (cit. on p. 105).

Hang, H., Lin, Z., Liu, X., and Wen, H. (2019). “Histogram Transform Ensembles for Large-scale Regression”. In: *arXiv* (cit. on p. 105).

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*. English. New York: Springer (cit. on pp. 2, 3, 7, 9, 16, 17, 41).

Ho, T. K. (1995). “Random decision forests”. In: *Proceedings of the Third International Conference on Document Analysis and Recognition*. Vol. 1, pp. 278–282 (cit. on p. 2).

– (1998). “The random subspace method for constructing decision forests”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, pp. 832–844 (cit. on p. 2).

Householder, A. S. (1958). “Unitary Triangularization of a Nonsymmetric Matrix”. In: *Journal of the ACM* 5, pp. 339–342 (cit. on p. 12).

Hyafil, L. and Rivest, R. (1976). “Constructing optimal binary decision trees is NP-Complete”. In: *Information Processing Letters* 5, pp. 15–17 (cit. on p. 42).

James, G., Witten, D., and Hastie, T. (2013). *An introduction to statistical learning*. New York: Springer (cit. on p. 42).

Joly, A. (2016). “Exploiting random projections and sparsity with random forests and gradient boosting methods - Application to multi-label and multi-output learning, random forest model compression and leveraging input sparsity”. In: *arXiv* (cit. on p. 104).

Junior, A., de Freitas Filho, P., and Silveira, R. (2018). “iEnsemble2: Committee Machine Model-Based on Heuristically-Accelerated Multiagent Reinforcement Learning”. In: *Conference paper: Complex, Intelligent, and Software Intensive Systems (CISIS)* (cit. on p. 112).

Kadra, A., Lindauer, M., Hutter, F., and Grabocka, J. (2021). “Regularization is all you Need: Simple Neural Nets can Excel on Tabular Data”. In: *arXiv* (cit. on p. 7).

Kannao, R. and Guha, P. (2015). “A novel local success weighted ensemble classifier”. In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)* (cit. on p. 110).

Kazllarof, V., Karlos, S., and Kotsiantis, S. (2019). “Active learning Rotation Forest for multiclass classification”. In: *Computational Intelligence (Wiley)* (cit. on p. 104).

Kerr, A., Campbell, D., and Richards, M. (2009). “QR decomposition on GPUs”. In: *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. GPGPU-2. New York, NY, USA: ACM, pp. 71–78 (cit. on p. 13).

Kitov, V. (2016). “Investigation of the accuracy of the gradient boosting method with random rotations”. In: *Economics, Statistics and Informatics* 4 (cit. on p. 103).

Knuth, D. E. (Nov. 1997). *Art of computer programming, volume 2: seminumerical algorithms*. 3 edition. Reading, Mass: Addison-Wesley Professional (cit. on p. 12).

Kuhn, M. and Johnson, K. (Sept. 2013). *Applied predictive modeling*. English. 2013 edition. New York: Springer (cit. on pp. 3, 24).

Kuncheva, L. I. and Rodriguez, J. J. (2007). “An experimental study on rotation forest ensembles”. In: *Proceedings of the 7th International Conference on Multiple Classifier Systems*. MCS’07. Berlin, Heidelberg: Springer-Verlag, pp. 459–468 (cit. on p. 4).

Ledermann, D. and Alexander, C. (2011). *ROM Simulation with Random Rotation Matrices*. SSRN Scholarly Paper ID 1805662. Rochester, NY: Social Science Research Network (cit. on p. 11).

Lefevre, J., Pepe, A., and Muscato, J. (2018). “SPANOL (Spectral ANalysis of Lobes): A spectral clustering framework for individual and group parcellation of cortical surfaces in lobes”. In: *Frontiers in Neuroscience* (cit. on p. 101).

Li, C., Lin, Y., and Xu, Q. (2020). “An enhanced random forest with canonical partial least squares for classification”. In: *Communications in Statistics - Theory and Methods (Taylor Francis)* (cit. on p. 106).

Liaw, A. and Wiener, M. (2002). “Classification and Regression by randomForest”. In: *R News* 2.3, pp. 18–22 (cit. on p. 22).

Liu, F. T., Ting, K. M., and Fan, W. (2005). “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. Berlin, Heidelberg: Springer-Verlag, pp. 605–610 (cit. on p. 3).

Liu, Y. and Yin, G. (2019). “Nonparametric Functional Approximation with Delaunay Triangulation”. In: *IEEE International Conference on Big Knowledge (ICBK)* (cit. on p. 114).

Lulli, A., Oneto, L., and Anguita, D. (2017). “Crack random forest for arbitrary large datasets”. In: *Conference paper: IEEE Conference on Big Data* (cit. on p. 102).

– (2019). “Mining big data with random forests”. In: *Cognitive Computation (Springer)* 11 (cit. on p. 102).

Majumder, T. (2020). “Ensembles of Oblique Decision Trees”. In: *PhD thesis* (cit. on p. 109).

Matsumoto, M. and Nishimura, T. (1998). “Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator”. In: *ACM Transactions on Modeling and Computer Simulation* 8, pp. 3–30 (cit. on p. 13).

Menze, B. H., Kelm, B. M., Splitthoff, D. N., Koethe, U., and Hamprecht, F. A. (2011). “On oblique random forests”. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*. ECML PKDD’11. Berlin, Heidelberg: Springer-Verlag, pp. 453–469 (cit. on p. 4).

Mezzadri, F. (2007). “How to generate random matrices from the classical compact groups”. In: *Notices of the AMS* 54. NOTICES of the AMS, Vol. 54 (2007), 592-604, pp. 592–604 (cit. on p. 11).

Mikhailovich, S. (2018). “Development of a gradient boosting algorithm with random rotations of a sign space for solving the classification problem”. In: *elibrary.ru* (cit. on p. 103).

Moosavi, T. and Mohebbi, K. (2020). “Software fault prediction based on random forest algorithm”. In: *Journal of software engineering intelligent systems* 5 (cit. on p. 107).

Narassiguin, A. (2018). “Ensemble Learning, Comparative Analysis and Further Improvements with Dynamic Ensemble Selection”. In: *PhD thesis* (cit. on p. 111).

Nawar, S. and Mouazen, A. (2017). “Comparison between Random Forests, Artificial Neural Networks and Gradient Boosted Machines Methods of On-Line Vis-NIR Spectroscopy Measurements of Soil Total Nitrogen and Total Carbon”. In: *Sensors* 17(10) (cit. on p. 6).

Nguyen, T., Dang, M., Liew, A., and Bezdek, J. (2019). “A weighted multiple classifier framework based on random projection”. In: *Information Sciences (Elsevier)* (cit. on p. 113).

Nguyen, T., Nguyen, M., and Pham, X. (2018). “Combining heterogeneous classifiers via granular prototypes”. In: *Applied Soft Computing (Elsevier)* 73 (cit. on p. 111).

Oneto, L., Cipollini, F., Ridella, S., and Anguita, D. (2018). “Randomized learning: Generalization performance of old and new theoretically grounded algorithms”. In: *Neurocomputing (Elsevier)* 298 (cit. on p. 107).

Oneto, L., Cordaddu, A., and Anguita, D. (2017). “Marine safety and data analytics: Vessel crash stop maneuvering performance prediction”. In: *Conference paper: ICANN 2017 Artificial Neural Networks and Machine Learning* (cit. on p. 100).

Oneto, L., Donini, M., Pontil, M., and Shawe-Taylor, J. (2020). “Randomized learning and generalization of fair and private classifiers: From PAC-Bayes to stability and differential privacy”. In: *Neurocomputing (Elsevier)* (cit. on p. 107).

Oneto, L., Siri, A., Luria, G., and Anguita, D. (2017). “Dropout Prediction at University of Genoa: a Privacy Preserving Data Driven Approach”. In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)* (cit. on pp. 101, 107).

Paramanik, M., Pradhan, R., Nandy, P., and Bhoi, A. (2021). “Machine Learning Methods with Decision Forests for Parkinson’s Detection”. In: *MDPI Applied Sciences* 11(2) (cit. on p. 100).

Pohjalainen, V. (2017). “Predicting service contract churn with decision tree models”. In: *MSc thesis* (cit. on p. 101).

Poona, N., Niekerk, A. V., and Ismail, R. (2016). “Investigating the utility of oblique tree-based ensembles for the classification of hyperspectral data”. In: *Sensors (MDPI)* (cit. on p. 101).

Pretorius, A. (2016). “Advances in random forests with application to classification”. In: *MSc thesis* (cit. on p. 114).

Rainforth, T. and Wood, F. (2015). “Canonical correlation forests”. In: *arXiv* (cit. on p. 108).

Ridgeway, G. (2013). *gbm: generalized boosted regression models*. R package version 2.1 (cit. on p. 30).

Rodrigues, G., Albertini, M., and Yang, X. (2020). “An empirical evaluation of random transformations applied to ensemble clustering”. In: *Multimedia Tools and Applications (Springer)* (cit. on p. 104).

Rodrigues, O. (1840). “Des lois géométriques qui régissent les déplacements d’un système solide dans l’espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendants des causes qui peuvent les produire”. In: *Journal de Mathématiques Pures et Appliquées* 5, pp. 380–440 (cit. on p. 12).

Rodriguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). “Rotation forest: a new classifier ensemble method”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, pp. 1619–1630 (cit. on pp. 4, 22, 40, 116).

Rokach, L. (2010). “Ensemble-based classifiers”. en. In: *Artificial Intelligence Review* 33, pp. 1–39 (cit. on p. 1).

Sagi, O. and Rokach, L. (2018). “Ensemble learning: A survey”. In: *Wiley Interdisciplinary Reviews Data Mining and Knowledge Discovery* 8(4) (cit. on p. 114).

Sergeyevich, G. (2017). “Research of boosting algorithms with built-in turns of the feature space”. In: *MSc thesis* (cit. on p. 103).

Simm, J. and de Abril, I. M. (2013). *ExtraTrees: extraTrees method*. R package version 0.4-5 (cit. on p. 22).

Takacs, G., Chandrasekhar, V., and Tsai, S. (2013). “Fast computation of rotation-invariant image features by approximate radial gradient transform”. In: *IEEE Transactions on Image Processing* 22, pp. 2970–2982 (cit. on p. 39).

Tasoulis, S. and Vrahatis, A. (2018). “Visualizing high-dimensional single-cell RNA-sequencing data through multiple random projections”. In: *Conference paper: 2018 IEEE International Conference on Big Data* (cit. on p. 113).

Tian, Y. and Feng, Y. (2021). “RaSE: Random subspace ensemble classification”. In: *Journal of Machine Learning Research (JMLR)* 22(45), pp. 1–93 (cit. on p. 113).

Tibshirani, R. (1996). “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B* 58, pp. 267–88 (cit. on p. 42).

Tomita, T., Browne, J., Shen, C., and Chung, J. (2020). “Sparse Projection Oblique Randomer Forests”. In: *Journal of Machine Learning Research (JMLR)* (cit. on p. 113).

Tomita, T., Maggioni, M., and Vogelstein, J. (2017). “ROFLMAO: Robust Oblique Forests with Linear Matrix Operations”. In: *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM)* 1, pp. 498–506 (cit. on pp. 61, 108).

Treboux, J., Genoud, D., and Ingold, R. (2018). “Decision Tree Ensemble Vs. N.N. Deep Learning: Efficiency Comparison For A Small Image Dataset”. In: *IEEE International Workshop on Big Data and Information Security (IWBIS)* (cit. on p. 6).

Verma, G. (2019). “Error Correcting Output Codes Improve Probability Estimation and Adversarial Robustness of Deep Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 32 (cit. on p. 110).

Wang, F., Wang, Q., and Li, Z. (2020). “A forest of trees with principal direction specified oblique split on random subspace”. In: *Neurocomputing (Elsevier)* 379 (cit. on p. 110).

Wuelker, C., Ruan, S., and Chirikjian, G. (2019). “Quantizing Euclidean motions via double-coset decomposition”. In: *Research (Science partner)* (cit. on p. 105).

Yang, M., McFowland, E., and Burtch, G. (2019). “Achieving Reliable Causal Inference with Data-Mined Variables: A Random Forest Approach to the Measurement Error Problem”. In: *Kelley School of Business Research Paper* (cit. on p. 105).

Zhang, Y. and Cao, G. (2020). “Multiview-Based Random Rotation Ensemble Pruning for Hyperspectral Image Classification”. In: *IEEE Transactions on Instrumentation and Measurement* (cit. on p. 101).

Zhou, L., Wang, H., and Xu, Q. (2018). “Survival forest with partial least squares for high dimensional censored data”. In: *Chemometrics and Intelligent Laboratory Systems* 179 (cit. on p. 106).

List of Figures

List of Tables