ODRF: An R Package for Oblique Decision Random Forest

Yu Liu
National University of Singapore

Yingcun Xia
National University of Singapore

Abstract

CART and Random Forest (*RF*) are arguably the most popular methods in statistical data analysis and forecasting. The use of linear combinations of predictors as splitting variables is one of the important extensions of *CART* and is known as Oblique Decision Trees (*ODT*) and ODT-based Random Forests (*ODRF*). Recent studies have also shown the theoretical advantages of *ODT* and *ODRF* over *CART* and *RF*. However, there is still no integrated and efficient software package that can demonstrate the numerical advantages of *ODT* and *ODRF*. To fill this gap, we developed an **ODRF** R package for both `ODT` and `ODRF`, and provided online structure learning algorithms. The main computational part of `ODT` is executed using the **Rcpp** package, and `ODRF` allows parallel computation. Through numerical experiments, our package was compared with other packages for decision trees and forests, showing a clear overall improvement.

Keywords: CART, oblique decision tree, random forest, projection pursuit, R.

1. Introduction

The Classification and Regression Tree (CART) proposed by Professor Leo Brieman (1984) has attracted a great deal of attention from statisticians and data analysts of other disciplines. The method is widely used because it is easy to train and the resulting tree makes the analysis results visual and easy to interpret (Johnson and Tong 2014). On the other hand, much attention has been paid to the algorithm and many improvements have been proposed. Classification and regression trees (Quinlan 1987, CART) and *C4.5* Quinlan (1993) are the most commonly used decision trees. There is a long list of other decision trees, including the Evolutionary Learning of Globally Optimal Classification and Regression Trees (*EVT*) Grubinger, Zeileis, Pfeiffer, and KP (2014), Conditional Inference Trees (*CT*) Hothorn, Hornik, and Zeileis (2006), Extremely randomized trees (*ERT*) Geurts, Ernst, and Wehenkel (2006), Model-Based Recursive Partitioning (*MOB*) Zeileis and Hothorn (2015), bayesian additive regression trees (*BART*) Maia, Murphy, and Parnell (2022) and generalized linear mixed-model trees (*glmertree*) Fokkema, Edbrooke-Childs, and Wolpert (2020). The Random Forests (Breiman 2001, RF), which is an ensemble of CARTS by either feature bagging or boosting, is arguably the most efficient machine method especially for the tabular data Grinsztajn, Oyallon, and Varoquaux (2022). Again, there are many ensemble methods that based on different decision trees. For example, Conditional Random Forests (*cforest*) Hothorn *et al.* (2006), Learning Nonlinear Functions Using Regularized Greedy Forest (*RGF*) Johnson and Tong (2014), Generalized Random Forest (*GRF*) Athey, Tibshirani, and Wager (2019) and extreme gradient boosting (*XGB*) Chen and Guestrin (2016).

One of the most appealing extensions to CART is the use of linear combinations of the predictors as splitting variables that is known as the Oblique Decision tree Heath, Kasif, and Salzberg (1993). Recently, Zhan, Liu, and Xia (2022) proved the consistency of The Oblique Decision Tree (*ODT*) and Its Random Forest for very general regression functions as long as they are continuous, while CART or RF are consistency mainly for regressions with special structures such as additive structure. Again, ensemle can be made based on ODT resulting the Oblique-type Random Forests, including Random Rotation Random Forest (*RR-RF*) of Blaser and Fryzlewicz (2016), Random Projection Forests (*RPFs*) of Lee, Yang, and Oh (2015) and Sparse Projection Oblique Random Forests (*SPORF*) of Tomita, Browne, Shen, Chung, Patsolic, Falk, Priebe, Yim, Burns, Maggioni *et al.* (2020). Another type is the model-based oblique decision forest, including mainly Canonical Correlation Forests (*CCF*) with classic correlation analysis Rainforth and Wood (2015), projection pursuit forest (*PPF*) with linear discriminant analysis Silva, Cook, and Lee (2021), oblique random forests (*ORF*) with ridge regression Menze, Kelm, Splitthoff, Koethe, and Hamprecht (2011), oblique random survival forests (*ORSF*) **?** and Heterogeneous oblique random forest (*HORF*) Katuwal, Suganthan, and Zhang (2020).

Although the theoretical advantages of oblique decision trees and their random forests have been well understood, the existing packages implementing those extensions only show their better numerical performance in some special cases, and thus not commonly received and have not got as much as popularity as they deserve. As a consequence, the conventional CART and RF are still the most commonly used packages. The main difficulty in implementing *ODT* or *ODRF* is the estimation of the coefficient, θ, for the linear combinations, which is also one of the main differences amongst all the existing packages. The estimation methods of θ include random projection, logistic regression, dimension reduction and many others. For example, the functions `RerF()` in **rerf** package Tomita *et al.* (2020) use random projections, the functions `baggtree()` in **PPforest** package Silva *et al.* (2021) uses `"LDA"` model to estimate the projection directions, and they also provide the `"PDA"`, `"GINI"` and `"ENTROPY"` models in parameter `PPmethod`; the functions `obliqueRF()` in **obliqueRF** package Menze *et al.* (2011) uses `"ridge"` for fast ridge regression using SVD (default), `"pls"` for partial least squares regression, `"svm"` for a linear support vector machine, `"log"` for logistic regression and `"rnd"` for a random hyperplane in parameter `training_method`. Some of the R packages have been taken off from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/` by (R Core Team 2017) due to some problems and have not been modified.

We refine the existing computer R (R Core Team 2017) packages according to the established theory in the article of Zhan *et al.* (2022). Our package, called **ODRF**, is modified from several existing R packages. In **ODRF**, the projection pursuit regression function is used to find θ for each set of q predictors, but other options are also provided in the package. Of course many R packages have been developed to implement *ODT* and *ODRF*. Our **ODRF** R package can be used for classification and regression, and the computational time consumption and estimation accuracy are better than the competitive R package. Comparing with the existing forests, the advantages of **ODRF** are as follows.

- Both the tree and forests of **ODRF** have better overall accuracy than existing trees and forests, including traditional CART and RF, in both classification and regression prediction, respectively.

- **ODRF** can be used for both classification and regression, while most existing packages of oblique-type trees or forests only make classification.
- **ODRF** allows users to define their own functions to find the projections of at each node, which is essential to the performance of the forests.
- **ODRF** also applies to streaming data and continuously improves the existing tree.

The remainder of this paper is organized as follows: Section 2, shows the model or algorithm details of the main functions in the **ODRF** package, so that the user has a clear understanding of the relevant principles. In Section 3, Introduce the usage of **ODRF** package. In Section sec:examples, use the **ODRF** package to compare the predictive effects in classification and regression with other R packages and showcase the specific application of the ODRF package in practice by two data sets with continuous and categorical responses. The summary in Section **??** gives concluding remarks about the implementation and the performance of the new algorithm.

2. The underlying algorithm

Suppose $Y = (y_1, ..., y_q)$ is the response vector of interest and $X : p \times 1$ is the predictor. We allow Y to be multiple to accommodate the categorical response. That is if Y has q classes, then it is represented by q dummy variables with each taking values 0 and 1.

2.1. Create ODT

With observations $\mathbb{A}_0^0 = \{(X_i, Y_i), i = 1, ..., n\}$, where $Y_i = (y_{i1}, ..., y_{ip})$, an ODT is illustrated by the following diagram. For ease of exposition, even if a node will not be split further, we still rewrite it in the next layer. For any node $\mathbb{A}_k^\tau$, where the subscript k represent the layer and superscript the number of nodes in the layer, the splitting is as follows. Given any p-dimensional vector θ and a splitting value c, define the daughter nodes

$$\mathbb{A}_{k+1}^{\tau'} = \{X_i : X_i \in \mathbb{A}_k^\tau,\ \theta^\top X_i \le c\}, \quad \mathbb{A}_{k+1}^{\tau''} = \{X_i : X_i \in \mathbb{A}_k^\tau,\ \theta^\top X_i > c\}.$$

Define objective function

$$\Delta(c|\mathbb{A}_k^\tau, \theta) = \sum_{j=1}^q \sum_{X_i \in \mathbb{A}_k^{\tau'}} (y_{ij} - \bar{y}_{.j})^2 + \sum_{j=1}^q \sum_{X_i \in \mathbb{A}_k^{\tau''}} (y_{ij} - \bar{y}_{.j})^2.$$

When θ is given, the splitting value c should minimize $\Delta(c|\theta, \mathbb{A}_k^\tau)$. If Y is one dimensional quantitative response, then this is the criterion for regression of CART. If Y is categorical, $\Delta(c|\theta, \mathbb{A}_k^\tau)$ is different from the Gini impurity. However, the Gini impurity can also be used and is one option in **ODRF**.

To determine whether the above split is necessary, the CV method is used as follows. For any set $\mathbb{A}$, the CV is value

$$CV(A) = \sum_{j=1}^q \sum_{X_i \in A} (y_{ij} - \hat{y}_{.j}^i)^2 = \Big(\frac{\#A}{\#A - 1}\Big)^2 \sum_{j=1}^q \sum_{X_i \in A} (y_{ij} - \hat{y}_{.j})^2.$$

If $CV(\mathbb{A}_k^\tau) \leq CV(\mathbb{A}_{k+1}^{\tau'}) + CV(\mathbb{A}_{k+1}^{\tau''})$ then $\mathbb{A}_k^\tau$ is a leave and no longer be divided; otherwise, it will be split and in to $\mathbb{A}_{k+1}^{\tau'}$ and $\mathbb{A}_{k+1}^{\tau''}$.

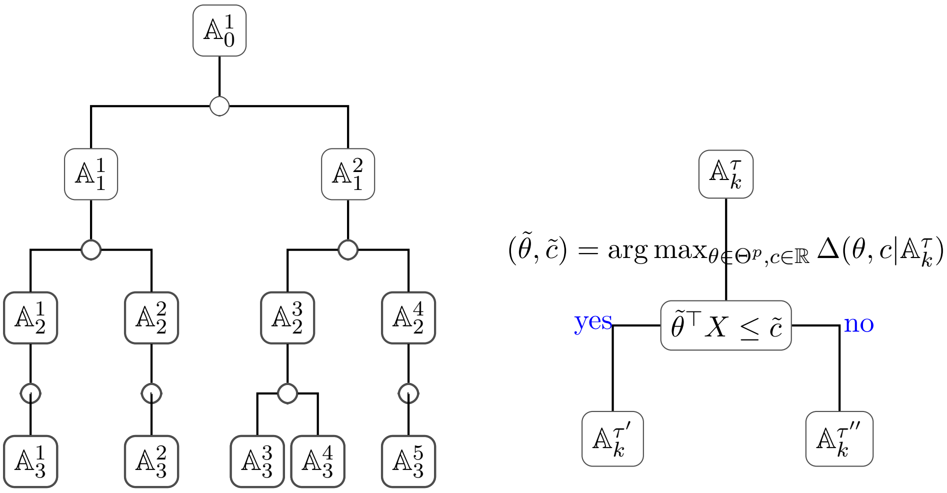
Let $\{\mathbb{A}^j\}_{j=1}^{t_n}$ be all the leaves (or the nodes in the last layer) of above generated tree. Then, $m(x)$ is estimated by

$$m_n(x) = \sum_{j=1}^{t_n} \mathbb{I}(x \in \mathbb{A}^j) \cdot \bar{Y}_{\mathbb{A}_L^j},$$

where

$$\bar{Y}_{\mathbb{A}_L^j} = \sum_{X_i \in \mathbb{A}_L^j} (y_{i1}, ..., y_{iq})/\#\mathbb{A}_L^j$$

is the average of the Y_i in $\mathbb{A}_L^j$. Note that if Y is categorical, $m_n(x)$ is the probability of each class.



Thus, the main step in the ODT or ODRF is the estimation of the projection θ or the selection of the linear combinations. Although many methods have been proposed, we find the projection pursuit regression is still the most efficient and is used in **ODRF**. The estimation is as follows. In any node $\mathbb{A}$, define loss function

$$\Delta(\theta) = \sum_{j=1}^{q} \sum_{X_i \in \mathbb{A}} \{y_{ij} - m_i(\theta^\top X_j)\}^2$$

where m_i is a nonparametric smoothing of the regression that minimizes $\sum_{j=1}^{b} \{y_{ij} - m_i(\theta^\top X_j)\}^2$ with θ given. The nonparametric smoothing can be either the spline representation, or kernel smoothing, or the “supsmu" of **ppr**.

2.2. Build an ODRF

ODRF builds the random forest in a slightly different ways from the existing forests. The detail is as follows. Denote by $X_{[q]} = (x_1', ..., x_q')$ a (random) subset of $X = (x_1, ..., x_p$ where $q < p$ and $\{x_1', ..., x_q'\} \subset \{x_1, ..., x_p\}$, and thus by $X_{[q],r},\ r = 1, 2, ...,$ a sequence of such subsets that may differ from one another. In other words, with the same q and r, set $X_{[q],r}$ changes from place to place. Using the idea of feature bagging, We first define B random ODTs as follows.

- For each node $\mathbb{A}^{\tau}_{k}$, randomly select q, e.g. $INT(p/3)$, find variables $X_{[q],r} = (x'_1, ..., x'_q) \subset X$, where $r = 1, ..., R$ denotes R random sets of variables.
- Find
$$\tilde{\theta}_r = arg \min_{\theta} \sum_{j=1}^{q} \sum_{X_i \in \mathbb{A}} \{y_{ij} - m_j(\theta^{\top} X_{[q],r,i})\}^2$$
where $X_{[q],r,i} = (x'_{i1}, ..., x'_{iq})$.
- Define
$$\mathbb{A}^{\tau'}_{k,r} = \{X_i : X_i \in \mathbb{A}^{\tau}_{k},\ \theta^{\top}_{(q)} X_{[q],r,i} \leq c_{(q)}\},\ \ \mathbb{A}^{\tau''}_{k,r} = \{X_i : X_i \in \mathbb{A}^{\tau}_{k}, \theta^{\top}_{(q)} X_{[q],r,i} > c_{(q)}\},$$
- calculate
$$(\tilde{c}_{(q)}, \tilde{r}) = arg \min_{c, r=1,...,R} \sum_{j=1}^{q} \sum_{X_i \in \mathbb{A}^{\tau'}_{k,r}} (y_{ij} - \bar{y}_{.j})^2 + \sum_{j=1}^{q} \sum_{X_i \in \mathbb{A}^{\tau''}_{k,r}} (y_{ij} - \bar{y}_{.j})^2.$$
- Split the node with $(\tilde{\theta}_{\tilde{r}}, \tilde{c}_{\tilde{r}}, \tilde{r})$, and the daughter nodes
$$\mathbb{A}^{\tau'}_{k+1} = \mathbb{A}^{\tau'}_{k+1,\tilde{r}} \text{ and } \mathbb{A}^{\tau''}_{k+1} = \mathbb{A}^{\tau''}_{k+1,\tilde{r}}.$$
- each tree produces one estimator $\hat{m}_{n,b}(x)$

Finally, the ODR forest (ODRF) estimator is

$$\hat{m}_{ODRF}(x) = B^{-1} \sum_{\tilde{r}=1}^{B} \hat{m}_{n,b}(x).$$

3. Overview of the functions

In this section, we introduce **ODRF** package main function implementation in R. We use some R's S3 method, including the base R functions `print()`, `predict()` and `plot()` in the **base** (R Core Team 2017) package, the transform function `as.part()` in the **partykit** (Hothorn and Zeileis 2015) package and our self-defined functions `ODT()`, `ODRF()`, `online()`, `prune()` in the **ODRF** package. In the **ODRF** package, the function `best.cut.node()` to find the optimal split variables and split nodes and the projection pursuit function `PPO()` to estimate the projection direction. They both make R interact with C++ by the **Rcpp** package, which greatly speeds up the the computation time to our program. The details of the function usage are described below.

3.1. `print` the tree structure of `ODT` and the estimation error of `ODRF`

Functions `ODT()` and `ODRF()` are the two main functions of the **ODRF** package, `ODRF()` is ODT-based random forests. They can both be used for classification and regression and are similar in usage. We provide two data input ways for these two S3 methods.

The first way is `formula = y ~ .,data = data.frame(X, y = y)` or `formula = y ~ X` with class `formula`, and usage are

```
## S3 method for class 'formula'
ODT(formula, data = NULL, type = "auto", NodeRotateFun = "RotMatPPO",
  FunDir = getwd(), paramList =NULL, MaxDepth = Inf, numNode = Inf,
  MinLeaf = 5, Levels = NULL, subset = NULL, weights = NULL,
  na.action = na.fail, catLabel = NULL, Xcat = 0, Xscale = "Min-max",
  TreeRandRotate = FALSE, ...)
ODRF(formula, data = NULL, type = "auto", NodeRotateFun = "RotMatPPO",
  FunDir = getwd(), paramList = NULL, ntrees = 100, storeOOB = TRUE,
  replacement = TRUE, stratify = TRUE, numOOB = 1/3, parallel = TRUE,
  numCores = Inf, seed = 220924, MaxDepth = Inf, numNode = Inf,
  MinLeaf = 5, subset = NULL, weights = NULL, na.action = na.fail,
  catLabel = NULL, Xcat = 0, Xscale = "Min-max", TreeRandRotate = FALSE, ...)
```

The second way is `X = X, y = y` with class `default`, and usage are

```
## Default S3 method:
ODT(X, y, type = "auto", NodeRotateFun = "RotMatPPO", ...)
ODRF(X, y, type = "auto", NodeRotateFun = "RotMatPPO", ...)
```

Arguments

- `type`: The criterion used for splitting the nodes, 'i-classification': information gain and 'g-classification': gini impurity index for classification, 'regression': mean square error for regression. 'auto' (default): If the response in data or y is a factor, 'g-classification' is used, otherwise regression is assumed.(see `?best.cut.node`)
- `NodeRotateFun`: Name of the function of class character that implements a linear combination of predictors in the split node. Default is `"RotMatPPO"` with `model = "PPR"` (see `?RotMatPPO`). Users can define this function, for details see `?RotMatMake`.
- `catLabel`: A category labels of class `list` in predictors. (default NULL, for details see the following examples)
- `other arguments`: Other arguments we do not introduce here, users can see `?ODT` and `?ODRF`. These arguments are defaulted to the optimal values, so that the user does not need to modify them except for special needs.

where `formula` plus `data` is the now standard way of specifying relationships in R. The remaining arguments in the first line (`subset`, `na.action`, and `weights`) are also standard for setting up formula-based models in R.

Before classification or regression, it is necessary to do some pre-processing to the data. In addition to the standard arguments `subset`, `na.action`, and `weights`, we also provide `Xscale` for the predictors to be normalized. Any feature x of predictor X were scaled to $[0,1]$ using the maximal or quantile value of the in-sample, that is $\frac{x-x_L}{x_U-x_L}$. Where x_U and x_L denote the upper and lower bounds of x, respectively. When using minima (defaulte `Xscale = "Min-max"`), $x_U =$ `max(x)`, $x_L =$ `min(x)`, when using quantile (`Xscale = "Quantile"`), x_U $=$ `quantile(x, 0.95)`, $x_L =$ `quantile(x, 0.05)`. Sometimes the predictor X has category variables that must be transformed into dummy variables. The arguments `catLabel` and

Xcat in our program can automatically deal with category variables. The user can enter the ordinal number of the category variable with Xcat and let catLabel = NULL. It even is allowed to let Xcat = NULL, we will use the condition (length(unique(x)) < 10) & (n > 20) to judge which one is the category variable, and for details see examples of ODT or ODRF.Print the tree structure of class ODT and party, and the model estimation error of class ODRF.

```
R> data(iris, package = "datasets")
R> tree <- ODT(Species ~ ., data = iris)
R> tree

============================================================
Oblique Classification Tree structure
============================================================

1) root
   node2)# proj1*X < 0.25 -> (leaf1 = setosa)
   node3)  proj1*X >= 0.25
      node4)  proj2*X < 0.69
         node6)# proj3*X < 0.67 -> (leaf2 = versicolor)
         node7)  proj3*X >= 0.67
            node10)# proj5*X < 0.6 -> (leaf5 = virginica)
            node11)# proj5*X >= 0.6 -> (leaf6 = versicolor)
      node5)  proj2*X >= 0.69
         node8)# proj4*X < 0.65 -> (leaf3 = virginica)
         node9)# proj4*X >= 0.65 -> (leaf4 = virginica)

R> party.tree <- as.party(tree, data = iris)
R> party.tree

Model formula:
Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width

Fitted party:
[1] root
|   [2] proj1X >= 0.24576
|   |   [3] proj2X >= 0.6875
|   |   |   [4] proj4X >= 0.65254: virginica (n = 43, err = 0.0%)
|   |   |   [5] proj4X < 0.65254: virginica (n = 3, err = 33.3%)
|   |   [6] proj2X < 0.6875
|   |   |   [7] proj3X >= 0.66949
|   |   |   |   [8] proj5X >= 0.60417: versicolor (n = 3, err = 33.3%)
|   |   |   |   [9] proj5X < 0.60417: virginica (n = 3, err = 0.0%)
|   |   |   [10] proj3X < 0.66949: versicolor (n = 48, err = 2.1%)
|   [11] proj1X < 0.24576: setosa (n = 50, err = 0.0%)

Number of inner nodes:    5
Number of terminal nodes: 6
```

```
R> forest <- ODRF(Species ~ ., data = iris, parallel = FALSE)
R> forest
```

```
Call:
 ODRF.formula(formula = Species ~ ., data = data, parallel = FALSE)
               Type of oblique decision random forest: classification
                                      Number of trees: 100
                           OOB estimate of error rate: 4.67%
Confusion matrix:
           setosa versicolor virginica class_error
setosa         50          0         0  0.00000000
versicolor      0         46         3  0.06122436
virginica       0          4        47  0.07843122
```

`print()` is R's standard `S3` method used to print the results of various objects of class. We use a similar way to `print()` in **PPtreeViz** package Lee (2018) to print the tree structure of `ODT()`, which shows each node partition of `ODT` in detail. When the options `projection = TRUE, cutvalue = TRUE` denotes to print projection coefficient and cutoff values in each node respectively. Currently, the **partykit** package is commonly used to summarize and vi-sualize tree structure in various ways. The function `as.party()` in **partykit** package can convert the trees in other `R` packages to `party` class. We define the `as.party.ODT()` function to add `ODT` class to the `party` class, so that we can use the same function `print()` to print the tree structure of the `party` class. In addition, we can also use the function `print()` to print the model estimation error for the class `ODRF()`.

3.2. Classification and regression with functions `ODT()` and `ODRF()`

`predict()` is the standard `S3` method used to predict new data for various objects of class. we defined the functions `predict.ODT()` and `predict.ODRF()` to predict `Xnew` for classes `ODT()` and `ODRF()` respectively. The default output of `predict()` is `response` which is the prediced values of the new data. When the argument `leafnode = TRUE` in `predict.ODT()`, outputs the leaf node sequence number that the new data is partitioned, and it can be used for clustering the data. `predict.ODT()` also provides options `type` and `weight.tree` to denote the output type and whether to weight the tree respectively, see `?predict.ODRF` for details.

```
## S3 method for class 'ODT'
predict(ppTree, Xnew, leafnode = FALSE)
## S3 method for class 'ODRF'
predict(ppForest, Xnew, type = "response", weight.tree = FALSE)
```

We also defined `S3` methods `online` and `prune` used to online structure training and estimation error pruning for classes `ODT` and `ODRF`, respectively, and they can significantly improve the model accuracy. `online` Update existing `ODT` and `ODRF` using batches of data. `prune` is judged to prune or not based on whether the error of computing new data is reduced or not, and our pruning begins from the last leaf node. For class `ODRF`, let `prune`'s argument `useOOB=TRUE` to use 'out-of-bag' for pruning.

```
## S3 method for class 'ODT' and 'ODRF'
online(obj, X = NULL, y = NULL, weights = NULL, ...)
## S3 method for class 'ODT'
prune(ppTree, X, y, MaxDepth = 1, ...)
## S3 method for class 'ODRF'
prune(ppForest, X, y, MaxDepth = 1, useOOB = TRUE, ...)
```

Classification and regression with `ODRF()` and `ODT()` respectively, and the model is trained with `online()` and prune with `prune()`, respectively.

```
R> data(seeds, package = "ODRF")
R> set.seed(18)
R> train <- sample(1:209, 120)
R> train_data <- data.frame(seeds[train, ])
R> test_data <- data.frame(seeds[-train, ])
R> index <- seq(floor(nrow(train_data) / 2))
R> forest <- ODRF(varieties_of_wheat ~ ., train_data,
+    type = "i-classification", parallel = FALSE
+  )
R> pred <- predict(forest, test_data[, -8])
R> e.forest <- mean(pred != test_data[, 8])
R> forest1 <- ODRF(varieties_of_wheat ~ ., train_data[index, ],
+    type = "i-classification", parallel = FALSE
+  )
R> pred <- predict(forest1, test_data[, -8])
R> e.forest.1 <- mean(pred != test_data[, 8])
R> forest2 <- ODRF(varieties_of_wheat ~ ., train_data[-index, ],
+    type = "i-classification", parallel = FALSE
+  )
R> pred <- predict(forest2, test_data[, -8])
R> e.forest.2 <- mean(pred != test_data[, 8])
R> forest.online <- online(
+    forest1, train_data[-index, -8],
+    train_data[-index, 8]
+  )
R> pred <- predict(forest.online, test_data[, -8])
R> e.forest.online <- mean(pred != test_data[, 8])
R> forest.prune <- prune(forest1, train_data[-index, -8],
+    train_data[-index, 8],
+    useOOB = FALSE
+  )
R> pred <- predict(forest.prune, test_data[, -8])
R> e.forest.prune <- mean(pred != test_data[, 8])
R> print(c(
+    forest = e.forest, forest1 = e.forest.1, forest2 = e.forest.2,
+    forest.online = e.forest.online, forest.prune = e.forest.prune
+  ))
```

```
      forest       forest1       forest2 forest.online  forest.prune
  0.03370787    0.07865169    0.08988764    0.04494382    0.06741573
```

```
R> data(body_fat, package = "ODRF")
R> set.seed(9)
R> train <- sample(1:252, 120)
R> train_data <- data.frame(body_fat[train, ])
R> test_data <- data.frame(body_fat[-train, ])
R> index <- seq(floor(nrow(train_data) / 2))
R> tree <- ODT(Density ~ ., train_data, type = "regression")
R> pred <- predict(tree, test_data[, -1])
R> e.tree <- mean((pred - test_data[, 1])^2)
R> tree1 <- ODT(Density ~ ., train_data[index, ], type = "regression")
R> pred <- predict(tree1, test_data[, -1])
R> e.tree.1 <- mean((pred - test_data[, 1])^2)
R> tree2 <- ODT(Density ~ ., train_data[-index, ], type = "regression")
R> pred <- predict(tree2, test_data[, -1])
R> e.tree.2 <- mean((pred - test_data[, 1])^2)
R> tree.online <- online(tree1, train_data[-index, -1], train_data[-index, 1])
R> pred <- predict(tree.online, test_data[, -1])
R> e.tree.online <- mean((pred - test_data[, 1])^2)
R> tree.prune <- prune(tree1, train_data[-index, -1], train_data[-index, 1])
R> pred <- predict(tree.prune, test_data[, -1])
R> e.tree.prune <- mean((pred - test_data[, 1])^2)
R> print(c(
+    tree = e.tree, tree1 = e.tree.1, tree2 = e.tree.2,
+    tree.online = e.tree.online, tree.prune = e.tree.prune
+  ))
```

```
        tree        tree1        tree2  tree.online   tree.prune
2.730683e-05 4.576357e-05 4.330298e-05 4.191764e-05 3.595299e-05
```

As shown in the classification and regression results above, the training data `train_data` is divided into two batches equally, then the first batch is used to train `ODT` and `ODRF`, and the second batch is used to update the model by `online()`. The error after the model update is significantly smaller than that of one batch of data alone, and the model is also pruned by `prune()` and the same effect is achieved.

3.3. Create a projection matrix with the `RotMat*` function

We provide the functions `RotMatPPO()`, `RotMatRand()` and `RotMatRF()` for creating rotation matrix by projection pursuit optimization model (*PPO*) Cook, Buja, Lee, and Wickham (2008), same random rotation as `RandMatBinary()` in **rerf** package and single feature similar to random forest, respectively. The function `PPO()` is to find the best projection using various projectin pursuit models, including `"PPR"` (default): projection projection regression from `ppr()` in **stats** package Friedman and Stuetzle (1981), `"Log"`: logistic based on `nnet()` in **nnet** package Venables and Ripley (2002), `"Rand"`: The random projection generated from $\{-1, 1\}$, argument PPmethod of function PPopt() in **PPtreeViz** package, and argument findex of function PP_Optimizer() in **Pursuit** package Ossani and Cirillo (2022). Note that **PPtreeViz** and **Pursuit** are only available for classification. The generated rotation matrix has three columns, the first column (Variable). Variable to be projected, the second column (Number): Number of projections, and the third column (Coefficient): the coefficient of the projected matrix. In addition, the user can define a projection matrix function, see the **ODRF** help file for more details on usage.

```
RotMatPPO(X,y,model = "PPR",type = "i-classification",weights = NULL,
  dimProj,numProj,catLabel = NULL, ...)
RotMatRand(dimX,randDist = "Binary",numProj = ceiling(sqrt(dimX)),
  dimProj = "Rand",sparsity, prob = 0.5,lambda = 1,catLabel = NULL,...)
RotMatRF(dimX, numProj, catLabel = NULL, ...)
RotMatMake(X = NULL,y = NULL,RotMatFun = "RotMatPPO",PPFun = "PPO",
  FunDir = getwd(),paramList = NULL, ...)
PPO(X, y, model = "PPR", type = "i-classification", weights = NULL
```

To show that PPO() with different model to do classification and regression and used to RotMatPPO(). after that show simple usage of functions RotMatRand() and RotMatRF().

```
R> data(seeds, package = "ODRF")
R> (PP <- PPO(seeds[, 1:7], seeds[, 8], model = "LDA", type = "i-classification"))

[1]  0.541389011  0.147840867 -0.141470331  0.490670015  0.370939106
[6]  0.003959901 -0.535405078

R> RotMat <- RotMatPPO(seeds[, 1:7], seeds[, 8],
+    model = "Log",
+    type = "i-classification"
+  )
R> head(RotMat)

     Variable Number Coefficient
[1,]        2      1   1.0000000
[2,]        7      2   1.0000000
[3,]        4      3   1.0000000
[4,]        5      4  -0.3558198
[5,]        3      4   0.9345546
[6,]        4      5   1.0000000

R> data(body_fat, package = "ODRF")
R> (PP <- PPO(body_fat[, 2:15], body_fat[, 1], model = "Log", type = "regression"))

 [1] -0.2019004 -0.4171691  0.6436646 -0.3221755 -0.4999429  0.1583641
 [7]  0.2958124 -0.2856300  0.2391311  0.6917452  0.5816309 -0.5984203
[13] -0.1813335 -0.5909681
```

```
R> RotMat <- RotMatPPO(seeds[, 1:7], seeds[, 8],
+    model = "PPR",
+    type = "i-classification"
+  )
R> head(RotMat)

     Variable Number Coefficient
[1,]        3      1  1.00000000
[2,]        1      2  1.00000000
[3,]        6      3  1.00000000
[4,]        6      4  0.01858663
[5,]        7      4  0.99982725
[6,]        2      5  1.00000000

R> set.seed(22)
R> X <- matrix(rnorm(1000), 100, 10)
R> y <- (rnorm(100) > 0) + 0
R> paramList <- list(dimX = 8, numProj = 3, sparsity = 0.25, prob = 0.5)
R> (RotMat <- do.call(RotMatRand, paramList))

     Variable Number Coefficient
[1,]        7      1           1
[2,]        4      2          -1
[3,]        5      2           1
[4,]        6      2           1
[5,]        7      2           1
[6,]        8      3          -1

R> paramList <- list(dimX = 8, numProj = 3, catLabel = NULL)
R> (RotMat <- do.call(RotMatRF, paramList))

     Variable Number Coefficient
[1,]        6      1           1
[2,]        7      2           1
[3,]        5      3           1
```

4. Real examples

In this section, we compare ODT with other axis-aligned and oblique tree methods in a more rigorous benchmark comparison. the tree algorithms are compared on 52 real data sets with continuous and categorical response. In addition, we add the forest methods and consider the time consumption and tree complexity. The rest of the section describes how to use our ODRF package in real data, and we explain the use of two data sets with continuous and categorical response.

4.1. Performance comparison

we follow the experimental design of Zhan *et al.* (2022), where we show the performance of `ODRF`, but in this article we show the performance of `ODT`. We use 26 real data sets with continuous responses and 26 data sets with categorical response for regression prediction and classification respectively. Where the data set has categorical responses, seeds, breast tissue and waveforms dataset are triple, six and triple category responses respectively, and all other data are binary category responses (0 and 1). Our data are mainly obtained from the UCI machine learning database (A) `https://archive.ics.uci.edu/ml/datasets`, the kaggle database (B) `https://www.kaggle.com`, and Rainforth and Wood Rainforth and Wood (2015) Collection Datasets (C) `https://github.com/twgr/ccfs/`. If there are any missing values in a data, the corresponding samples are removed from the data. In the calculation, Each predictor is scaled to [0, 1] using the minima method of Section **??** proposed.

We compare two different random rotation oblique tree methods, Blaser and FryzlewiczBlaser and Fryzlewicz (2016) proposed the Random Rotation Random Forest (*RotRF*), and Tomita et al.Tomita *et al.* (2020) proposed Sparse Projection Oblique Randomer Forests (*SPORF*). Note that, SPORF unlike RotRF, the random rotation in SPORF is carried out at separately each node, as opposed to using a single rotation for the whole tree. We use single trees and denote as *RotT* and *SPOT* respectively, and we use function `ODT()` in **ODRF** package to implement RotT and SPOT. In addition, there are two oblique decision tree methods used for classification, projection pursuit classification trees (PPT) Lee (2018) with function `PPTreeclass()` in **PPtreeViz** package and Oblique Trees for Classification Data (OT) Truong (2009) with function `oblique.tree()` in **oblique.tree** package. To show the performance of ODT, we also compare four axis-aligned tree methods in Section 1, including *CART* with function `rpart()` in **rpart** package, *ERT* with function `RLT()` in **RLT** package, *EVT* with function `evtree()` in **evtree** package, and *CT* with function `ctree()` in **partykit** package. For all the R functions and packages, their default values of tuning parameters are used. Note that because *PPT* and *OT* cannot do the regression, we only report the classification results. In order to improve speed, this portion of code was implemented in C++ and integrated into R using the Rcpp package. Further speedup is achieved through multicore parallelization of tree construction and byte-compilation via the R compiler package.

For each data set, we randomly partition it into training set and test set. The training set consists of $n = \min(\lfloor 2N/3 \rfloor, 2000)$ randomly selected observations, where N is the number of observations in the original data sets, and the remaining observations form the test set. For regression, the relative prediction error, defined as

$$RPE = \sum_{i \in \text{test set}} (\hat{y}_i - y_i)^2 / \sum_{i \in \text{test set}} (\bar{y}_{\text{train}} - y_i)^2,$$

where $\bar{y}_{\text{train}}$ is naive predictions based on the average of y in the training sets, is used to measure the performance of a method. For classification, the misclassification rate, defined as

$$MR = \sum_{i \in \text{test set}} 1(\hat{y}_i \neq y_i)/(N - n),$$

is used to measure the performance. For each data set, the random partition is repeated 100 times, and averages of the RPEs or MRs are calculated to compare different methods. The calculation results are listed in Table 1 and Table 2. The smallest RPE or MR for each data set is highlighted in **bold** font.

			Axis-aligned				Oblique		
Dataset	n	p	CART	ERT	EVT	CT	RotT	SPOT	ODT
Servo (C)	166	4	0.298	0.871	**0.246**	0.300	0.673	0.406	0.256
Auto MPG (C)	391	7	0.213	0.318	0.221	0.192	0.237	0.235	**0.185**
Concrete Compressive Strength (A)	1030	8	0.314	0.501	0.307	0.248	0.453	0.357	**0.189**
Boston house price (A)	506	13	0.275	0.444	0.293	0.275	0.427	0.358	**0.256**
Wild blueberry yield (B)	777	13	0.228	0.329	0.225	0.189	0.262	0.199	**0.098**
Body fat (B)	252	14	0.073	0.442	0.084	0.061	0.401	0.353	**0.059**
Paris housing price (B)	10000	16	0.018	0.284	0.004	**0.000**	0.681	0.435	**0.000**
House sales in King County (B)	21613	18	0.326	0.425	0.292	0.256	0.388	0.314	**0.254**
Bar crawl (B)	7590	21	0.318	0.416	0.303	**0.268**	0.464	0.370	**0.268**
Auto 93 (C)	81	22	0.620	0.967	0.697	0.602	0.778	0.702	**0.543**
Auto horsepower (C)	159	24	0.236	0.347	0.251	0.238	0.448	0.325	**0.186**
Wave Energy Converters (A)	71998	32	0.543	0.717	0.557	**0.488**	0.544	0.534	0.547
Sidney house price (B)	30000	37	0.703	1.040	0.705	**0.646**	0.803	0.737	0.720
Facebook comment volume (A)	18370	52	0.630	1.134	0.676	**0.620**	0.985	0.843	0.708
Baseball player statistics (B)	4535	74	0.039	0.162	0.013	0.008	0.631	0.433	**0.003**
Gold price prediction (B)	1718	74	0.042	0.020	0.021	0.014	0.109	0.021	**0.010**
CNNpred (A)	1441	76	0.032	0.016	0.011	0.003	0.119	0.065	**0.002**
Warsaw flat rent price (B)	3472	78	0.433	0.678	0.424	**0.398**	1.019	0.565	0.498
Superconductivity (A)	21263	81	0.284	0.319	0.263	0.241	0.280	0.253	**0.240**
Buzz in social media (A)	28179	96	0.140	0.252	0.198	**0.126**	0.292	0.219	0.196
Communities and Crime (A)	1994	101	0.460	0.720	0.467	**0.434**	0.604	0.561	0.614
Residential building-Sales (A)	372	103	0.060	0.243	0.060	**0.040**	0.330	0.267	**0.040**
Residential building-Cost (A)	372	103	0.111	0.212	0.129	0.094	0.241	0.204	**0.086**
Credit score (B)	80000	259	0.274	0.299	0.269	**0.234**	0.490	0.267	0.236
CT slices (A)	53500	380	0.224	0.209	0.238	0.187	0.374	0.225	**0.154**
UJIndoor-Longitude (A)	19937	465	0.087	0.101	0.140	0.064	0.198	0.054	**0.026**
Average RPE(%) across all data sets			0.269	0.441	0.273	0.239	0.470	0.358	0.245
no. of bests in 26 datasets			0	0	1	10	0	0	18

Table 1: Regression: average RPE based on 100 random partitions of each data set into training and test sets

By comparing the prediction errors, both the RPE of regression and MR of classification, our ODT is generally smaller than other methods. ODT is quite stable and attains the smallest RPE and MR in most data sets as indicated in Table 1 and Table 2. The competency of ODT is also verified by the fact that it has the smallest average of RPEs (or MSs) across all the data sets amongst all the methods. *no. of bests* denotes the number of datasets in which a method performs best among all competitors. This suggests that ODT outperforms other methods for regression in most datasets, and PPT outperforms other methods for classification in most datasets. However, the Average RPE of PPT is lower than that of ODT because PPT performs especially bad on several datasets such as Financial indicators and Hill valley, i.e., ODT performs more consistently than PPT.

Next, we compare the performance of ODT, ODRF and the competitors for classification

			Axis-aligned				Oblique				
Dataset	n	p	CART	ERT	EVT	CT	RotT	SPOT	PPT	OT	ODT
Seeds (C)	210	7	9.66	95.59	10.89	12.39	10.87	11.56	**4.27**	5.30	7.21
Breast tissue (C)	106	9	36.25	91.90	38.78	42.62	41.90	39.94	**33.63**	39.14	36.67
MAGIC Gamma telescope (A)	19020	10	17.83	25.17	18.44	**17.79**	22.74	21.73	20.69	18.88	19.84
Indian liver patient (A)	579	10	32.01	33.67	30.56	**29.64**	34.46	32.91	37.30	33.49	33.30
Heart disease (A)	270	13	21.91	27.20	23.18	25.28	27.20	25.91	**16.20**	23.61	24.09
EEG eye state (A)	14980	14	29.34	33.18	28.92	32.38	28.81	26.55	37.75	23.89	**22.30**
seismic-bumps (A)	2584	15	7.05	9.17	**6.56**	6.62	9.50	9.94	18.31	10.18	10.54
Retinopathy debrecen (A)	1151	19	35.89	40.03	35.48	37.03	39.13	36.59	**29.37**	32.66	31.84
Waveform (C)	5000	21	26.36	91.06	25.29	24.72	26.63	26.57	21.95	**19.08**	20.24
Parkinson multiple sound (B)	1208	26	**34.39**	38.82	34.89	37.10	36.44	35.50	35.23	36.88	35.18
Pistachio (B)	2148	28	13.62	17.29	13.75	13.67	16.41	15.63	**11.55**	11.75	12.52
Breast cancer (B)	569	30	7.07	9.37	6.74	6.34	7.66	7.06	**4.41**	6.16	5.32
Ionosphere (A)	351	33	12.40	17.84	11.52	**10.22**	15.27	13.56	13.92	16.26	12.40
QSAR biodegradation (A)	1055	41	17.88	20.84	18.30	19.65	20.60	19.41	**15.52**	19.60	18.26
Spambase (A)	4601	57	10.57	11.27	9.82	10.49	16.49	10.96	9.75	10.14	**8.88**
Mice protein expression (A)	1047	70	13.60	17.63	16.87	13.93	16.42	14.04	**3.58**	4.62	5.70
Ozone level detection (A)	1847	72	8.02	9.94	**6.89**	7.51	9.47	9.40	19.55	11.53	9.99
Company bankruptcy (B)	6819	94	3.82	4.71	**3.25**	3.44	5.16	4.65	17.13	7.02	5.41
Hill valley noisy (C)	1212	100	47.52	75.83	49.07	51.75	40.61	38.54	33.63	20.95	**18.27**
Hill valley (C)	1212	100	48.22	46.38	48.29	51.41	30.87	10.23	29.90	13.95	**0.04**
Musk (A)	6598	166	6.82	9.87	10.03	7.46	10.01	8.40	**6.75**	8.55	8.32
ECG heartbeat (B)	14550	186	14.72	17.47	18.97	16.64	21.56	**14.71**	25.11	24.99	15.34
Arrhythmia (A)	420	192	25.14	36.96	**22.84**	32.18	37.21	33.53	38.04	40.24	31.74
Financial indicators (B)	986	216	**0.05**	17.31	0.19	0.17	23.69	13.93	48.56	**0.05**	1.10
Madelon (A)	2000	500	**23.43**	44.70	33.24	33.17	48.89	47.72	46.54	48.58	43.86
Human activity recognition (A)	2633	561	**0.00**	0.33	0.22	0.05	0.32	0.20	**0.00**	**0.00**	0.08
Average MR(%) across all data sets			19.37	32.44	20.11	20.91	23.01	20.35	22.26	18.75	16.86
no. of bests in 26 datasets			4	0	4	3	0	1	10	3	4

Table 2: Classification: average MR (%) based on 100 random partitions of each data set into training and test sets

and regression in three aspects, including prediction error (MR or RPR), time consumption (Time) and the number of terminal nodes (Complexity) for the tree method only. We still use the above dataset, but the other oblique tree and forest related R packages cannot be used for regression, **rotationForest** package can only be used for binary classification, and the **PPforest** package has an error and cannot be calculated for 5 binary classification datasets such as Company bankruptcy and MAGIC Gamma telescope. So Table 3 shows the average based on 23 and 18 binary classification datasets for tree and forest motheds respectively, and 26 regression datasets. To be fair, all methods are implemented with own R package. Specifically, RotT with function `rotationForest()` in **rotationForest** package, RotT and RotRF with function `rotationForest()` in **rotationForest** package, *SPOT* and *SPORF* with function `RerF()` in **rerf** package, *RF* with function `randomForest()` in **randomForest** package,

		Classification			Regression		
Method		MR (%)	Time	Complexity	RPE	Time	Complexity
Axis-aligned tree	CART	18.86 (2)	0.22 (1)	10.83 (0)	0.271 (3)	0.11 (3)	8.58 (17)
	ERT	25.71 (0)	0.02 (21)	107.61 (0)	0.443 (0)	0.01 (23)	307.50 (0)
	EVT	19.95 (5)	0.75 (0)	6.48 (2)	0.276 (1)	1.67 (0)	11.62 (9)
	CT	19.42 (3)	0.29 (0)	10.43 (2)	0.233 (10)	0.85 (0)	36.38 (0)
Oblique tree	RotT	18.92 (2)	0.47 (0)	11.00 (1)	-	-	-
	SPOT	20.74 (0)	0.05 (1)	83.00 (0)	-	-	-
	PPT	22.47 (1)	0.27 (0)	2.00 (16)	-	-	-
	OT	18.72 (2)	30.00 (0)	24.61 (1)	-	-	-
	ODT	15.85 (5)	0.24 (0)	36.87 (1)	0.243 (12)	0.61 (0)	222.00 (0)
Axis-aligned forest	RF	15.63 (0)	1.84 (1)	-	0.155 (3)	12.80 (0)	-
	GRF	18.33 (0)	0.52 (13)	-	0.196 (0)	0.38 (26)	-
	ERT	17.95 (1)	1.69 (1)	-	0.187 (1)	4.90 (0)	-
	XGB	17.91 (0)	0.78 (3)	-	0.144 (8)	1.89 (0)	-
Oblique forest	RotRF	17.16 (2)	154.60 (0)	-	-	-	-
	SPORF	11.01 (3)	29.93 (0)	-	-	-	-
	PPF	20.61 (1)	13.61 (0)	-	-	-	-
	ORF	10.38 (1)	140.21 (0)	-	-	-	-
	ODRF	9.86 (10)	107.21 (0)	-	0.143 (14)	638.53 (0)	-

Table 3: Performance comparison of different methods for classification and regression. () denotes the number of datasets in which a method performs best among all competitors, and - denotes the method is not supported and no calculation result is available.

GRF with functions `regression_forest()` and `Classification_forest()` in package **grf**, *XGB* with function `xgboost()` in **xgboost** package, *ORF* with function `obliqueRF()` in **obliqueRF** package and *PPF* with function `PPforest()` in **PPforest** package. The details of these methods are shown in Section 1. As shown by Table3, our ODT and ODRF are generally smaller than other methods, both in terms of the average RPE of regression and the average MR of classification. For the compare of tree method, OT has the greatest average time and the average Complexity of PPT is only 2, ERT has the greatest average Complexity but the smallest average time, while ODT has the common performance in these two aspects. For the comparison of the forest methods, our ODRF is significantly improved relative to ODT, but the average Time also increases much more, especially for the regression. Nevertheless ODRF still outperforms RotRF and ORF. In a word, ODT has outstanding performance in both prediction error and time consumption, while ODRF can significantly improve prediction error but is more time consuming. Compared with other oblique tree or forest R packages, our ODRF package has higher prediction accuracy and equal time consumption, and our ODRF package is more complete.

4.2. Wisconsin Breast Cancer Database

Breast cancer is the most common cancer amongst women in the world. or 25% of all cancer cases, and affected over 2.1 Million people in 2015 alone. It starts when cells in the breast

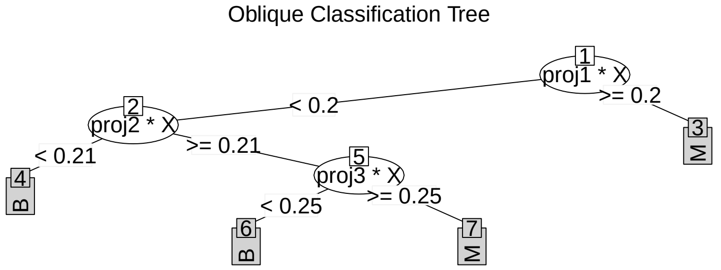


Figure 1: The ODT tree structure of breast cancer

begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area. This data was obtained from the ODRF package by codedata(breast_cancer,package = "ODRF"). The predictors contain three cell nuclei, mean, se and worst, and each cell nucleus accounts for ten real-valued features of radius, texture, perimeter, area, smoothness, compactness, depression, dimple, symmetry and fractal dimension. The key challenges against it's detection is how to classify tumors into malignant (M,cancerous) or benign(B,noncancerous).

```
==============================================================
Oblique Classification Tree structure
==============================================================

1) root
   node2)  proj1*X < 0.2
      node4)# proj2*X < 0.21 -> (leaf2 = B)
      node5)  proj2*X >= 0.21
         node6)# proj3*X < 0.25 -> (leaf3 = B)
         node7)# proj3*X >= 0.25 -> (leaf4 = M)
   node3)# proj1*X >= 0.2 -> (leaf1 = M)
```

We use the function `ODT()` to construct a decision tree, and print and plot the tree structure. It is shown that the tree structure is very simple with 3 split nodes, 4 leaf nodes and the depth of 4. In addition, we can obtain the projection coefficients of the variables at each split, and the results are shown in Table 2 after removing the variables with projection coefficients less than 0.1 from the 30 variables. It shows that perimeter (x3), concave points (x8, x28), area (x14) and radius (x21) play a major role in the first partition. compactness (x6), perimeter (x13) and concave points (x28) have a significant impact on the second partitioning. In the third partitioning, perimeter_worst (x23) was randomly selected as the partitioning variable from 30 variable according to our procedure due to the small amount of data. We noticed that mean cell nuclei had the most influence in the whole decision tree.

References

varibles	proj1	proj2	proj3
x3: perimeter_mean	-0.434	0.000	0.000
x6: compactness_mean	0.000	-0.571	0.000
x8: concave.points_mean	-0.206	0.000	0.000
x9: symmetry_mean	0.000	-0.275	0.000
x13: perimeter_se	0.000	0.413	0.000
x14: area_se	0.608	0.000	0.000
x21: radius_worst	0.486	0.000	0.000
x22: texture_worst	0.000	0.175	0.000
x23: perimeter_worst	0.000	0.000	1.000
x24: area_worst	0.000	0.153	0.000
x28: concave.points_worst	0.400	0.612	0.000

Table 4: the projections of breast cancer

Athey S, Tibshirani J, Wager S (2019). "Generalized random forests." *The Annals of Statistics*, **47**(2), 1148–1178.

Blaser R, Fryzlewicz P (2016). "Random rotation ensembles." *The Journal of Machine Learning Research*, **17**(1), 126–151.

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

Chen T, Guestrin C (2016). "**Xgboost**: A scalable tree boosting system." In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794.

Cook D, Buja A, Lee EK, Wickham H (2008). "Grand tours, projection pursuit guided tours, and manual controls." In *Handbook of data visualization*, pp. 295–314. Springer.

Fokkema M, Edbrooke-Childs J, Wolpert M (2020). "Generalized linear mixed-model (GLMM) trees: A flexible decision-tree method for multilevel and longitudinal data." *Psychotherapy Research*, (22), 1–13.

Friedman JH, Stuetzle W (1981). "Projection pursuit regression." *Journal of the American statistical Association*, **76**(376), 817–823.

Geurts P, Ernst D, Wehenkel L (2006). "Extremely randomized trees." *Machine Learning*, **63**(1), 3–42.

Grinsztajn L, Oyallon E, Varoquaux G (2022). "Why do tree-based models still outperform deep learning on tabular data?" `doi:10.48550/ARXIV.2207.08815`. URL `https://arxiv.org/abs/2207.08815`.

Grubinger, Zeileis, Pfeiffer, KP (2014). "**evtree**: Evolutionary Learning of Globally Optimal Classification and Regression Trees in R." *J STAT SOFTW*.

Heath D, Kasif S, Salzberg S (1993). "Induction of Oblique Decision Trees." *Journal of Artificial Intelligence Research*, **2**(2), 1–32.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased recursive partitioning: A conditional inference framework." *Journal of Computational and Graphical statistics*, **15**(3), 651–674.

Hothorn T, Zeileis A (2015). "**Partykit**: a modular toolkit for recursive partytioning in R." *The Journal of Machine Learning Research*, (1).

Johnson R, Tong Z (2014). "Learning Nonlinear Functions Using Regularized Greedy Forest." *IEEE Trans Pattern Anal Mach Intell*, **36**(5), 942–954.

Katuwal R, Suganthan PN, Zhang L (2020). "Heterogeneous oblique random forest." *Pattern Recognition*, **99**, 107078.

Lee D, Yang MH, Oh S (2015). "Fast and accurate head pose estimation via random projection forests." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1958–1966.

Lee EK (2018). "**PPtreeViz**: An R package for visualizing projection pursuit classification trees." *Journal of Statistical Software*, **83**, 1–30.

Maia M, Murphy K, Parnell AC (2022). "**GP-BART**: a novel Bayesian additive regression trees approach using Gaussian processes." *arXiv preprint arXiv:2204.02112*.

Menze BH, Kelm BM, Splitthoff DN, Koethe U, Hamprecht FA (2011). "On oblique random forests." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 453–469. Springer.

Ossani PC, Cirillo MA (2022). *Projection Pursuit*. R package version 1.0.3, URL https://CRAN.R-project.org/package=Pursuit.

Quinlan JR (1987). "Decision trees as probabilistic classifiers." In *Proceedings of the Fourth International Workshop on Machine Learning*, pp. 31–37. Elsevier.

Quinlan JR (1993). "Program for machine learning." C4. 5.

Rainforth T, Wood F (2015). "Canonical correlation forests." *arXiv preprint arXiv:1507.05444*.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Silva Nd, Cook D, Lee EK (2021). "A projection pursuit forest algorithm for supervised classification." *Journal of Computational and Graphical Statistics*, **30**(4), 1168–1180.

Tomita TM, Browne J, Shen C, Chung J, Patsolic JL, Falk B, Priebe CE, Yim J, Burns R, Maggioni M, *et al.* (2020). "Sparse projection oblique randomer forests." *Journal of machine learning research*, **21**(104).

Truong AKY (2009). *Fast growing and interpretable oblique trees via logistic regression models*. Ph.D. thesis, Oxford University, UK.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL https://www.stats.ox.ac.uk/pub/MASS4/.

Zeileis A, Hothorn T (2015). "Parties, Models, Mobsters: A New Implementation of Model-Based Recursive Partitioning in R."

Zhan H, Liu Y, Xia Y (2022). "Consistency of The Oblique Decision Tree and Its Random Forest." *arXiv preprint arXiv:2211.12653*.

Affiliation:

Yu Liu
Department of Statistics and Data Science
National University of Singapore, Singapore
E-mail: `liuyuchina123@gmail.com`

Yingcun Xia
Department of Statistics and Data Science
National University of Singapore, Singapore
E-mail: `staxyc@nus.edu.sg`