Package 'rerf'

December 4, 2018

Type Package

Title Randomer Forest

Version 2.0.2

Date 2018-12-03

Description R-RerF (aka Randomer Forest (RerF) or Random Projection Forests) is an algorithm developed by Tomita (2016) <arXiv:1506.03410v2> which is similar to Random Forest - Random Combination (Forest-RC) developed by Breiman (2001) <doi:10.1023/A:1010933404324>. Random Forests create axis-parallel, or orthogonal trees. That is, the feature space is recursively split along directions parallel to the axes of the feature space. Thus, in cases in which the classes seem inseparable along any single dimension, Random Forests may be suboptimal. To address this, Breiman also proposed and characterized Forest-RC, which uses linear combinations of coordinates rather than individual coordinates, to split along. This package, 'rerf', implements RerF which is similar to Forest-RC. The difference between the two algorithms is where the random linear combinations occur: Forest-RC combines features at the per tree level whereas RerF takes linear combinations of coordinates at every node in the tree.

Depends R (>= 3.3.0)

License Apache License 2.0 | file LICENSE

URL https://github.com/neurodata/R-RerF

BugReports https://github.com/neurodata/R-RerF/issues

Imports parallel, RcppZiggurat, utils, stats, dummies

Suggests roxygen2 (>= 5.0.0), testthat

LinkingTo Rcpp, RcppArmadillo

SystemRequirements GNU make

ByteCompile true

RoxygenNote 6.1.0

NeedsCompilation yes

Author Jesse Patsolic [ctb, cre],
Benjamin Falk [ctb],
Jaewon Chung [ctb],
James Browne [aut],
Tyler Tomita [aut],
Joshua Vogelstein [ths]

Maintainer Jesse Patsolic <software@neurodata.io>

Repository CRAN

Date/Publication 2018-12-04 09:40:17 UTC

R topics documented:

BuildTree . . . 3
checkInputMatrix . . . 4
ComputeSimilarity . . . 4
FeatureImportance . . . 5
GrowUnsupervisedForest . . . 6
makeAB . . . 6
mnist . . . 7
OOBPredict . . . 8
PackForest . . . 9
PackPredict . . . 9
Predict . . . 10
RandMatBinary . . . 11
RandMatContinuous . . . 12
RandMatCustom . . . 13
RandMatFRC . . . 13
RandMatFRCN . . . 14
RandMatImageControl . . . 15
RandMatImagePatch . . . 16
RandMatPoisson . . . 17
RandMatRF . . . 17
RandMatTSpatch . . . 18
RerF . . . 19
RunFeatureImportance . . . 22
RunOOB . . . 22
RunPredict . . . 23
RunPredictLeaf . . . 23
StrCorr . . . 24
TwoMeansCut . . . 24
Urerf . . . 25

Index **26**

BuildTree *RerF Tree Generator*

Description

Creates a single decision tree based on an input matrix and class vector. This is the function used by rerf to generate trees.

Usage

```
BuildTree(X, Y, FUN, paramList, min.parent, max.depth, bagging,
  replacement, stratify, class.ind, class.ct, store.oob, store.impurity,
  progress, rotate)
```

Arguments

X	an n by d numeric matrix (preferable) or data frame. The rows correspond to observations and columns correspond to features.
Y	an n length vector of class labels. Class labels must be integer or numeric and be within the range 1 to the number of classes.
FUN	a function that creates the random projection matrix.
paramList	parameters in a named list to be used by FUN. If left unchanged, default values will be populated, see defaults for details.
min.parent	the minimum splittable node size. A node size < min.parent will be a leaf node. (min.parent = 6)
max.depth	the longest allowable distance from the root of a tree to a leaf node (i.e. the maximum allowed height for a tree). If max.depth=0, the tree will be allowed to grow without bound.
bagging	a non-zero value means a random sample of X will be used during tree creation. If replacement = FALSE the bagging value determines the percentage of samples to leave out-of-bag. If replacement = TRUE the non-zero bagging value is ignored.
replacement	if TRUE then n samples are chosen, with replacement, from X.
stratify	if TRUE then class sample proportions are maintained during the random sampling. Ignored if replacement = FALSE.
class.ind	a vector of lists. Each list holds the indexes of its respective class (e.g. list 1 contains the index of each class 1 sample).
class.ct	a cumulative sum of class counts.
store.oob	if TRUE then the samples omitted during the creation of a tree are stored as part of the tree. This is required to run OOBPredict().
store.impurity	if TRUE then the reduction in Gini impurity is stored for every split. This is required to run FeatureImportance().
progress	if true a pipe is printed after each tree is created. This is useful for large datasets.
rotate	if TRUE then the data matrix X is uniformly randomly rotated.

Value

Tree

Examples

```
x <- iris[, -5]
y <- as.numeric(iris[, 5])
# BuildTree(x, y, RandMatBinary, p = 4, d = 4, rho = 0.25, prob = 0.5)
```

`checkInputMatrix` *Determine if given input can be processed by Urerf.*

Description

Determine if given input can be processed by Urerf.

Usage

```
checkInputMatrix(X)
```

Arguments

`X`	an Nxd matrix or Data frame of numeric values.

Value

stops function execution and outputs error if invalid input is detected.

`ComputeSimilarity` *Compute Similarities*

Description

Computes pairwise similarities between observations. The similarity between two points is defined as the fraction of trees such that two points fall into the same leaf node.

Usage

```
ComputeSimilarity(X, forest, num.cores = 0L, Xtrain = NULL)
```

Arguments

X	an n sample by d feature matrix (preferable) or data frame which was used to train the provided forest.
forest	a forest trained using the rerf function, with COOB=TRUE.
num.cores	the number of cores to use while training. If num.cores=0 then 1 less than the number of cores reported by the OS are used. (num.cores=0)
Xtrain	an n by d numeric matrix (preferable) or data frame. This should be the same data matrix/frame used to train the forest, and is only required if RerF was called with rank.transform = TRUE. (Xtrain=NULL)

Value

similarity a normalized n by n matrix of pairwise similarities

Examples

```
library(rerf)
X <- as.matrix(iris[, 1:4])
Y <- iris[[5L]]
forest <- RerF(X, Y, num.cores = 1L)
sim.matrix <- ComputeSimilarity(X, forest, num.cores = 1L)
```

FeatureImportance *Compute Feature Importance of a RerF model*

Description

Computes feature importance of every unique feature used to make a split in the RerF model.

Usage

```
FeatureImportance(forest, num.cores = 0L)
```

Arguments

forest	a forest trained using the RerF function with argument store.impurity = TRUE
num.cores	number of cores to use. If num.cores = 0, then 1 less than the number of cores reported by the OS are used. (num.cores = 0)

Value

feature.imp

Examples

```
library(rerf)
forest <- RerF(as.matrix(iris[, 1:4]), iris[[5L]], num.cores = 1L, store.impurity = TRUE)
feature.imp <- FeatureImportance(forest, num.cores = 1L)
```

GrowUnsupervisedForest *Creates Urerf Tree.*

Description

Creates Urerf Tree.

Usage

```
GrowUnsupervisedForest(X, MinParent = 1, trees = 100, MaxDepth = Inf,
  bagging = 0.2, replacement = TRUE, FUN = makeAB, options = list(p
  = ncol(X), d = ceiling(ncol(X)^0.5), sparsity = 1/ncol(X)),
  Progress = TRUE)
```

Arguments

X	an Nxd matrix or Data frame of numeric values.
MinParent	the minimum splittable node size (MinParent=1).
trees	the number of trees to grow in a forest (trees=100).
MaxDepth	the maximum depth allowed in a forest (MaxDepth=Inf).
bagging	only used experimentally. Determines the hold out size if replacement=FALSE (bagging=.2).
replacement	method used to determine boot strap samples (replacement=TRUE).
FUN	the function to create the rotation matrix used to determine mtry features.
options	options provided to FUN.
Progress	logical that determines whether to show tree creation status (Progress=TRUE).

Value

tree

makeAB *Create rotation matrix used to determine mtry features.*

Description

This function is the default option to make the projection matrix for unsupervised random forest. The sparseM matrix is the projection matrix. The creation of this matrix can be changed, but the nrow of sparseM should remain p. The ncol of the sparseM matrix is currently set to mtry but this can actually be any integer > 1; can even be greater than p.

Usage

```
makeAB(p, d, sparsity, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
sparsity	a real number in $(0, 1)$ that specifies the distribution of non-zero elements in the random matrix.
...	used to handle superfluous arguments passed in using paramList.

Value

rotationMatrix the matrix used to determine which mtry features or combination of features will be used to split a node.

---

mnist *A subset of the MNIST dataset for handwritten digit classification*

Description

A dataset consiting of 10 percent of the MNIST training set and the full test set

Usage

```
data(mnist)
```

Format

A list with four items: Xtrain is a training set matrix with 6000 rows (samples) and 784 columns (features), Xtrain is an integer array of corresponding training class labels, Xtest is a test set matrix of 10000 rows and 784 columns, and Ytest is the corresponding class labels. Rows in Xtrain and Xtest correspond to different images of digits, and columns correspond to the pixel intensities in each image, obtained by flattening the image pixels in column-major ordering.

Source

MNIST

References

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.

Examples

```
data(mnist)
```

`OOBPredict` *Compute out-of-bag predictions*

Description

Computes out-of-bag class predictions for a forest trained with store.oob=TRUE.

Usage

```
OOBPredict(X, forest, num.cores = 0L, Xtrain = NULL,
  output.scores = FALSE)
```

Arguments

`X`	an n sample by d feature matrix (preferable) or data frame which was used to train the provided forest.
`forest`	a forest trained using the RerF function, with store.oob=TRUE.
`num.cores`	the number of cores to use while training. If num.cores=0 then 1 less than the number of cores reported by the OS are used. (num.cores=0)
`Xtrain`	an n by d numeric matrix (preferable) or data frame. This should be the same data matrix/frame used to train the forest, and is only required if RerF was called with rank.transform = TRUE. (Xtrain=NULL)
`output.scores`	if TRUE then predicted class scores (probabilities) for each observation are returned rather than class labels. (output.scores = FALSE)

Value

predictions a length n vector of predictions in a format similar to the Y vector used to train the forest

Examples

```
library(rerf)
X <- as.matrix(iris[, 1:4])
Y <- iris[[5L]]
forest <- RerF(X, Y, store.oob = TRUE, num.cores = 1L)
predictions <- OOBPredict(X, forest, num.cores = 1L)
oob.error <- mean(predictions != Y)
```

PackForest — *Packs a forest and saves modified forest to disk for use by PackPredict function*

Description

Efficiently packs a forest trained with the RF option. Two intermediate data structures are written to disk, forestPackTempFile.csv and traversalPackTempFile.csv. The size of these data structures is proportional to a trained forest and training data respectively. Both data structures are removed at the end of the operation. The resulting forest is saved as forest.out. The size of this file is similar to the size of the trained forest.

Usage

```
PackForest(X, Y, forest)
```

Arguments

X	an n by d numeric matrix (preferable) or data frame used to train the forest.
Y	a numeric vector of size n. If the Y vector used to train the forest was not of type numeric then a simple call to as.numeric(Y) will suffice as input.
forest	a forest trained using the RerF function using the RF option.

PackPredict — *Compute class predictions for each observation in X*

Description

Predicts the classification of samples using a trained forest.

Usage

```
PackPredict(X, num.cores = 1)
```

Arguments

X	an n by d numeric matrix (preferable) or data frame. The rows correspond to observations and columns correspond to features of a test set, which should be different from the training set.
num.cores	the number of cores to use while predicting. (num.cores=0)

Value

predictions an n length vector of prediction class numbers

Examples

```
library(rerf)
trainIdx <- c(1:40, 51:90, 101:140)
X <- as.matrix(iris[, 1:4])
Y <- as.numeric(iris[, 5])

paramList <- list(p = ncol(X), d = ceiling(sqrt(ncol(X))))

forest <- RerF(X, Y, FUN = RandMatRF, paramList = paramList, rfPack = TRUE, num.cores = 1)

predictions <- PackPredict(X)
```

Predict *Compute class predictions for each observation in X*

Description

Predicts the classification of samples using a trained forest.

Usage

```
Predict(X, forest, OOB = FALSE, num.cores = 0L, Xtrain = NULL,
  aggregate.output = TRUE, output.scores = FALSE)
```

Arguments

X	an n by d numeric matrix (preferable) or data frame. The rows correspond to observations and columns correspond to features of a test set, which should be different from the training set.
forest	a forest trained using the RerF function.
OOB	if TRUE then run predictions using out-of-bag samples.
num.cores	the number of cores to use while training. If NumCores=0 then 1 less than the number of cores reported by the OS are used. (NumCores=0)
Xtrain	an n by d numeric matrix (preferable) or data frame. This should be the same data matrix/frame used to train the forest, and is only required if RerF was called with rank.transform = TRUE. (Xtrain=NULL)
aggregate.output	if TRUE then the tree predictions are aggregated weighted by their probability estimates. Otherwise, the individual tree probabilities are returned. (aggregate.output=TRUE) TODO: Remove option for aggregate output? Only options for returning aggregate predictions or probabilities
output.scores	if TRUE then predicted class scores (probabilities) for each observation are returned rather than class labels. (output.scores = FALSE)

Value

predictions an n length vector of predictions

Examples

```
library(rerf)
trainIdx <- c(1:40, 51:90, 101:140)
X <- as.matrix(iris[, 1:4])
Y <- as.numeric(iris[, 5])
forest <- RerF(X[trainIdx, ], Y[trainIdx], num.cores = 1L, rank.transform = TRUE)
# Using a set of samples with unknown classification
predictions <- Predict(X[-trainIdx, ], forest, num.cores = 1L, Xtrain = X[trainIdx, ])
error.rate <- mean(predictions != Y[-trainIdx])
```

RandMatBinary *Create a Random Matrix: Binary*

Description

Create a Random Matrix: Binary

Usage

```
RandMatBinary(p, d, sparsity, prob, catMap = NULL, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
sparsity	a real number in $(0, 1)$ that specifies the distribution of non-zero elements in the random matrix.
prob	a probability $\in (0, 1)$ used for sampling from $-1, 1$ where `prob = 0` will only sample -1 and `prob = 1` will only sample 1.
catMap	a list specifying specifies which one-of-K encoded columns in X correspond to the same categorical feature.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 8
d <- 3
sparsity <- 0.25
prob <- 0.5
set.seed(4)
(a <- RandMatBinary(p, d, sparsity, prob))
```

RandMatContinuous *Create a Random Matrix: Continuous*

Description

Create a Random Matrix: Continuous

Usage

```
RandMatContinuous(p, d, sparsity, catMap = NULL, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
sparsity	a real number in $(0, 1)$ that specifies the distribution of non-zero elements in the random matrix.
catMap	a list specifying specifies which one-of-K encoded columns in X correspond to the same categorical feature.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 8
d <- 3
sparsity <- 0.25
set.seed(4)
(a <- RandMatContinuous(p, d, sparsity))
```

RandMatCustom *Create a Random Matrix: custom*

Description

Create a Random Matrix: custom

Usage

```
RandMatCustom(p, d, nnzSample, nnzProb, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
nnzSample	a vector specifying the number of non-zeros to sample at each d. Each entry should be less than p.
nnzProb	a vector specifying probabilities in one-to-one correspondance with nnzSample.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 28
d <- 8
nnzSample <- 1:8
nnzProb <- 1 / 36 * 1:8
paramList <- list(p = p, d = d, nnzSample, nnzProb)
set.seed(8)
(a <- do.call(RandMatCustom, paramList))
```

RandMatFRC *Create a Random Matrix: FRC*

Description

Create a Random Matrix: FRC

Usage

```
RandMatFRC(p, d, nmix, catMap = NULL, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
nmix	mupliplier to d to specify the number of non-zeros.
catMap	a list specifying specifies which one-of-K encoded columns in X correspond to the same categorical feature.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 8
d <- 8
nmix <- 5
paramList <- list(p = p, d = d, nmix = nmix)
set.seed(4)
(a <- do.call(RandMatFRC, paramList))
```

RandMatFRCN *Create a Random Matrix: FRCN*

Description

Create a Random Matrix: FRCN

Usage

```
RandMatFRCN(p, d, nmix, catMap = NULL, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
nmix	mupliplier to d to specify the number of non-zeros.
catMap	a list specifying specifies which one-of-K encoded columns in X correspond to the same categorical feature.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 8
d <- 8
nmix <- 5
paramList <- list(p = p, d = d, nmix = nmix)
set.seed(8)
(a <- do.call(RandMatFRCN, paramList))
```

RandMatImageControl *Create a Random Matrix: image-control*

Description

Create a Random Matrix: image-control

Usage

```
RandMatImageControl(p, d, ih, iw, pwMin, pwMax, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
ih	the height (px) of the image.
iw	the width (px) of the image.
pwMin	the minimum patch size to sample.
pwMax	the maximum patch size to sample.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 28^2
d <- 8
ih <- iw <- 28
pwMin <- 3
pwMax <- 6
paramList <- list(p = p, d = d, ih = ih, iw = iw, pwMin = pwMin, pwMax = pwMax)
set.seed(8)
(a <- do.call(RandMatImageControl, paramList))
```

RandMatImagePatch *Create a Random Matrix: image-patch*

Description

Create a Random Matrix: image-patch

Usage

```
RandMatImagePatch(p, d, ih, iw, pwMin, pwMax, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
ih	the height (px) of the image.
iw	the width (px) of the image.
pwMin	the minimum patch size to sample.
pwMax	the maximum patch size to sample.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 28^2
d <- 8
ih <- iw <- 28
pwMin <- 3
pwMax <- 6
paramList <- list(p = p, d = d, ih = ih, iw = iw, pwMin = pwMin, pwMax = pwMax)
set.seed(8)
(a <- do.call(RandMatImagePatch, paramList))
```

RandMatPoisson *Create a Random Matrix: Poisson*

Description

Samples a binary projection matrix where sparsity is distributed $Poisson(\lambda)$.

Usage

```
RandMatPoisson(p, d, lambda, catMap = NULL, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
lambda	passed to the rpois function for generation of non-zero elements in the random matrix.
catMap	a list specifying specifies which one-of-K encoded columns in X correspond to the same categorical feature.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 8
d <- 8
lambda <- 0.5
paramList <- list(p = p, d = d, lambda = lambda)
set.seed(8)
(a <- do.call(RandMatPoisson, paramList))
```

RandMatRF *Create a Random Matrix: Random Forest (RF)*

Description

Create a Random Matrix: Random Forest (RF)

Usage

```
RandMatRF(p, d, catMap = NULL, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
catMap	a list specifying specifies which one-of-K encoded columns in X correspond to the same categorical feature.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 8
d <- 3
paramList <- list(p = p, d = d)
set.seed(4)
(a <- do.call(RandMatRF, paramList))
```

RandMatTSpatch *Create a Random Matrix: ts-patch*

Description

Create a Random Matrix: ts-patch

Usage

```
RandMatTSpatch(p, d, pwMin, pwMax, ...)
```

Arguments

p	the number of dimensions.
d	the number of desired columns in the projection matrix.
pwMin	the minimum patch size to sample.
pwMax	the maximum patch size to sample.
...	used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running RerF.

Examples

```
p <- 8
d <- 8
pwMin <- 3
pwMax <- 6
paramList <- list(p = p, d = d, pwMin = pwMin, pwMax = pwMax)
set.seed(8)
(a <- do.call(RandMatTSpatch, paramList))
```

RerF *RerF forest Generator*

Description

Creates a decision forest based on an input matrix and class vector. This is the main function in the rerf package.

Usage

```
RerF(X, Y, FUN = RandMatBinary, paramList = list(p = NA, d = NA,
  sparsity = NA, prob = NA), min.parent = 1L, trees = 500L,
  max.depth = 0, bagging = 0.2, replacement = TRUE,
  stratify = TRUE, rank.transform = FALSE, store.oob = FALSE,
  store.impurity = FALSE, progress = FALSE, rotate = FALSE,
  num.cores = 0L, seed = sample(0:1e+08, 1), cat.map = NULL,
  rfPack = FALSE)
```

Arguments

X	an n by d numeric matrix (preferable) or data frame. The rows correspond to observations and columns correspond to features.
Y	an n length vector of class labels. Class labels must be integer or numeric and be within the range 1 to the number of classes.
FUN	a function that creates the random projection matrix. If NULL and cat.map is NULL, then RandMat is used. If NULL and cat.map is not NULL, then RandMatCat is used, which adjusts the sampling of features when categorical features have been one-of-K encoded. If a custom function is to be used, then it must return a matrix in sparse representation, in which each nonzero is an array of the form (row.index, column.index, value). See RandMat or RandMatCat for details.
paramList	parameters in a named list to be used by FUN. If left unchanged, default values will be populated, see defaults for details.
min.parent	the minimum splittable node size. A node size < min.parent will be a leaf node. (min.parent = 1)
trees	the number of trees in the forest. (trees=500)

`max.depth`	the longest allowable distance from the root of a tree to a leaf node (i.e. the maximum allowed height for a tree). If max.depth=0, the tree will be allowed to grow without bound. (max.depth=0)
`bagging`	a non-zero value means a random sample of X will be used during tree creation. If replacement = FALSE the bagging value determines the percentage of samples to leave out-of-bag. If replacement = TRUE the non-zero bagging value is ignored. (bagging=.2)
`replacement`	if TRUE then n samples are chosen, with replacement, from X. (replacement=TRUE)
`stratify`	if TRUE then class sample proportions are maintained during the random sampling. Ignored if replacement = FALSE. (stratify = FALSE).
`rank.transform`	if TRUE then each feature is rank-transformed (i.e. smallest value becomes 1 and largest value becomes n) (rank.transform=FALSE)
`store.oob`	if TRUE then the samples omitted during the creation of a tree are stored as part of the tree. This is required to run OOBPredict(). (store.oob=FALSE)
`store.impurity`	if TRUE then the decrease in impurity is stored for each split. This is required to run FeatureImportance() (store.impurity=FALSE)
`progress`	if TRUE then a pipe is printed after each tree is created. This is useful for large datasets. (progress=FALSE)
`rotate`	if TRUE then the data matrix X is uniformly randomly rotated for each tree. (rotate=FALSE)
`num.cores`	the number of cores to use while training. If num.cores=0 then 1 less than the number of cores reported by the OS are used. (num.cores=0)
`seed`	the seed to use for training the forest. For two runs to match you must use the same seed for each run AND you must also use the same number of cores for each run. (seed=sample((0:100000000,1)))
`cat.map`	a list specifying which columns in X correspond to the same one-of-K encoded feature. Each element of cat.map is a numeric vector specifying the K column indices of X corresponding to the same categorical feature after one-of-K encoding. All one-of-K encoded features in X must come after the numeric features. The K encoded columns corresponding to the same categorical feature must be placed contiguously within X. The reason for specifying cat.map is to adjust for the fact that one-of-K encoding cateogorical features results in a dilution of numeric features, since a single categorical feature is expanded to K binary features. If cat.map = NULL, then RerF assumes all features are numeric (i.e. none of the features have been one-of-K encoded).
`rfPack`	boolean flag to determine whether to pack a random forest in order to improve prediction speed. This flag is only applicable when training a forest with the "rf" option. (rfPack = FALSE)

Value

forest

Examples

```
### Train RerF on numeric data ###
library(rerf)
forest <- RerF(as.matrix(iris[, 1:4]), iris[[5L]], num.cores = 1L)

### Train RerF on one-of-K encoded categorical data ###
df1 <- as.data.frame(Titanic)
nc <- ncol(df1)
df2 <- df1[NULL, -nc]
for (i in which(df1$Freq != 0L)) {
  df2 <- rbind(df2, df1[rep(i, df1$Freq[i]), -nc])
}
n <- nrow(df2) # number of observations
p <- ncol(df2) - 1L # number of features
num.categories <- apply(df2[, 1:p], 2, function(x) length(unique(x)))
p.enc <- sum(num.categories) # number of features after one-of-K encoding
X <- matrix(0, nrow = n, ncol = p.enc) # initialize training data matrix X
cat.map <- vector("list", p)
col.idx <- 0L
# one-of-K encode each categorical feature and store in X
for (j in 1:p) {
  cat.map[[j]] <- (col.idx + 1L):(col.idx + num.categories[j])
  # convert categorical feature to K dummy variables
  X[, cat.map[[j]]] <- dummies::dummy(df2[[j]])
  col.idx <- col.idx + num.categories[j]
}
Y <- df2$Survived

# specifying the cat.map in RerF allows training to
# be aware of which dummy variables correspond
# to the same categorical feature
forest <- RerF(X, Y, num.cores = 1L, cat.map = cat.map)
## Not run:
# takes longer than 5s to run.
# adding a continuous feature along with the categorical features
# must be prepended to the categorical features.
set.seed(1234)
xp <- runif(nrow(X))
Xp <- cbind(xp, X)
cat.map1 <- lapply(cat.map, function(x) x + 1)
forestW <- RerF(Xp, Y, num.cores = 1L, cat.map = cat.map1)

## End(Not run)

### Train a random rotation ensemble of CART decision trees (see Blaser and Fryzlewicz 2016)
forest <- RerF(as.matrix(iris[, 1:4]), iris[[5L]],
  num.cores = 1L,
  FUN = RandMatRF, paramList = list(p = 4, d = 2), rotate = TRUE
)
```

RunFeatureImportance *Compute Feature Importance of a single RerF tree*

Description

Computes feature importance of every unique feature used to make a split in a single tree.

Usage

```
RunFeatureImportance(tree, unique.projections)
```

Arguments

tree	a single tree from a trained RerF model with argument store.impurity = TRUE.
unique.projections	a list of all of the unique split projections used in the RerF model.

Value

feature.imp

RunOOB *Predict class labels on out-of-bag observations using a single tree.*

Description

This is the base function called by OOBPredict.

Usage

```
RunOOB(X, tree)
```

Arguments

X	an n sample by d feature matrix (preferable) or data frame which was used to train the provided forest.
tree	a tree from a forest returned by RerF.

Value

out prediction matrix used by OOBPredict

RunPredict *Predict class labels on a test set using a single tree.*

Description

This is the base function called by Predict.

Usage

```
RunPredict(X, tree)
```

Arguments

X	an n sample by d feature matrix (preferable) or data frame which was used to train the provided forest.
tree	a tree from a forest returned by RerF.

Value

predictions an n length vector of prediction based on the tree provided to this function

RunPredictLeaf *Calculate similarity using a single tree.*

Description

This is the base function called by ComputeSimilarity.

Usage

```
RunPredictLeaf(X, tree)
```

Arguments

X	an n sample by d feature matrix (preferable) or data frame which was used to train the provided forest.
tree	a tree from a forest returned by RerF.

Value

similarity based on one tree

StrCorr *Compute tree strength and correlation*

Description

Computes estimates of tree strength and correlation according to the definitions in Breiman's 2001 Random Forests paper.

Usage

```
StrCorr(Yhats, Y)
```

Arguments

Yhats	predicted class labels for each tree in a forest.
Y	true class labels.

Value

scor

Examples

```
library(rerf)
trainIdx <- c(1:40, 51:90, 101:140)
X <- as.matrix(iris[, 1:4])
Y <- iris[[5]]
forest <- RerF(X[trainIdx, ], Y[trainIdx], num.cores = 1L)
predictions <- Predict(X[-trainIdx, ], forest, num.cores = 1L, aggregate.output = FALSE)
scor <- StrCorr(predictions, Y[-trainIdx])
```

TwoMeansCut *Find minimizing Two Means Cut for Vector*

Description

Find minimizing Two Means Cut for Vector

Usage

```
TwoMeansCut(X)
```

Arguments

X	a one dimensional vector

Value

list containing minimizing cut point and corresponding sum of left and right variances.

`Urerf` *Unsupervised RerF forest Generator*

Description

Creates a decision forest based on an input matrix.

Usage

```
Urerf(X, trees = 100, min.parent = round(nrow(X)^0.5),
  max.depth = NA, mtry = ceiling(ncol(X)^0.5), normalizeData = TRUE,
  Progress = TRUE)
```

Arguments

`X`	an n by d numeric matrix. The rows correspond to observations and columns correspond to features.
`trees`	the number of trees in the forest. (trees=100)
`min.parent`	the minimum splittable node size. A node size < min.parent will be a leaf node. (min.parent = round(nrow(X)^.5))
`max.depth`	the longest allowable distance from the root of a tree to a leaf node (i.e. the maximum allowed height for a tree). If max.depth=NA, the tree will be allowed to grow without bound. (max.depth=NA)
`mtry`	the number of features to test at each node. (mtry=ceiling(ncol(X)^.5))
`normalizeData`	a logical value that determines if input data is normalized to values ranging from 0 to 1 prior to processing. (normalizeData=TRUE)
`Progress`	boolean for printing progress.

Value

urerfStructure

Examples

```
### Train RerF on numeric data ###
library(rerf)
urerfStructure <- Urerf(as.matrix(iris[, 1:4]))

dissimilarityMatrix <- hclust(as.dist(1 - urerfStructure$similarityMatrix), method = "mcquitty")
clusters <- cutree(dissimilarityMatrix, k = 3)
```

Index

∗Topic **datasets**
    mnist, 7

BuildTree, 3

checkInputMatrix, 4
ComputeSimilarity, 4

defaults, *3*, *19*

FeatureImportance, 5

GrowUnsupervisedForest, 6

makeAB, 6
mnist, 7

OOBPredict, 8

PackForest, 9
PackPredict, 9
Predict, 10

RandMatBinary, 11
RandMatContinuous, 12
RandMatCustom, 13
RandMatFRC, 13
RandMatFRCN, 14
RandMatImageControl, 15
RandMatImagePatch, 16
RandMatPoisson, 17
RandMatRF, 17
RandMatTSpatch, 18
RerF, *11–18*, 19
rpois, *17*
RunFeatureImportance, 22
RunOOB, 22
RunPredict, 23
RunPredictLeaf, 23

StrCorr, 24

TwoMeansCut, 24

Urerf, 25