# Package 'semiArtificial'

March 31, 2017

**Title** Generator of Semi-Artificial Data

**Version** 2.2.5

**Date** 2017-03-28

**Author** Marko Robnik-Sikonja

**Maintainer** Marko Robnik-Sikonja <marko.robnik@fri.uni-lj.si>

**Description** Contains methods to generate and evaluate semi-artificial data sets.
Based on a given data set different methods learn data properties using machine learning algorithms and
generate new data with the same properties.
The package currently includes the following data generators:
i) a RBF network based generator using rbfDDA() from package 'RSNNS',
ii) a Random Forest based generator for both classification and regression problems
iii) a density forest based generator for unsupervised data
Data evaluation support tools include:
a) single attribute based statistical evaluation: mean, median, standard deviation, skewness, kurtosis, medcouple, L/RMC, KS test, Hellinger distance
b) evaluation based on clustering using Adjusted Rand Index (ARI) and FM
c) evaluation based on classification performance with various learning models, e.g., random forests.

**License** GPL-3

**URL** http://lkm.fri.uni-lj.si/rmarko/software/

**Imports** CORElearn (>= 1.50.3),RSNNS,MASS,nnet,cluster,fpc,stats,timeDate,robustbase,ks,logspline,methods,mcclust,flexclust,StatMatch

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2017-03-31 06:13:09 UTC

## R topics documented:

---

`semiArtificial-package`

*Generation and evaluation of semi-artificial data*

---

## Description

The package semiArtificial contains methods to generate and evaluate semi-artificial data sets. Different data generators take a data set as an input, learn its properties using machine learning algorithms and generates new data with the same properties.

## Details

The package currently includes the following data generators:

- a RBF network based generator using rbfDDA model from RSNNS package.
- generator using density tree forest for unsupervised data,
- generator using random forest for classification and regression.

Data evaluation support tools include:

- statistical evaluation: mean, median,standard deviation, skewness, kurtosis, medcouple, L/RMC,
- evaluation based on clustering using Adjusted Rand Index (ARI) and Fowlkes-Mallows index (FM),
- evaluation based on prediction with a model, e.g., random forests.

Further software and development versions are available at `http://lkm.fri.uni-lj.si/rmarko/software`.

## Author(s)

Marko Robnik-Sikonja

## References

Marko Robnik-Sikonja: Not enough data? Generate it!. *Technical Report, University of Ljubljana, Faculty of Computer and Information Science*, 2014

Other references are available from `http://lkm.fri.uni-lj.si/rmarko/papers/`

## See Also

`rbfDataGen`, `treeEnsemble`, `newdata`, `dataSimilarity`, `dsClustCompare`, `performanceCompare`.

cleanData *Rejection of new instances based on their distance to existing instances*

## Description

The function contains three data cleaning methods, the first two reject instances whose distance to their nearest neighbors in the existing data are too small or too large. The first checks distance between instances disregarding class, the second checks distances between instances taking only instances from the same class into account. The third method reassigns response variable using the prediction model stored in the generator `teObject`.

## Usage

```
cleanData(teObject, newdat, similarDropP=NA, dissimilarDropP=NA,
          similarDropPclass=NA, dissimilarDropPclass=NA,
  nearestInstK=1, reassignResponse=FALSE, cleaningObject=NULL)
```

## Arguments

| | |
|---|---|
| `teObject` | An object of class `TreeEnsemble` containing a generator structure as returned by `treeEnsemble`. The `teObject` contains generator's training instances from which we compute a distance distribution of instances to their `nearestInsK` nearest instances. This distance distribution, computed on the training data of the generator, serves as a criterion to reject new instances from `newdata`, i.e. based on parameters below we reject the instances too close or to far away from their nearest neighbors in generator's training data. The computed distance distributions are stored and returned as `cleaningObject` component of returned list. If it is provided on subsequent calls, this reduces computational load. |
| `newdat` | A `data.frame` object with the (newly generated) data to be cleaned. |
| `similarDropP` | With numeric parameters `similarDropP` and `dissimilarDropP` (with the default value NA and the valid value range in [0, 1]) one removes instances in `newdat` too close to generator's training instances or too far away from these instances. The distance distribution is computed based on instances stored in `teObject`. For each instance in $teObject$ we store the distance to its `nearestInsK` nearest instances (disregarding the identical instances). These distances are sorted and represent a distribution of nearest distances for all training instances. The values `similarDropP` and `dissimilarDropP` represent a proportion of allowed smaller/larger distances computed on the generator's training data contained in the `teObject`. |
| `dissimilarDropP` | See `similarDropP`. |
| `similarDropPclass` | For classification problems only and similarly to the `similarDropP` and `dissimilarDropP` above, with the `similarDropPclass` and `dissimilarDropPclass` |

(also in a [0, 1] range) we also removes instances in newdat too close to generator's training instances or too far away from these instances, but only taking near instances from the same class into account. The similarDropPclass conatins either a single integer giving thresholds for all class values or a vector of thresholds, one for each class. If the vector is of insufficient leght iit is replicated to the proper length using function rep. The generated distance distributions are stored in the cleaningObject component of the returned list.

dissimilarDropPclass
: See similarDropPclass.

nearestInstK
: An integer with default value of 1, controls how many generator's training instances we take into account when computing the distance distribution of nearest instances.

reassignResponse
: is a logical value controlling whether the response variable of the newdat shall be set anew using a random forest prediction model or taken as it is. The default value reassign=FALSE means that values of response are not changed.

cleaningObject
: is a list object with a precomputed distance distributions and predictor from previous runs of the same function. If provided this saves computation time.

## Details

The function uses the training instances stored in the generator teObject to compute distribution of distances from instances to their nearestInstK nearest instances. For classification problems the distributions can also be computed only for instances from the same class. Using these near distance distributions the function rejects all instances too close or too far away from existing instances.

The default value of similarDropP, dissimilarDropP, similarDropPclass, and dissimilarDropPclass is NA and means that the near/far values are not rejected. The same effect has value 0 for similarDropP and similarDropPclass, and value 1 for dissimilarDropP and dissimilarDropPclass.

## Value

The method returns a list object with two components:

cleanData
: is a data.frame containing the instances left after rejection of too close or too distant instances from newdata.

cleaningObject
: is a list containing computed distributions of nearest distances (also class-based fro classification problems, and possibly a predictor used for reassigning the response variable.

## Author(s)

Marko Robnik-Sikonja

## See Also

treeEnsemble, newdata.TreeEnsemble.

## Examples

```
# inspect properties of the iris data set
plot(iris, col=iris$Species)
summary(iris)

irisEnsemble<- treeEnsemble(Species~.,iris,noTrees=10)

# use the generator to create new data with the generator
irisNewEns <- newdata(irisEnsemble, size=150)

#inspect properties of the new data
plot(irisNewEns, col = irisNewEns$Species) #plot generated data
summary(irisNewEns)

clObj <- cleanData(irisEnsemble, irisNewEns, similarDropP=0.05, dissimilarDropP=0.95,
                   similarDropPclass=0.05, dissimilarDropPclass=0.95,
           nearestInstK=1, reassignResponse=FALSE, cleaningObject=NULL)
```

---

`dataSimilarity` *Evaluate statistical similarity of two data sets*

---

## Description

Use mean, standard deviation, skewness, kurtosis, Hellinger distance and KS test to compare similarity of two data sets.

## Usage

```
dataSimilarity(data1, data2, dropDiscrete=NA)
```

## Arguments

| | |
|---|---|
| `data1` | A `data.frame` containing the reference data. |
| `data2` | A `data.frame` with the same number and names of columns as `data1`. |
| `dropDiscrete` | A vector discrete attribute indices to skip in comparison. Typically we might skip class, because its distribution was forced by the user. |

## Details

The function compares data stored in `data1` with `data2` on per attribute basis by computing several statistics: mean, standard deviation, skewness, kurtosis, Hellinger distance and KS test.

## Value

The method returns a list of statistics computed on both data sets:

| | |
|---|---|
| `equalInstances` | The number of instances in `data2` equal to the instances in `data1`. |
| `stats1num` | A matrix with rows containing statistics (mean, standard deviation, skewness, and kurtosis) computed on numeric attributes of `data1`. |
| `stats2num` | A matrix with rows containing statistics (mean, standard deviation, skewness, and kurtosis) computed on numeric attributes of `data2`. |
| `ksP` | A vector with p-values of Kolmogorov-Smirnov two sample tests, performed on matching attributes from `data1` and `data2`. |
| `freq1` | A list with value frequencies for discrete attributes in `data1`. |
| `freq2` | A list with value frequencies for discrete attributes in `data2`. |
| `dfreq` | A list with differences in frequencies of discrete attributes' values between `data1` and `data2`. |
| `dstatsNorm` | A matrix with rows containing difference between statistics (mean, standard deviation, skewness, and kurtosis) computed on [0,1] normalized numeric attributes for `data1` and `data2`. |
| `hellingerDist` | A vector with Hellinger distances between matching attributes from `data1` and `data2`. |

## Author(s)

Marko Robnik-Sikonja

## See Also

`newdata.RBFgenerator`.

## Examples

```
# use iris data set, split into training and testing data
set.seed(12345)
train <- sample(1:nrow(iris),size=nrow(iris)*0.5)
irisTrain <- iris[train,]
irisTest <- iris[-train,]

# create RBF generator
irisGenerator<- rbfDataGen(Species~.,irisTrain)

# use the generator to create new data
irisNew <- newdata(irisGenerator, size=100)

# compare statistics of original and new data
dataSimilarity(irisTest, irisNew)
```

dsClustCompare *Evaluate clustering similarity of two data sets*

## Description

Similarity of two data sets is compared with a method using any of clustering comparison metrics: Adjusted Rand Index (ARI), Fowlkes-Mallows index(FM), Jaccard Index (J), or Variation of Information index (VI).

## Usage

```
dsClustCompare(data1, data2)
```

## Arguments

| | |
|---|---|
| data1 | A data.frame containing the reference data. |
| data2 | A data.frame with the same number and names of columns as data1. |

## Details

The function compares data stored in data1 with data2 by first performing partitioning around medoids (PAM) clustering on data1. Instances from data2 are than assigned to the cluster with the closest medoid. In second step PAM clustering is performed on data2 and instances from data1 are assigned to the clusters with closest medoids. The procedure gives us two clusterings on the same instances which we can compare using any of ARI, FM, J, or VI. The higher the value of ARI/FM/J the more similar are the two data sets, and reverse is true for VI, where two perfectly matching partitions produce 0 score. For random clustering ARI returns a value around zero (negative values are possible) and for perfectly matching clustering ARI is 1. FM and J values are strictly in [0, 1].

## Value

The method returns a value of a list containing ARI and/or FM, depending on the parameters.

## Author(s)

Marko Robnik-Sikonja

## See Also

newdata.RBFgenerator.

## Examples

```
# use iris data set

# create RBF generator
irisGenerator<- rbfDataGen(Species~.,iris)
```

```
# use the generator to create new data
irisNew <- newdata(irisGenerator, size=200)

# compare ARI computed on clustering with original and new data
dsClustCompare(iris, irisNew)
```

---

newdata *Generate semi-artificial data using a generator*

---

## Description

Using a generator build with rbfDataGen or treeEnsemble the method generates size new instances.

## Usage

```
## S3 method for class 'RBFgenerator'
newdata(object, size, var=c("estimated","Silverman"),
                                classProb=NULL, defaultSpread=0.05, ... )
## S3 method for class 'TreeEnsemble'
newdata(object, size=1, onlyPath=FALSE, classProb=NULL, predictClass=FALSE, ...)
```

## Arguments

| | |
|---|---|
| object | An object of class RBFgenerator or TreeEnsemble containing a generator structure as returned by rbfDataGen or treeEnsemble, respectively. |
| size | A number of instances to generate. |
| var | For the generator of type RBFgenerator the parameter var determines the method of kernel width (variance) estimation. Supported options are "estimated" and "Silverman". |
| classProb | For classification problems, a vector of desired class value probability distribution. Default value classProb=NULL uses probability distribution of the generator's training instances. |
| defaultSpread | For the generator of type RBFgenerator the parameter is a numeric value replacing zero spread in case var="estimated" is used. The value defaultSpread=NULL keeps zero spread values. |
| onlyPath | For the generator of type TreeEnsemble and attribute density data in the leaves (densityData="leaf"), the parameter is a boolean variable indicating if only attributes on the path from the root to the leaf are generated in the leaf. If onlyPath=FALSE all value are generated in the first randomly chosen leaf of a tree, else only attributes on the path are generated and then the next random tree is selected. |

| | |
|---|---|
| predictClass | For classification problems and the generator of type TreeEnsemble the parameter determines if the class value is set through prediction with the forest or set according to the class value distribution of the selected leaf. |
| ... | Additional parameters passed to density estimation functions kde, logspline, and quantile. |

## Details

The function uses the object structure as returned by rbfDataGen or treeEnsemble. In case of RBFgenerator the object contains descriptions of the Gaussian kernels, which model the original data. The kernels are used to generate a required number of new instances. The kernel width of provided kernels can be set in two ways. By setting var="estimated" the estimated spread of the training instances that have the maximal activation value for the particular kernel is used. Using var="Silverman" width is set by the generalization of Silverman's rule of thumb to multivariate case (unreliable for larger dimensions).

In case of TreeEnsemble generator no additional parameters are needed, except for the number of generated instances.

## Value

The method returns a data.frame object with required number of instances.

## Author(s)

Marko Robnik-Sikonja

## See Also

rbfDataGen.

## Examples

```
# inspect properties of the iris data set
plot(iris, col=iris$Species)
summary(iris)

# create RBF generator
irisRBF<- rbfDataGen(Species~.,iris)
# create treesemble  generator
irisEnsemble<- treeEnsemble(Species~.,iris,noTrees=10)


# use the generator to create new data with both generators
irisNewRBF <- newdata(irisRBF, size=150)
irisNewEns <- newdata(irisEnsemble, size=150)

#inspect properties of the new data
plot(irisNewRBF, col = irisNewRBF$Species) #plot generated data
summary(irisNewRBF)
plot(irisNewEns, col = irisNewEns$Species) #plot generated data
summary(irisNewEns)
```

`performanceCompare` *Evaluate similarity of two data sets based on predictive performance*

## Description

Depending on the type of problem (classification or regression), a classification performance (accuracy, AUC, brierScore, etc) or regression performance (RMSE, MSE, MAE, RMAE, etc) on two data sets is used to compare the similarity of two data sets.

## Usage

```
performanceCompare(data1, data2, formula, model="rf", stat=NULL, ...)
```

## Arguments

| | |
|---|---|
| `data1` | A `data.frame` containing the reference data. |
| `data2` | A `data.frame` with the same number and names of columns as `data1`. |
| `formula` | A `formula` specifying the response and predictive variables. |
| `model` | A predictive model used for performance comparison. The default value "rf" stands for random forest, but any classification or regression model supported by function `CoreModel` in CORElearn package can be used. |
| `stat` | A statistics used as performance indicator. The default value is NULL and means that for classification "accuracy" is used, and for regression "RMSE"" (relative mean squared error) is used. Other values supported and output by `modelEval` from CORElearn package can be used e.g., "AUC" or "brierScore". |
| `...` | Additional parameters passed to `CoreModel` function. |

## Details

The function compares data stored in `data1` with `data2` by comparing models constructed on `data1` and evaluated on both `data1` and `data2` with models built on `data2` and evaluated on both `data1` and `data2`. The difference between these performances are indicative on similarity of the data sets if used in machine learning and data mining. The performance indicator used is determined by parameter `stat`.

## Value

The method returns a list of performance indicators computed on both data sets:

| | |
|---|---|
| `diff.m1` | The difference between performance of model built on `data1` (and evaluated on both `data1` and `data2`.) |
| `diff.m2` | The difference between performance of model built on `data2` (and evaluated on both `data1` and `data2`.) |
| `perf.m1d1` | The performance of model built on `data1` on `data1`. |
| `perf.m1d2` | The performance of model built on `data1` on `data2`. |
| `perf.m2d1` | The performance of model built on `data2` on `data1`. |
| `perf.m2d2` | The performance of model built on `data2` on `data2`. |

## Author(s)

Marko Robnik-Sikonja

## See Also

newdata.RBFgenerator.

## Examples

```
# use iris data set

# create RBF generator
irisGenerator<- rbfDataGen(Species~.,iris)

# use the generator to create new data
irisNew <- newdata(irisGenerator, size=200)

# compare statistics of original and new data
performanceCompare(iris, irisNew, Species~.)
```

---

# rbfDataGen *A data generator based on RBF network*

---

## Description

Using given formula and data the method builds a RBF network and extracts its properties thereby preparing a data generator which can be used with newdata.RBFgenerator method to generate semi-artificial data.

## Usage

```
rbfDataGen(formula, data, eps=1e-4, minSupport=1,
           nominal=c("encodeBinary","asInteger"))
```

## Arguments

| | |
|---|---|
| formula | A formula specifying the response and variables to be modeled. |
| data | A data frame with training data. |
| eps | The minimal probability considered in data generator to be larger than 0. |
| minSupport | The minimal number of instances defining a Gaussian kernel to copy the kernel to the data generator. |
| nominal | The way how to treat nominal features. The option "asInteger" converts factors into integers and treats them as numeric features. The option "encodeBinary" converts each nominal attribute into a set of binary features, which encode the nominal value, e.g., for three valued attribute three binary attributes are constructed, each encoding a presence of one nominal value with 0 or 1. |

## Details

Parameter formula is used as a mechanism to select features (attributes) and the prediction variable (response, class). Only simple terms can be used and interaction terms are not supported. The simplest way is to specify just the response variable using e.g. class ~ .. See examples below.

A RBF network is build using rbfDDA from RSNNS package. The learned Gaussian kernels are extracted and used in data generation with newdata.RBFgenerator method.

## Value

The created model is returned as a structure of class RBFgenerator, containing the following items:

| | |
|---|---|
| noGaussians | The number of extracted Gaussian kernels. |
| centers | A matrix of Gaussian kernels' centers, with one row for each Gaussian kernel. |
| probs | A vector of kernel probabilities. Probabilities are defined as relative frequencies of training set instances with maximal activation in the given kernel. |
| unitClass | A vector of class values, one for each kernel. |
| bias | A vector of kernels' biases, one for each kernel. The bias is multiplied by the kernel activation to produce output value of given RBF network unit. |
| spread | A matrix of estimated variances for the kernels, one row for each kernel. The j-th value in i-th row represents the variance of training instances for j-th attribute with maximal activation in i-th Gaussian. |
| gNoActivated | A vector containing numbers of training instances with maximal activation in each kernel. |
| noAttr | The number of attributes in training data. |
| datNames | A vector of attributes' names. |
| originalNames | A vector of original attribute names. |
| attrClasses | A vector of attributes' classes (i.e., data types like numeric or factor). |
| attrLevels | A list of levels for discrete attributes (with class factor). |
| attrOrdered | A vector of type logical indicating whether the attribute is ordered (only possible for attributes of type factor. |
| normParameters | A list of parameters for normalization of attributes to [0,1]. |
| noCol | The number of columns in the internally generated data set. |
| isDiscrete | A vector of type logical, each value indicating whether a respective attribute is discrete. |
| noAttrGen | The number of attributes to generate. |
| nominal | The value of parameter nominal. |

## Author(s)

Marko Robnik-Sikonja

## References

Marko Robnik-Sikonja: Not enough data? Generate it!. *Technical Report, University of Ljubljana, Faculty of Computer and Information Science*, 2014

Other references are available from http://lkm.fri.uni-lj.si/rmarko/papers/

## See Also

newdata.RBFgenerator.

## Examples

```
# use iris data set, split into training and testing, inspect the data
set.seed(12345)
train <- sample(1:nrow(iris),size=nrow(iris)*0.5)
irisTrain <- iris[train,]
irisTest <- iris[-train,]

# inspect properties of the original data
plot(irisTrain, col=iris$Species)
summary(irisTrain)

# create rbf generator
irisGenerator<- rbfDataGen(Species~.,irisTrain)

# use the generator to create new data
irisNew <- newdata(irisGenerator, size=200)

#inspect properties of the new data
plot(irisNew, col = irisNew$Species) #plot generated data
summary(irisNew)
```

---

# treeEnsemble *A data generator based on forest*

---

## Description

Using given `formula` and `data` the method `treeEnsemble` builds a tree ensemble and turns it into a data generator, which can be used with `newdata` method to generate semi-artificial data. The methods supports classification, regression, and unsupervised data, depending on the input and parameters. The method `indAttrGen` generates data from the same distribution as the input data, but assuming conditionally independent attributes.

## Usage

```
treeEnsemble(formula, dataset, noTrees = 100, minNodeWeight=2, noSelectedAttr=0,
  problemType=c("byResponse","classification", "regression","density"),
   densityData=c("leaf", "topDown", "bottomUp","no"),
   cdfEstimation = c("ecdf","logspline","kde"),
```

```
densitySplitMethod=c("balancedSplit","randomSplit","maxVariance"),
    estimator=NULL,...)

indAttrGen(formula, dataset, cdfEstimation = c("ecdf","logspline","kde"),
           problemType="byResponse")
```

## Arguments

| | |
|---|---|
| `formula` | A formula specifying the response and variables to be modeled. |
| `dataset` | A data frame with training data. |
| `noTrees` | The number of trees in the ensemble. |
| `minNodeWeight` | The minimal number of instances in a tree leaf. |
| `noSelectedAttr` | Number of randomly selected attributes in each node which are considered as possible splits. In general this should be a positive integer, but values 0, -1, and -2 are also possible. The default value is `noSelectedAttr=0`, which causes random selection of integer rounded $\sqrt{a}$ attributes, where $a$ is the number of all attributes. Value -1 means that $1 + \log_2 a$ attributes are selected and value -2 means that all attributes are selected. |
| `problemType` | The type of the problem modeled: classification, regression, or unsupervised (density estimation). The default value `"byResponse"` indicates that the problem type is deducted based on `formula` and `data`. |
| `densityData` | The type of generator data and place where new instances are generated: in the leafs, top down from the root of the tree to the leaves, bottom up from the leaves to root. In case of value `"no"` the ensemble contains no generator data and can be used as an ordinary ensemble predictor (although probably slow, as it is written entirely in R). |
| `cdfEstimation` | The manner values are generated and the type of data stored in the generator: `"ecdf"` indicates values are generated from empirical cumulative distributions stored for each variable separately; `"logspline"` means that value distribution is modeled with logsplines, and `"kde"` indicates that Gaussian kernel density estimation is used. |
| `densitySplitMethod` | In case `problemType="density"` the parameters determines the criteria for selection of split in the density tree. Possible choices are balanced (a split value is chosen in such a way that the split is balanced), random (split value is chosen randomly) and maxVariance (split with maximal variance is chosen). |
| `estimator` | The attribute estimator used to select the node split in classification and regression trees. Function `attrEval` from `CORElearn` package is used, so the values have to be compatible with that function. The default value `NULL` chooses Gini index in case of classification problems and MSE (mean squared error in resulting splits) in case of regression. |
| `...` | Further parameters to be passed onto probability density estimators. |

## Details

Parameter `formula` is used as a mechanism to select features (attributes) and the prediction variable (response) from the data. Only simple terms can be used and interaction terms are not supported.

The simplest way is to specify just the response variable using e.g. `class ~ .`. For unsupervised problems all variables can be selected using formula `~ .`. See examples below.

A forest of trees is build using R code. The base models of the ensemble are classification, regression or density trees with additional information stored at the appropriate nodes. New data can be generated using `newdata` method.

The method `indAttrGen` generates data from the same distribution as the input data (provided in `dataset`), but assumes conditional independence of attributes. This assumption makes the generated data a simple baseline generator. Internally, the method calls `treeEnsemble` with parameters `noTrees=1`, `minNodeWeight=nrow(dataset)`, `densityData="leaf"`.

## Value

The created model is returned with additional data stored as a list and also in the trees. The model can be used with function `newdata` to generate new values.

## Author(s)

Marko Robnik-Sikonja

## References

Marko Robnik-Sikonja: Not enough data? Generate it!. *Technical Report, University of Ljubljana, Faculty of Computer and Information Science*, 2014

Other references are available from http://lkm.fri.uni-lj.si/rmarko/papers/

## See Also

`newdata`.

## Examples

```
# use iris data set, split into training and testing, inspect the data
set.seed(12345)
train <- sample(1:nrow(iris),size=nrow(iris)*0.5)
irisTrain <- iris[train,]
irisTest <- iris[-train,]

# inspect properties of the original data
plot(iris[,-5], col=iris$Species)
summary(iris)

# create tree ensemble generator for classification problem
irisGenerator<- treeEnsemble(Species~., irisTrain, noTrees=10)

# use the generator to create new data
irisNew <- newdata(irisGenerator, size=200)

#inspect properties of the new data
plot(irisNew[,-5], col = irisNew$Species) # plot generated data
summary(irisNew)
```

```
## Not run:
# create tree ensemble generator for unsupervised problem
irisUnsupervised<- treeEnsemble(~.,irisTrain[,-5], noTrees=10)
irisNewUn <- newdata(irisUnsupervised, size=200)
plot(irisNewUn) # plot generated data
summary(irisNewUn)

# create tree ensemble generator for regression problem
CO2gen<- treeEnsemble(uptake~.,CO2, noTrees=10)
CO2New <- newdata(CO2gen, size=200)
plot(CO2) # plot original data
plot(CO2New) # plot generated data
summary(CO2)
summary(CO2New)

## End(Not run)
```

# Index