# Package 'ganGenerativeData'

June 17, 2024

**Type** Package

**Title** Generate Generative Data for a Data Source

**Version** 2.0.1

**Date** 2024-06-17

**Author** Werner Mueller

**Maintainer** Werner Mueller <werner.mueller5@chello.at>

**Description**

Generative Adversarial Networks are applied to generate generative data for a data source. A generative model consisting of a generator and a discriminator network is trained. During iterative training the distribution of generated data is converging to that of the data source. Direct applications of generative data are the created functions for data classifying and missing data completion. A software service for accelerated training of generative models on graphics processing units is available. Reference: Goodfellow et al. (2014) <doi:10.48550/arXiv.1406.2661>.

**License** GPL (>= 2)

**Imports** Rcpp (>= 1.0.3), tensorflow (>= 2.0.0), httr (>= 1.4.7)

**LinkingTo** Rcpp

**RoxygenNote** 7.2.3

**SystemRequirements** TensorFlow (https://www.tensorflow.org)

**NeedsCompilation** yes

**Encoding** UTF-8

**Repository** CRAN

**Date/Publication** 2024-06-17 08:00:02 UTC

## Contents

---

`ganGenerativeData-package`

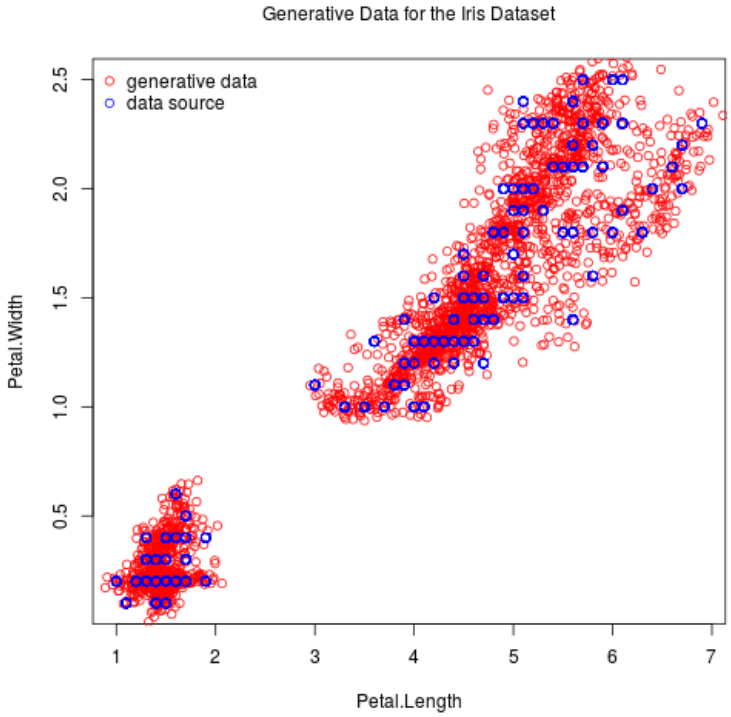
*Generate generative data for a data source*

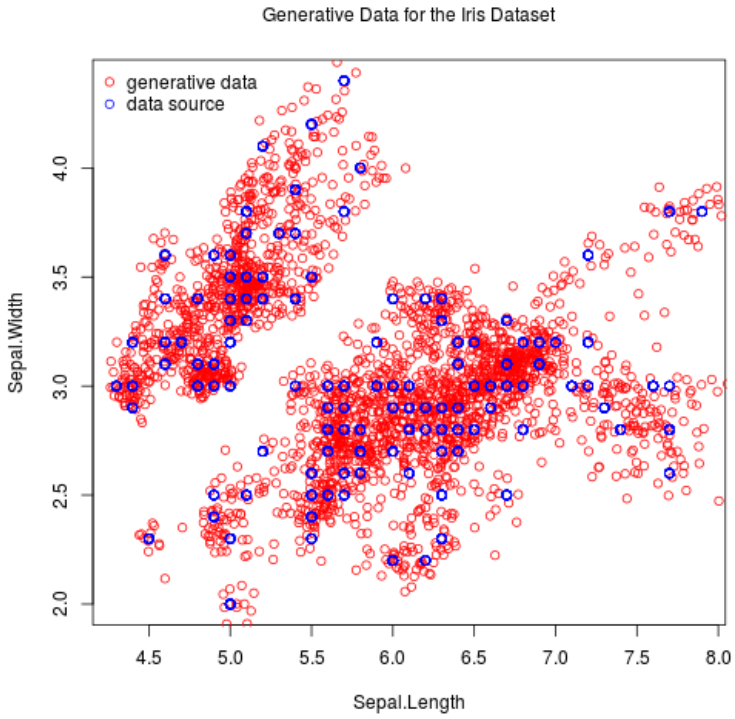
---

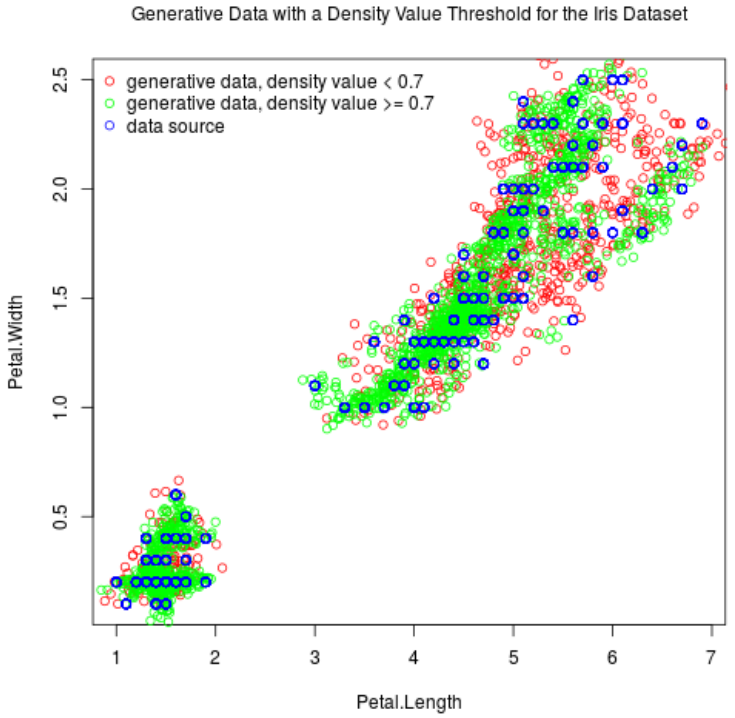
## Description

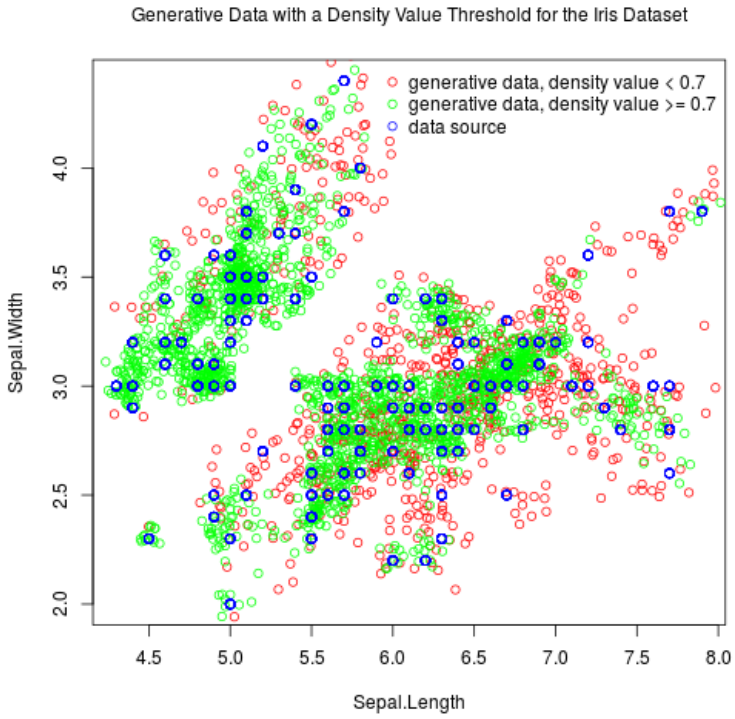
Generative Adversarial Networks are applied to generate generative data for a data source. A generative model consisting of a generator and a discriminator network is trained. During iterative training the distribution of generated data is converging to that of the data source. Direct applications of generative data are the created functions for data classifying and missing data completion. Generated data can be written to a file in training and after finished training in a separate generation step. First method accumulates generative data using a dynamic model, second method generates generative data using a static model.

Inserted images show two-dimensional projections of generative data for the iris dataset:




Generative Data for the Iris Dataset
generative data
data source
Sepal.Width
Sepal.Length
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


Generative Data with a Density Value Threshold for the Iris Dataset
generative data, density value < 0.7
generative data, density value >= 0.7
data source
Petal.Width
Petal.Length
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7




**Method of missing data completion:**

Without loss of generality we consider the case of two random variables $X, Y$. Let $g_{XY}(x, y)$ be the continuous probability density function of joint distribution $X, Y$ and $G = ((x_i, y_i))_{i=1}^{n}$ a sequence of occurrences of $X, Y$.

In order to complete missing data for a given $X = x_0$ we consider the conditional probability density function $g_{Y|X=x_0}(y)$.

We compute the mean of $g_{XY}(x, y)$ over interval $[x_0 - \epsilon, x_0 + \epsilon]$ and then the limit by applying the rule of de l'Hospital

$$\lim_{\epsilon \to 0} 1/2\epsilon \int_{x_0-\epsilon}^{x_0+\epsilon} g_{XY}(x, y)dx = g_{Y|X=x_0}(y)$$

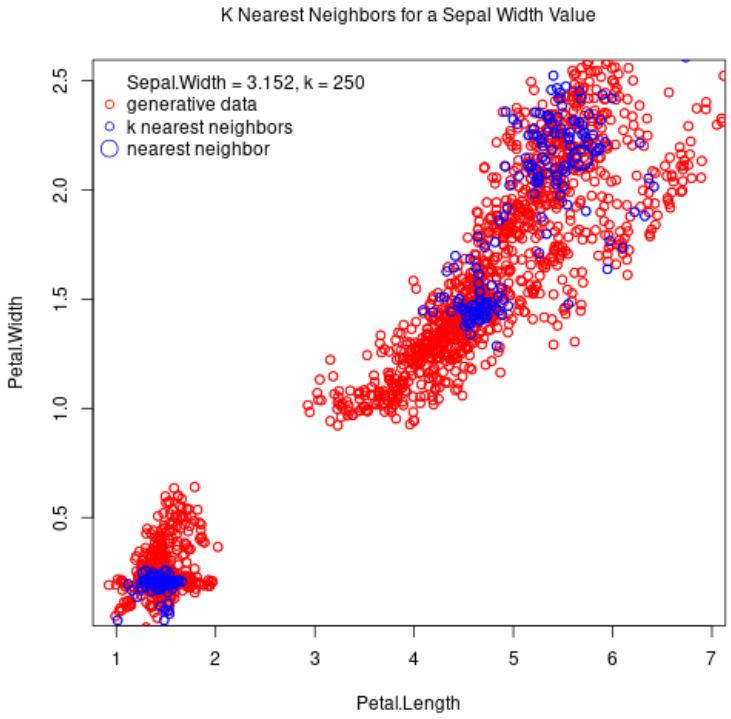
So the mean of $g_{XY}(x, y)$ over interval $[x_0 - \epsilon, x_0 + \epsilon]$ converges to $g_{Y|X=x_0}(y)$ for $\epsilon \to 0$ and the subsequence of occurrences of $G$ in the interval is given by $G_{x_0,\epsilon} = ((x_{i_j}, y_{i_j}))$ with $1 <= i_j <= n, x_{i_j} \in [x_0 - \epsilon, x_0 + \epsilon]$.

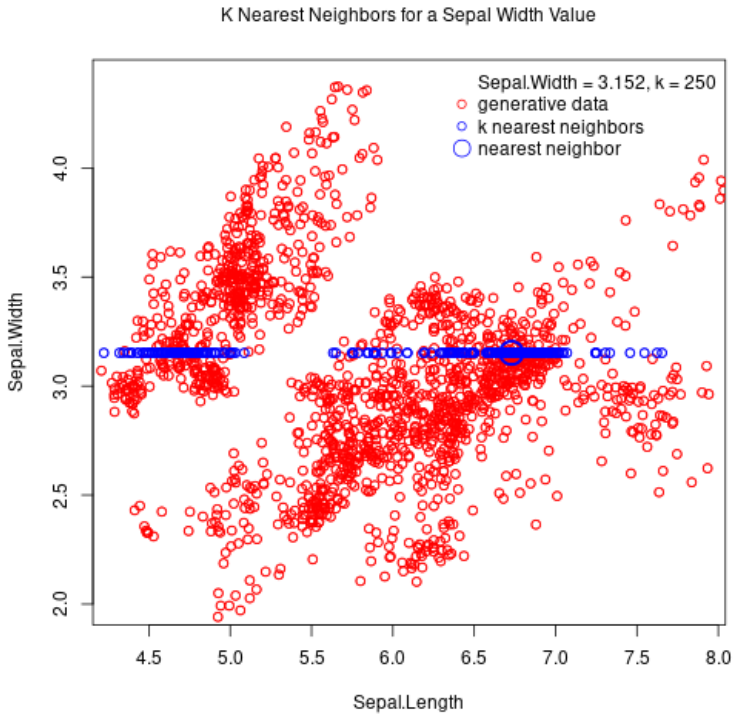
A candidate for missing $Y = y_0$ for a given $X = x_0$ is then $y_k$ of a uniform randomly selected $(x_k, y_k)$ from $G_{x_0,\epsilon}$.

This is applied for generated generative data as follows:

In function `gdComplete()` the nearest neighbor for a given incomplete data record is searched with respect to present values and then utilized to complete the incomplete data record. Also function `gdKNearestNeighbors()` can be called. A nearest neighbor uniform randomly selected from returned $k$ nearest neighbors can then be used for data record completion.

For illustration inserted images show $k$ nearest neighbors for a sepal width value:






**Accelerated training of generative models:**

Training of generative models can be accelerated by using graphics processing units for computation. For this kind of training a software service is available. For access details contact the author of the package and see in the documentation in section Examples how training can be executed. The software service also supports trainining of generative models for mixed numerical and categorical variables.

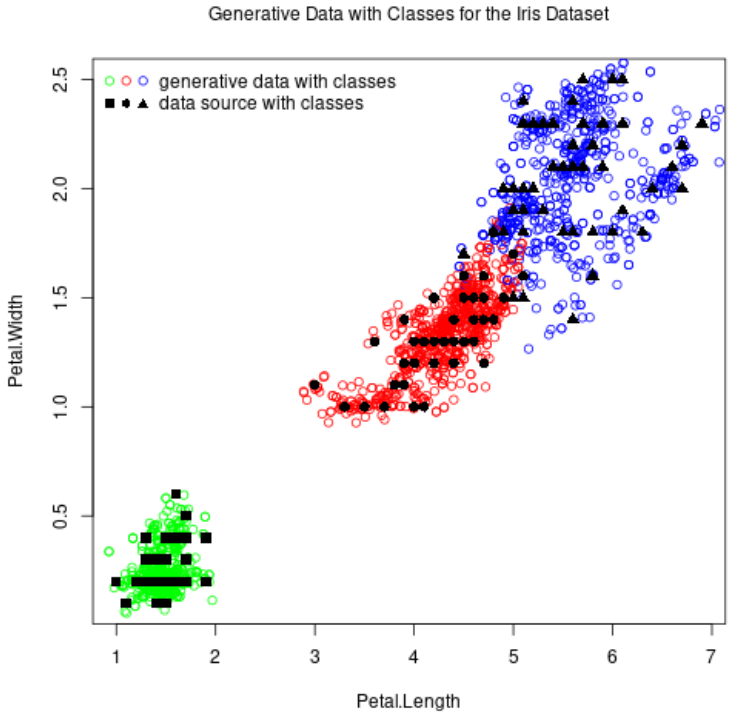
**Method of data classification:**

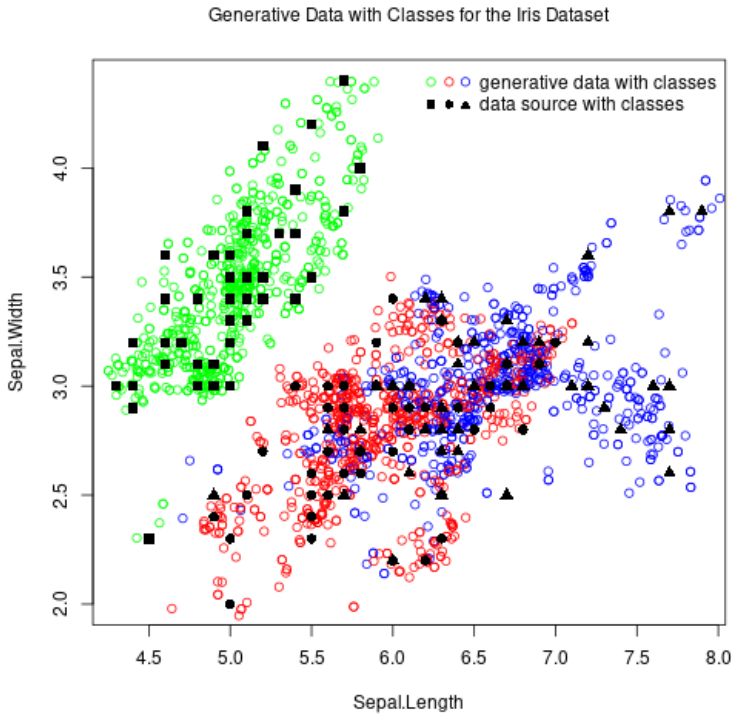
With support of training of generative models for mixed numerical and categorical variables data classification can be done for a given labeled dataset and unlabeled data as follows:

1. Considering class labels as a categorical variable train a generative model for joint distribution of data records and class labels.

2. Apply method of missing data completion to determine class labels for unlabeled data records.

For illustration inserted images show generative data for the iris dataset with data records consisting of numerical variables sepal length, sepal width, petal length, petal width and assigned class labels:


Generative Data with Classes for the Iris Dataset
generative data with classes
data source with classes
Petal.Width
Petal.Length
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7




## Details

The API includes functions for topics "definition of data source" and "generation of generative data". Main function of first topic is `dsCreateWithDataFrame()` which creates a data source with passed data frame. Main functions of second topic are `gdTrain()` which trains a generative model for a data source and `gdGenerate()` which uses a trained generative model to generate generative data. Additionally a software service for accelerated training of generative models is available.

### 1. Definition of data source

`dsCreateWithDataFrame()` Create a data source with passed data frame.

`dsActivateColumns()` Activate columns in a data source in order to include them in training of generative models. By default columns are active.

`dsDeactivateColumns()` Deactivate columns in a data source in order to exclude them from training of generative models. Note that the training function in the package supports only columns of type R-class numeric, R-type double. All columns of other type have to be deactivated. The training function in the software service for accelerated training of generative models supports columns of any type.

`dsGetActiveColumnNames()` Get names of active columns of a data source.

`dsGetInactiveColumnNames()` Get names of inactive columns of a data source.

`dsWrite()` Write created data source including settings of active columns to a file in binary format. This file will be used as input in functions of topic "generation of generative data".

`dsRead()` Read a data source from a file that was written with `dsWrite()`.

`dsGetNumberOfRows()` Get number of rows in a data source.

`dsGetRow()` Get a row in a data source.

**2. Generation of generative data**

`gdTrainParameters()` Specify parameters for training of generative model.

`gdTrain()` Read a data source from a file, train a generative model that generates generative data for the data source in iterative training steps, write trained generative model and generated data in training steps to a file in binary format..

`gdGenerateParameters()` Specify parameters for generation of generative data.

`gdGenerate()` Read a generative model and a data source from a file, generate generative data for the data source and write generated data to a file in binary format.

`gdCalculateDensityValues()` Read generative data from a file, calculate density values and write generative data with density values to original file.

`gdRead()` Read generative data and data source from specified files.

`gdPlotParameters()` Specify plot parameters for generative data.

`gdPlotDataSourceParameters()` Specify plot parameters for data source.

`gdPlotProjection()` Create an image file containing two-dimensional projections of generative data and data source.

`gdGetNumberOfRows()` Get number of rows in generative data.

`gdGetRow()` Get a row in generative data.

`gdCalculateDensityValue()` Calculate density value for a data record.

`gdCalculateDensityValueQuantile()` Calculate density value quantile for a percent value.

`gdKNearestNeighbors()` Search for k nearest neighbors in generative data.

`gdComplete()` Complete incomplete data record.

`gdWriteSubset()` Write subset of generative data.

**3. Software service for accelerated training of generative models**

`gdServiceTrain()` Send a request to software service to train a generative model.

`gdServiceGetGenerativeData()` Get generated generative data from software service.

`gdServiceGetGenerativeModel()` Get trained generative model from software service.

`gdServiveGetStatus()` Get status of generated job from software service.

`gdServiceDelete()` Delete generated job from software service.

## Author(s)

Werner Mueller

Maintainer: Werner Mueller <werner.mueller5@chello.at>

## References

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (2014), *"Generative Adversarial Nets"*, <arXiv:1406.2661v1>

## Examples

```
# Environment used for execution of examples:

# Operating system: Ubuntu 22.04.1
# Compiler: g++ 11.3.0 (supports C++17 standard)
# R applications: R 4.1.2, RStudio 2022.02.2
# Installed packages: 'Rcpp' 1.0.10, 'tensorflow' 2.11.0,
# 'ganGenerativeData' 2.0.0

# Package 'tensorflow' provides an interface to machine learning framework
# TensorFlow. To complete the installation function install_tensorflow() has to
# be called.
## Not run:
```

```
library(tensorflow)
install_tensorflow()
## End(Not run)

# Generate generative data for the iris dataset

# Load library
library(ganGenerativeData)

# 1. Definition of data source for the iris dataset

# Create a data source with iris data frame.
dsCreateWithDataFrame(iris)

# Deactivate the column with name Species and index 5 in order to exclude it in
# trainng of generative model.
dsDeactivateColumns(c(5))

# Get the active column names: Sepal.Length, Sepal.Width, Petal.Length,
# Petal.Width.
dsGetActiveColumnNames()

# Write the data source including settings of active columns to file
# "ds.bin" in binary format.
## Not run:
dsWrite("ds.bin")
## End(Not run)

# 2. Generation of generative data for the iris data source

# Read data source from file "ds.bin", train a generative model in iterative
# training steps (used number of iterations in tests is in the range of 10000 to
# 50000), write trained generative model and generated data in training steps to
# files "gm.bin" and "gd.bin".
## Not run:
gdTrain("gm.bin", "gd.bin", "ds.bin", c(1, 2),
gdTrainParameters(numberOfTrainingIterations = 1000))
## End(Not run)

# Read generative data from file "gd.bin", calculate density values and
# write generative data with density values to original file.
## Not run:
gdCalculateDensityValues("gd.bin")
## End(Not run)

# Read generative data from file "gd.bin" and data source from "ds.bin". Read in
# data will be accessed in following function calls.
## Not run:
gdRead("gd.bin", "ds.bin")
## End(Not run)

# Create an image showing two-dimensional projections of generative data and
# data source for column indices 3, 4 and write it to file "gd34d.png".
```

```
## Not run:
gdPlotProjection("gd34d.png",
"Generative Data for the Iris Dataset",
c(3, 4),
gdPlotParameters(50),
gdPlotDataSourceParameters(100))
## End(Not run)

# Create an image showing two-dimensional projections of generative data and
# data source for column indices 3, 4 with density value threshold 0.71 and
# write it to file "gd34ddv.png".
## Not run:
gdPlotProjection("gd34ddv.png",
"Generative Data with a Density Value Threshold for the Iris Dataset",
c(3, 4),
gdPlotParameters(50, c(0.71), c("red", "green")),
gdPlotDataSourceParameters(100))
## End(Not run)

# Get number of rows in generative data
## Not run:
gdGetNumberOfRows()
## End(Not run)

# Get row with index 1000 in generative data
## Not run:
gdGetRow(1000)
## End(Not run)

# Calculate density value for a data record
## Not run:
gdCalculateDensityValue(list(6.1, 2.6, 5.6, 1.4))
## End(Not run)

# Calculate density value quantile for 50 percent
## Not run:
gdCalculateDensityValueQuantile(50)
## End(Not run)

# Search for k nearest neighbors for a data record
## Not run:
gdKNearestNeighbors(list(5.1, 3.5, 1.4, 0.2), 3)
## End(Not run)

# Complete incomplete data record containing an NA value
## Not run:
gdComplete(list(5.1, 3.5, 1.4, NA))
## End(Not run)

# Write subset containing 50 percent of randomly selected rows of
# generative data
## Not run:
gdRead("gd.bin")
```

```
gdWriteSubset("gds.bin", 50)
## End(Not run)

# 3. Usage of software service for accelerated training of a generative
# model

# Initialize variables for URL and access key.
## Not run:
url <- "http://xxx.xxx.xxx.xxx/gdService"
accessKey <- "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
## End(Not run)

# Send a request to software service to train a generative model for a data
# source. A job id will be returned.
## Not run:
trainParameters <- gdTrainParameters(numberOfTrainingIterations = 10000,
numberOfInitializationIterations = 2500)
jobId <- gdServiceTrain(url, accessKey, "gmService.bin", "gdService.bin", "ds.bin",
trainParameters)
## End(Not run)

# Get status of generated job from software service. When job is processed
# successfully status will be set to TRAINED.
## Not run:
gdServiceGetStatus(url, accessKey, jobId)
## End(Not run)

# Get generated generative data from software service for processed job
## Not run:
gdServiceGetGenerativeData(url, accessKey, jobId, "gdService.bin")
## End(Not run)

# Get trained generative model from software service for processed job
## Not run:
gdServiceGetGenerativeModel(url, accessKey, jobId, "gmService.bin")
## End(Not run)
```

---

`dsActivateColumns` *Activate columns*

---

## Description

Activate columns in a data source in order to include them in training of generative models. By default columns are active.

## Usage

```
dsActivateColumns(columnVector)
```

### Arguments

| | |
|---|---|
| columnVector | Vector of column indices |

### Value

None

### Examples

```
dsCreateWithDataFrame(iris)
dsGetActiveColumnNames()
dsDeactivateColumns(c(5))
dsGetActiveColumnNames()
dsActivateColumns(c(5))
dsGetActiveColumnNames()
```

---

## dsCreateWithDataFrame *Create a data source with passed data frame*

---

### Description

Create a data source with passed data frame.

### Usage

```
dsCreateWithDataFrame(dataFrame)
```

### Arguments

| | |
|---|---|
| dataFrame | Name of data frame |

### Value

None

### Examples

```
dsCreateWithDataFrame(iris)
```

## dsDeactivateColumns *Deactivate columns*

### Description

Deactivate columns in a data source in order to exclude them from training of generative models. Note that the training function in the package supports only columns of type R-class numeric, R-type double. All columns of other type have to be deactivated. The training function in the software service for accelerated training of generative models supports columns of any type.

### Usage

```
dsDeactivateColumns(columnVector)
```

### Arguments

| | |
|---|---|
| columnVector | Vector of column indices |

### Value

None

### Examples

```
dsCreateWithDataFrame(iris)
dsDeactivateColumns(c(5))
dsGetInactiveColumnNames()
```

## dsGetActiveColumnNames *Get active column names*

### Description

Get active column names of a data source

### Usage

```
dsGetActiveColumnNames()
```

### Value

Vector of names of active columns

### Examples

```
dsCreateWithDataFrame(iris)
dsDeactivateColumns(c(5))
dsGetActiveColumnNames()
```

## dsGetInactiveColumnNames *Get inactive column names*

### Description

Get inactive column names of a data source

### Usage

```
dsGetInactiveColumnNames()
```

### Value

Vector of names of inactive columns

### Examples

```
dsCreateWithDataFrame(iris)
dsDeactivateColumns(c(5))
dsGetInactiveColumnNames()
```

## dsGetNumberOfRows *Get number of rows*

### Description

Get number of rows in a data source

### Usage

```
dsGetNumberOfRows()
```

### Value

Number of rows

### Examples

```
dsCreateWithDataFrame(iris)
dsGetNumberOfRows()
```

## dsGetRow *Get a row in a data source*

### Description

Get a row in a data source for a row index.

### Usage

```
dsGetRow(index)
```

### Arguments

| | |
|---|---|
| index | Index of row |

### Value

List containing row in data source

### Examples

```
dsCreateWithDataFrame(iris)
dsGetRow(1)
```

## dsRead *Read a data source from file*

### Description

Read a data source from a file in binary format

### Usage

```
dsRead(fileName)
```

### Arguments

| | |
|---|---|
| fileName | Name of data source file |

### Value

None

## Examples

```
## Not run:
dsCreateWithDataFrame(iris)
dsDeactivateColumns(c(5))
dsWrite("ds.bin")
dsRead("ds.bin")
## End(Not run)
```

---

`dsWrite` *Write a data source to file*

---

## Description

Write a data source including settings of active columns to a file in binary format. This file will be used as input in functions for generation of generative data.

## Usage

```
dsWrite(fileName)
```

## Arguments

| | |
|---|---|
| `fileName` | Name of data source file |

## Value

None

## Examples

```
## Not run:
dsCreateWithDataFrame(iris)
dsDeactivateColumns(c(5))
dsWrite("ds.bin")
## End(Not run)
```

gdCalculateDensityValue *Calculate density value for a data record*

## Description

Calculate density value for a data record. By default for the calculation a linear search is performed on generative data. When a search tree is used search is performed on a tree for generative data which is built once in the first function call.

## Usage

```
gdCalculateDensityValue(dataRecord, useSearchTree = FALSE)
```

## Arguments

| | |
|---|---|
| dataRecord | List containing a data record |
| useSearchTree | Boolean value indicating if a search tree should be used. |

## Value

Normalized density value number

## Examples

```
## Not run:
gdRead("gd.bin")
gdCalculateDensityValue(list(6.1, 2.6, 5.6, 1.4))
## End(Not run)
```

gdCalculateDensityValueQuantile *Calculate density value quantile*

## Description

Calculate density value quantile for a percent value.

## Usage

```
gdCalculateDensityValueQuantile(percent)
```

## Arguments

| | |
|---|---|
| percent | Percent value |

## Value

Normalized density value quantile number

## Examples

```
## Not run:
gdRead("gd.bin")
gdCalculateDensityValueQuantile(50)
## End(Not run)
```

---

`gdCalculateDensityValues`
*Calculate density values for generative data*

---

## Description

Read generative data from a file, calculate density values and write generative data with density values to original file. Calculated density values are used to classiy generative data. In function gdPlotParameters() density value thresholds with assigned colors can be passed to draw generative data for different density value ranges.

## Usage

```
gdCalculateDensityValues(generativeDataFileName)
```

## Arguments

`generativeDataFileName`
Name of generative data file name

## Value

None

## Examples

```
## Not run:
gdCalculateDensityValues("gd.bin")
## End(Not run)
```

## gdComplete *Complete incomplete data record*

### Description

Search for first nearest neighbor in generative data for incomplete data record containing NA values. Found row in generative data is then used to replace NA values in inccomplete data record. This function calls gdKNearestNeighbors() with parameter k equal to 1.

### Usage

```
gdComplete(dataRecord, useSearchTree = FALSE)
```

### Arguments

| | |
|---|---|
| dataRecord | List containing incomplete data record |
| useSearchTree | Boolean value indicating if a search tree should be used. |

### Value

List containing completed data record

### Examples

```
## Not run:
gdRead("gd.bin")
gdComplete(list(5.1, 3.5, 1.4, NA))
## End(Not run)
```

## gdGenerate *Generate generative data for a data source*

### Description

Read a trained generative model for a data source, generate generative data and write generated data to a file in binary format.

### Usage

```
gdGenerate(
  generativeDataFileName,
  generativeModelFileName,
 generateParameters = gdGenerateParameters(numberOfSamples = 10000, dropout = 0.05)
)
```

### Arguments

| | |
|---|---|
| generativeDataFileName | Name of generative data file |
| generativeModelFileName | Name of generative model file |
| generateParameters | Generation of generative data parameters, see function gdGenerateParameters(). |

### Value

None

### Examples

```
## Not run:
generateParameters <- gdGenerateParameters(numberOfSamples = 10000)
gdGenerate("gd.bin", "gm.bin", generateParameters)
## End(Not run)
```

---

## gdGenerateParameters *Specify parameters for generation of generative data*

---

### Description

Specify parameters for generation of generative data. These parameters are passed to function gdGenerate().

### Usage

```
gdGenerateParameters(numberOfSamples = 1e+05, dropout = 0.05)
```

### Arguments

| | |
|---|---|
| numberOfSamples | Number of generated samples |
| dropout | Value in the range of 0 to 1. Specifies the rate of hidden units that are dropped. Dropout is a regularization method to prevent overfitting. See function gdTrainParameters(). |

### Value

List of parameters for generation of generative data

### Examples

```
## Not run:
generateParameters <- gdGenerateParameters(numberOfSamples = 100000)
## End(Not run)
```

## gdGetNumberOfRows *Get number of rows*

### Description

Get number of rows in generative data

### Usage

```
gdGetNumberOfRows()
```

### Value

Number of rows

### Examples

```
## Not run:
gdRead("gd.bin")
gdGetNumberOfRows()
## End(Not run)
```

## gdGetRow *Get a row in generative data*

### Description

Get a row in generative data for a row index

### Usage

```
gdGetRow(index)
```

### Arguments

| | |
|---|---|
| index | Index of row |

### Value

List containing row in generative data

### Examples

```
## Not run:
gdRead("gd.bin")
gdGetRow(1000)
## End(Not run)
```

## gdKNearestNeighbors *Search for k nearest neighbors*

### Description

Search for k nearest neighbors in generative data for a data record. When the data record contains NA values only the non-NA values are considered in search. By default a linear search is performed. When a search tree is used search is performed on a tree which is built once in the first function call. Building a tree is also triggered when NA values in data records change in subsequent function calls.

### Usage

```
gdKNearestNeighbors(dataRecord, k = 1L, useSearchTree = FALSE)
```

### Arguments

| | |
|---|---|
| dataRecord | List containing a data record |
| k | Number of nearest neighbors |
| useSearchTree | Boolean value indicating if a search tree should be used. |

### Value

A list of rows in generative data

### Examples

```
## Not run:
gdRead("gd.bin")
gdKNearestNeighbors(list(5.1, 3.5, 1.4, 0.2), 3)
## End(Not run)
```

## gdPlotDataSourceParameters *Specify plot parameters for data source*

### Description

Specify plot parameters for data source passed to function gdPlotProjection().

### Usage

```
gdPlotDataSourceParameters(percent = 100, color = "blue")
```

### Arguments

| | |
|---|---|
| percent | Percent of randomly selected rows in data source |
| color | Colour for data points of data source |

### Value

List of plot parameters for data source

### Examples

```
## Not run:
gdPlotDataSourceParameters(2500)
## End(Not run)
```

---

## gdPlotParameters *Specify plot parameters for generative data*

---

### Description

Specify plot parameters for generative data passed to function gdPlotProjection(). When density value thresholds with assigned colors are specified generative data is drawn for density value ranges in increasing order.

### Usage

```
gdPlotParameters(
  percent = 10,
  densityValueThresholds = c(),
  densityValueColors = c("red")
)
```

### Arguments

| | |
|---|---|
| percent | Percent of randomly selected rows in generative data |
| densityValueThresholds | Vector of density value thresholds |
| densityValueColors | Vector of colors assigned to density value thresholds. The size must be the size of densityValueThresholds plus one. |

### Value

List of plot parameters for generative data

### Examples

```
## Not run:
gdPlotParameters(50, c(0.75), c("red", "green"))
## End(Not run)
```

gdPlotProjection *Create an image file for generative data and data source*

## Description

Create an image file containing two-dimensional projections of generative data and data source. Plot pagd_2500_6.binrameters for generative data and data source are passed by functions gdPlotParameters() and gdPlotDataSourceParameters(). Data points of data source are drawn above data points of generative data.

## Usage

```
gdPlotProjection(
  imageFileName,
  title,
  columnIndices,
 generativeDataParameters = gdPlotParameters(percent = 10, densityValueThresholds = c(),
    densityValueColors = c("red")),
 dataSourceParameters = gdPlotDataSourceParameters(percent = 100, color = "blue")
)
```

## Arguments

| | |
|---|---|
| imageFileName | Name of image file |
| title | Title of image |
| columnIndices | Vector of two column indices that are used for the two-dimensional projections. Indices refer to indices of active columns of data source. |
| generativeDataParameters | Plot generative data parameters, see function gdPlotParameters(). |
| dataSourceParameters | Plot data source parameters, see function gdPlotDataSourceParameters(). |

## Value

None

## Examples

```
## Not run:
gdRead("gd.bin", "ds.bin")
gdPlotProjection("gd12ddv.png",
 "Generative Data with a Density Value Threshold for the Iris Dataset", c(1, 2),
gdPlotParameters(250000, c(0.71), c("red", "green")),
gdPlotDataSourceParameters(2500))
gdPlotProjection("gd34ddv.png",
"Generative Data with a Densit(y Value Threshold for the Iris Dataset", c(3, 4),
gdPlotParameters(250000, c(0.71), c("red", "green")),
```

```
gdPlotDataSourceParameters(2500))
## End(Not run)
```

---

gdRead *Read generative data and data source*

---

## Description

Read generative data and data source from specified files. Read in generative data and data source are accessed in gdPlot2dProjection(), generative data is accessed in gdGetRow(), gdCalculateDensityValue() and gdCalculateDensityValueQuantile().

## Usage

```
gdRead(generativeDataFileName, dataSourceFileName = "")
```

## Arguments

| | |
|---|---|
| generativeDataFileName | Name of generative data file |
| dataSourceFileName | Name of data source file |

## Value

None

## Examples

```
## Not run:
gdRead("gd.bin", "ds.bin")
## End(Not run)
```

---

gdServiceDelete *Delete a generated job from software service for accelerated training of generative models*

---

## Description

Delete a generated job from software service. If the job is currently executed it it will be stopped.

## Usage

```
gdServiceDelete(url, accessKey, jobId)
```

### Arguments

| | |
|---|---|
| url | URL of software service for accelerated training of generative models |
| accessKey | Unique key for access to software service |
| jobId | Job id for generated job for training a generative model |

### Value

None

### Examples

```
## Not run:
url <- "http://xxx.xxx.xxx.xxx/gdService"
accessKey <- "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
gdServiceDelete(url, accessKey, 1)
## End(Not run)
```

---

## gdServiceGetGenerativeData

*Get generative data from software service for accelerated training of generative models for processed job*

---

### Description

Download generated generative data from software service for processed job. The status of the job has to be TRAINED.

### Usage

```
gdServiceGetGenerativeData(
  url,
  accessKey,
  jobId,
  generativeDataFileName = NULL
)
```

### Arguments

| | |
|---|---|
| url | URL of software service for accelerated training of generative models |
| accessKey | Unique key for access to software service |
| jobId | Job id for generated job for training a generative model |
| generativeDataFileName | Name of generative data file. If name is NULL or empty string name of passed name to function gdServiceTrain() will be used. |

## Value

None

## Examples

```
## Not run:
url <- "http://xxx.xxx.xxx.xxx/gdService"
accessKey <- "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
gdServiceGetGenerativeData(url, accessKey, 1, "gd.bin")
## End(Not run)
```

---

`gdServiceGetGenerativeModel` *Get generative model from software service for accelerated training of generative models for processed job*

---

## Description

Download trained generative model from software service for processed job. The status of the job has to be TRAINED.

## Usage

```
gdServiceGetGenerativeModel(
  url,
  accessKey,
  jobId,
  generativeModelFileName = NULL
)
```

## Arguments

| | |
|---|---|
| `url` | URL of software service for accelerated training of generative models |
| `accessKey` | Unique key for access to software service |
| `jobId` | Job id for generated job for training a generative model |
| `generativeModelFileName` | Name of generative model file. If name is NULL or empty string name of passed name to function gdServiceTrain() will be used. |

## Value

None

## Examples

```
## Not run:
url <- "http://xxx.xxx.xxx.xxx/gdService"
accessKey <- "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
gdServiceGetGenerativeModel(url, accessKey, 1,  "gm.bin")
## End(Not run)
```

---

`gdServiceGetStatus` *Get status of generated job from software service for accelerated training of generative models*

---

## Description

Get status of generated job from software service. Defined status values are: CREATED, UPLOADED, TRAINING, TRAINED, DELETED, ERROR.

## Usage

```
gdServiceGetStatus(url, accessKey, jobId)
```

## Arguments

| | |
|---|---|
| `url` | URL of software service for accelerated training of generative models |
| `accessKey` | Unique key for access to software service |
| `jobId` | Job id for generated job for training a generative model |

## Value

List containing status information

## Examples

```
## Not run:
url <- "http://xxx.xxx.xxx.xxx/gdService"
accessKey <- "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
gdServiceGetStatus(url, accessKey, 1)
## End(Not run)
```

gdServiceTrain *Send a request to software service for accelerated training of generative models to train a generative model for a data source*

## Description

Send a request to software service to train a generative model. A data source file will be uploaded and a job for training will be generated. A job id for the generated job will be returned which has to be used in related requests. The job will be processed as soon as other waiting jobs are processed. When a name of an existing generative model file is passed the file will be also uploaded and the job will continue the training. See also functions gdTrain(), gdServiceGetGenerativeModel(), gdServiceGetGenerativeData(), gdServiceGetStatus(), gdServiceDelete().

## Usage

```
gdServiceTrain(
  url,
  accessKey,
  generativeModelFileName,
  generativeDataFileName,
  dataSourceFileName,
  trainParameters = gdTrainParameters(numberOfTrainingIterations = 10000,
    numberOfInitializationIterations = 2500, numberOfHiddenLayerUnits = 1024,
    learningRate = 7e-05, dropout = 0.05)
)
```

## Arguments

| | |
|---|---|
| url | URL of software service for accelerated training of generative models |
| accessKey | Unique key for access to software service |
| generativeModelFileName | Name of generative model file |
| generativeDataFileName | Name of generative data file. If name is NULL or empty string generated data will not be written to a file. |
| dataSourceFileName | Name of data source file |
| trainParameters | Generative model training parameters, see function gdTrainParameters(). |

## Value

Job Id number

## Examples

```
## Not run:
url <- "http://xxx.xxx.xxx.xxx/gdService"
accessKey <- "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
trainParameters <- gdTrainParameters(numberOfTrainingIterations = 10000)
gdServiceTrain(url, accessKey, "gm.bin", "gd.bin", "ds.bin", trainParameters)
## End(Not run)
```

---

`gdTrain` *Train a generative model for a data source*

---

## Description

Read a data source from a file, train a generative model that generates generative data for the data source in iterative training steps, write trained generative model and generated data in training steps to a file in binary format. When a higher number of iterations is used the distribution of generated data will get closer to that of the data source. When a name of an existing generative model file is passed training will be continued.

## Usage

```
gdTrain(
  generativeModelFileName,
  generativeDataFileName,
  dataSourceFileName,
  columnIndices,
  trainParameters = gdTrainParameters(numberOfTrainingIterations = 10000,
    numberOfInitializationIterations = 1500, numberOfHiddenLayerUnits = 1024,
    learningRate = 7e-05, dropout = 0.05)
)
```

## Arguments

`generativeModelFileName`
: Name of generative model file

`generativeDataFileName`
: Name of generative data file. When name is NULL or empty string generated data will not be written to a file.

`dataSourceFileName`
: Name of data source file

`columnIndices`
: Vector of two column indices that are used to plot two-dimensional projections of normalized generated generative data and data source for a training step. Indices refer to indices of active columns of data source. Plotting can be disabled by passing NULL or an empty vector.

`trainParameters`
: Generative model training parameters, see function gdTrainParameters().

## Value

None

## Examples

```
## Not run:
trainParameters <- gdTrainParameters(numberOfTrainingIterations = 10000)
gdTrain("gm.bin", "gd.bin", "ds.bin", c(1, 2), trainParameters)
## End(Not run)
```

---

`gdTrainParameters` *Specify parameters for training of generative model*

---

## Description

Specify parameters for training of neural networks used for generation of generative data. These parameters are passed to function gdTrain().

## Usage

```
gdTrainParameters(
  numberOfTrainingIterations = 10000,
  numberOfInitializationIterations = 1500,
  numberOfHiddenLayerUnits = 1024,
  learningRate = 7e-05,
  dropout = 0.05
)
```

## Arguments

| | |
|---|---|
| `numberOfTrainingIterations` | Number of training iterations |
| `numberOfInitializationIterations` | Number of initialization iterations |
| `numberOfHiddenLayerUnits` | Number of hidden layer units |
| `learningRate` | Learning rate for training of neural networks |
| `dropout` | Value in the range of 0 to 1. Specifies the rate of hidden units that are dropped. Dropout is a regularization method to prevent overfitting. |

## Value

List of parameters for training of generative model

## Examples

```
## Not run:
generateParameters <- gdGenerateParameters(numberOfTrainingIterations = 10000)
## End(Not run)
```

gdWriteSubset *Write subset of generative data*

## Description

Write subset of randomly selected rows of generative data

## Usage

```
gdWriteSubset(fileName, percent)
```

## Arguments

| | |
|---|---|
| fileName | Name of subset generative data file |
| percent | Percent of randomly selected rows |

## Value

None

## Examples

```
## Not run:
gdRead("gd.bin")
gdWriteSubset("gds.bin", 50)
## End(Not run)
```

# Index