Package 'gmum.r'

September 15, 2015

Version 0.2

Date 2015-08-05

Title Efficient C++ Implementations of SVM, GNG and CEC Models

Type Package

Author Wojciech Czarnecki, Stanislaw Jastrzebski, Marcin Data, Igor Sieradzki, Mateusz Bruno-Kaminski, Karol Jurek, Piotr Kowenzowski, Michal Pletty, Konrad Talik, Maciej Zgliczynski

Maintainer Stanislaw Jastrzebski <staszek.jastrzebski@gmail.com>

Description
Package focusing on efficiency (C++ implementations) and intuitive API. gmum.r is a close collaboration between GMUM group members (http://gmum.net) and students.

License MIT + file LICENSE

Repository CRAN

Depends ggplot2 (>= 1.0.0),
stats,
igraph,
SparseM,
httr,
Matrix,
MASS (>= 7.3),
Rcpp (>= 0.11.6)

LinkingTo Rcpp, RcppArmadillo, BH

NeedsCompilation yes

LazyData yes

Suggests car,
caret,
e1071,
klaR,
testthat,
methods,
mlbench

URL https://github.com/gmum/gmum.r

BugReports https://github.com/gmum/gmum.r/issues

R topics documented:

`calculateCentroids` *calculateCentroids*

Description

Using passed community.detection finds communities and for each community pick node with biggest betweenness score

Usage

```
calculateCentroids(object,
  community.detection.algorithm = spinglass.community)
```

Arguments

`object`	GNG object
`community.detection.algorithm`	Used algorithm from igraph package, by default spinglass.community

Details

Get centroids

Examples

```
gng <- GNG(gng.preset.sphere(100))
print(node(gng, calculateCentroids(gng)[1])$pos)
```

caret.gmumSvmLinear *Caret model representation for SVM with linear kernel*

Description

Supply as parameter "method" in the caret::train function

Usage

```
caret.gmumSvmLinear
```

Format

List of caret specific values

Examples

```
## Not run:
model <- train(Class ~ ., data = training,
method = caret.gmumSvmLinear,
preProc = c("center", "scale"),
tuneLength = 8,
trControl = fitControl,
tuneGrid = expand.grid(C=10^(c(-4:4)), gamma=10^(c(-4:4))),
core = "libsvm", # gmum.R parameter - pick library
verbosity = 0 # no outputs
)

## End(Not run)
```

caret.gmumSvmPoly *Caret model representation for SVM with linear kernel*

Description

Supply as parameter "method" in the caret::poly function

Usage

```
caret.gmumSvmPoly
```

Format

List of caret specific values

Examples

```
## Not run:
model <- train(Class ~ ., data = training,
method = caret.gmumSvmPoly,
preProc = c("center", "scale"),
tuneLength = 8,
trControl = fitControl,
tuneGrid = expand.grid(C=10^(c(-4:4)), gamma=10^(c(-4:4))),
core = "libsvm", # gmum.R parameter - pick library
verbosity = 0 # no outputs
)

## End(Not run)
```

---

caret.gmumSvmRadial *Caret model representation for SVM with radial kernel*

Description

Supply as parameter "method" in the caret::train function

Usage

```
caret.gmumSvmRadial
```

Format

List of caret specific values

Examples

```
## Not run:
model <- train(Class ~ ., data = training,
method = caret.gmumSvmRadial,
preProc = c("center", "scale"),
tuneLength = 8,
trControl = fitControl,
tuneGrid = expand.grid(C=10^(c(-4:4)), gamma=10^(c(-4:4))),
core = "libsvm", # gmum.R parameter - pick library
verbosity = 0 # no outputs
)

## End(Not run)
```

CEC *Cross-Entropy Clustering*

Description

Create CEC model object

Usage

```
CEC(x = NULL, k = 0, method.type = "standard", method.init = "kmeans++",
  params.r = 0, params.cov = matrix(0), params.centroids = NULL,
  params.mix = NULL, params.function = NULL, control.nstart = 10,
  control.eps = 0.05, control.itmax = 25, log.energy = TRUE,
  log.ncluster = TRUE, seed = NULL)
```

Arguments

x	Numeric matrix of data.
k	Initial number of clusters.
method.type	Type of clustering (Gaussian family). 1. 'diagonal' Gaussians with diagonal covariance. The clustering will try to divide the data into ellipsoid with radiuses parallel to coordinate axes 2. 'fixed_spherical' Spherical (radial) Gaussian densities (additional parameter - radius) 3. 'fixed_covariance' The clustering will have the tendency to divide the data into clusters resembling the unit circles in the Mahalanobis distance (additional parameter - covaraince matrix required) 4. 'func' Own function dependent on m and sigma (additional parameter) 5. 'standard' We divide dataset into ellipsoid-like clusters without any preferences (default) 6. 'spherical' The clustering will try to divide the data into circles of arbitrary sizes
method.init	Method to initialize clusters. 1. 'centroids' 2. 'kmeans++' 3. 'random'
params.r	Radius for spherical family.
params.cov	Covariance matrix for covariance family.
params.centroids	List of centroids.
params.mix	List of cluster with mixed Gaussian types.
params.function	User energy function

control.nstart	How many times to perform algorithm.
control.eps	What change of value should terminate algorithm.
control.itmax	Maximum number of iterations at each start.
log.energy	Records collected energy of all clusters in each iteration.
log.ncluster	Records number of clusters in each iteration.
seed	User seed

Examples

```
## Not run:
CEC(k=3, x=dataset)

CEC(k=3, x=dataset, control.nstart=10, method.type='spherical', control.eps=0.05)

CEC(k=2, x=dataset, method.type='spherical', method.init='centroids',
   params.centroids=list(c(-0.5,0.5),c(0,0)))

CEC(k=5, x=dataset, method.type='fixed_spherical', params.r=0.01,
   control.nstart=10, control.eps=0.07)

CEC(k=5, x=dataset, method.type='fixed_covariance',
   params.cov=matrix(c(0.03,0,0,0.01),2), control.nstart=10, control.eps=0.06)

CEC(k=1, x=dataset, method.type='func',
   params.function='name_of_my_own_function')

fixed_spherical_cluster_param = list(method.type = 'fixed_spherical', params.r = 0.001),
covariance_cluster_param = list(method.type = 'fixed_covariance',
   params.cov=matrix(c(0.05, 0, 0, 0.001), 2))
CEC(x = dataset, params.mix = list(covariance_cluster_param,
   fixed_spherical_cluster_param, fixed_spherical_cluster_param,
   fixed_spherical_cluster_param, fixed_spherical_cluster_param), control.nstart = 10)

p1 = list(method.type='spherical', k=3)
p2 = list(method.type='diagonal', k=2)
CEC(x=dataset, params.mix=list(p1, p2))

## End(Not run)
```

---

cec.ellipsegauss *cec.ellipsegauss*

Description

Simple dataset consisting in data drawn from set of elliptical gausses

cec.ellipsegauss.extra *cec.ellipsegauss.extra*

Description

Extra information for dataset cec.ellipsegauss (energy and cluster assignment)

cec.mouse1.spherical *cec.mouse1.spherical*

Description

Simple mouse-shaped dataset

cec.mouse1.spherical.extra *cec.mouse1.spherical.extra*

Description

Extra information for dataset cec.mouse1.extra (energy and cluster assignment)

centers *centers*

Description

Print centers of clusters

Usage

```
centers(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
centers(c)

## End(Not run)
```

clustering *clustering*

Description

Print labels assigned

Print labels assigned

Gets vector with node indexes assigned to examples in the dataset

Usage

```
clustering(c)

## S3 method for class 'Rcpp_CecModel'
clustering(c)

## S3 method for class 'Rcpp_GNGServer'
clustering(c)
```

Arguments

c	Object with clusters

Examples

```
## Not run:
clustering(c)

## End(Not run)
## Not run:
clustering(c)

## End(Not run)
gng <- GNG(gng.preset.sphere(100))
clustering(gng)
```

convertToIGraph *convertToIGraph*

Description

Converts GNG to igraph object, where every vertex contains attributes gng.index, error, data.label and 3 first spatial coordinates (as attributes v0, v1, v2). Additionally utility attribute is present if utility GNG is used.

Usage

```
convertToIGraph(object, calculate.dist = TRUE)
```

Arguments

object	GNG object
calculate.dist	If true will calculate all n^2 distances in the graph

---

covMatrix *covMatrix*

Description

Print covariances of clusters

Usage

```
covMatrix(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
covMatrix(c)

## End(Not run)
```

---

energy *energy*

Description

Print result energy of clustering found

Usage

```
energy(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
energy(c)

## End(Not run)
```

`errorStatistics` *errorStatistics*

Description

Gets vector with errors for every second of execution

Usage

```
errorStatistics(object)
```

Arguments

`object`	GNG object

Examples

```
gng <- GNG(gng.preset.sphere(100))
errorStatistics(gng)
```

`findClosests` *findClosests*

Description

Finds closest node from given list to vector. Often used together with calculateCentroids

Usage

```
findClosests(object, node.ids, x)
```

Arguments

`object`	GNG object
`node.ids`	List of indexes of nodes in gng.
`x`	Can be either `vector` or `data.frame`.

Details

Find closest node

Examples

```
gng <- GNG(gng.preset.sphere(100))
# Find closest centroid to c(1,1,1)
found.centroids <- calculateCentroids(gng)
findClosests(gng, found.centroids, c(1,1,1))
```

`get.wine.dataset.X` *get.wine.dataset.X*

Description

Retrieves wine dataset design matrix from UCI repository

Usage

```
get.wine.dataset.X(scale = TRUE)
```

Arguments

`scale`	if TRUE will perform feature scaling

`get.wine.dataset.y` *get.wine.dataset.y*

Description

Retrieves wine dataset labels from UCI repository

Usage

```
get.wine.dataset.y()
```

getDataset *getDataset*

Description

Print input dataset

Usage

```
getDataset(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
getDataset(c)

## End(Not run)
```

GNG *Constructor of GrowingNeuralGas object.*

Description

Construct GNG object. Can be used to train offline, or online.

Usage

```
GNG(x = NULL, labels = c(), beta = 0.99, alpha = 0.5,
  max.nodes = 1000, eps.n = 6e-04, eps.w = 0.05, max.edge.age = 200,
  train.online = FALSE, max.iter = 200, dim = -1,
  min.improvement = 0.001, lambda = 200, verbosity = 0, seed = -1,
  k = NULL)
```

Arguments

x	Passed data (matrix of data.frame) for offline training
labels	Every example can be associated with labels that are added to nodes later. By default empty
beta	Decrease the error variables of all node nodes by this fraction (forgetting rate). Default 0.99

alpha	Decrease the error variables of the nodes neighboring to the newly inserted node by this fraction. Default 0.5
max.nodes	Maximum number of nodes (after reaching this size it will continue running, but won't add new nodes)
eps.n	How strongly adapt neighbour node. Default 0.0006
eps.w	How strongly adapt winning node. Default 0.05
max.edge.age	Maximum edge age. Decrease to increase speed of change of graph topology. Default 200
train.online	default FALSE. If used will run in online fashion
max.iter	Uf training offline will stop if exceedes max.iter iterations. Default 200
dim	Used for training online, specifies training example size
min.improvement	Used for offline (default) training. Controls stopping criterion, decrease if training stops too early. Default 1e-3
lambda	Every lambda iteration is added new vertex. Default 200
verbosity	How verbose should the process be, as integer from $[0, 6]$, default: 0
seed	Seed for internal randomization
k	Utility constant, by default turned off. Good value is 1.3. Constant controlling speed of erasing obsolete nodes, see http://sund.de/netze/applets/gng/full/tex/DemoGNG/node20.html

Examples

```
X <- gng.preset.sphere(100)
y <- round(runif(100))
# Train in an offline manner
gng <- GNG(X, labels=y, max.nodes=20)
# Plot
plot(gng)

# Train in an online manner with utility (erasing obsolete nodes)
gng <- GNG(max.nodes=20, train.online=TRUE, k=1.3, dim=3)
insertExamples(gng, X, labels=y)
run(gng)
Sys.sleep(10)
terminate(gng)
# Plot
plot(gng)
```

gng.plot.layout.v2d *Use first two spatial coordinates as position in layout*

Description

Use first two spatial coordinates as position in layout

Usage

```
gng.plot.layout.v2d(g)
```

Arguments

g	GNG object

Note

You can pass any igraph layout algorithm to plot

gng.preset.cube *gng.preset.cube*

Description

Generate sample cube dataset

Usage

```
gng.preset.cube(N, r = 0.5, center = c(0.5, 0.5, 0.5))
```

Arguments

N	Number of points
r	Length of the side of cube
center	Center of the plane

Details

Cube preset dataset

Examples

```
X <- gng.preset.cube(100)
gng <- GNG(X)
```

gng.preset.plane *gng.preset.plane*

Description

Generate sample plane dataset

Usage

```
gng.preset.plane(N, side = 0.5, center = c(0.5, 0.5, 0.5))
```

Arguments

N	Number of points
side	Length of the side of plane
center	Center of the plane

Details

Plane preset dataset

Examples

```
X <- gng.preset.plane(100)
gng <- GNG(X)
```

gng.preset.sphere *gng.preset.sphere*

Description

Generate sample sphere dataset

Usage

```
gng.preset.sphere(N, r = 0.5, center = c(0.5, 0.5, 0.5))
```

Arguments

N	Number of points
r	Radius of the sphere
center	Center of the sphere

Details

Sphere preset dataset

Examples

```
X <- gng.preset.sphere(100)
gng <- GNG(X)
```

---

gngLoad *gngLoad*

Description

Writes model to a disk space efficient binary format.

Usage

```
gngLoad(filename)
```

Arguments

filename	Binary file location

Details

Load model from binary format

---

gngSave *gngSave*

Description

Writes model to a disk space efficient binary format.

Usage

```
gngSave(object, filename)
```

Arguments

object	GNG object
filename	File where binary will be saved

Details

Save model to binary format

insertExamples *insertExamples*

Description

Insert examples with optional labels.

Usage

```
insertExamples(object, examples, labels = c())
```

Arguments

object	GNG object
examples	matrix or data.frame with rows as examples. Note: if training online make sure number of columns matches dim parameter passed to GNG constructor.
labels	vector of labels, that will be associated with nodes in the graph. GNG will assign to each node a mean of labels of closest examples.

Note

It copies your examples twice in RAM. You might want to use object$insertExamples.

Examples

```
X <- gng.preset.sphere(100)
gng <- GNG(X, train.online=TRUE)
# Add more examples
X = gng.preset.sphere(100)
insertExamples(gng, X)
```

isRunning *isRunning*

Description

Returns TRUE if GNG object is training

Usage

```
isRunning(object)
```

Arguments

object	GNG object

Details

Check if GNG is running

Examples

```
gng <- GNG(gng.preset.sphere(100))
# FALSE, because did not pass train.online to constructor
print(isRunning(gng))
```

---

logClusters *logClusters*

Description

Print number of clusters that has been recorded at each stage of learning. Data is recorded only if you have chosen to when you created CEC model object.

Usage

```
logClusters(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
logClusters(c)

## End(Not run)
```

---

logEnergy *logEnergy*

Description

Print energy that has been recorded at each stage of learning. Data is recorded only if you have chosen to when you created CEC model object.

Usage

```
logEnergy(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
logEnergy(c)

## End(Not run)
```

`logIterations` *logIterations*

Description

Print how many iterations it took to learn CEC model

Usage

```
logIterations(c)
```

Arguments

`c`	object Trained CEC model object.

Examples

```
## Not run:
logIterations(c)

## End(Not run)
```

`meanError` *meanError*

Description

Gets mean error of the graph (note: blocks the execution, O(n))

Usage

```
meanError(object)
```

Arguments

`object`	GNG object

Examples

```
gng <- GNG(gng.preset.sphere(100))
meanError(gng)
```

MultiClassSVM-class *Class MultiClassSVM*

Description

Class MultiClassSVM defines a multiclass SVM model class.

node *node*

Description

Retrieves node from resulting graph

Usage

```
node(x, gng_id)
```

Arguments

x	GNG object
gng_id	Id of the node to retrieve. This is the id returned by functions like predict, or centroids

Details

Get GNG node

Examples

```
gng <- GNG(gng.preset.sphere(100))
print(node(gng, 10)$pos)
```

numberNodes *numberNodes*

Description

Get current number of nodes in the graph

Usage

```
numberNodes(object)
```

Arguments

object	GNG object

`OptimizedGNG` *Constructor of Optimized GrowingNeuralGas object.*

Description

Construct simplified and optimized GNG object. Can be used to train offline, or online. Data dimensionality shouldn't be too big, if it is consider using dimensionality reduction techniques.

Usage

```
OptimizedGNG(x = NULL, labels = c(), beta = 0.99, alpha = 0.5,
  max.nodes = 1000, eps.n = 6e-04, eps.w = 0.05, max.edge.age = 200,
  train.online = FALSE, max.iter = 200, dim = 0,
  min.improvement = 0.001, lambda = 200, verbosity = 0, seed = -1,
  value.range = c(0, 1))
```

Arguments

`x`	Passed data (matrix of data.frame) for offline training
`labels`	Every example can be associated with labels that are added to nodes later. By default empty
`beta`	Decrease the error variables of all node nodes by this fraction (forgetting rate). Default 0.99
`alpha`	Decrease the error variables of the nodes neighboring to the newly inserted node by this fraction. Default 0.5
`max.nodes`	Maximum number of nodes (after reaching this size it will continue running, but new noes won't be added)
`eps.n`	Strength of adaptation of neighbour node. Default `0.0006`
`eps.w`	Strength of adaptation of winning node. Default `0.05`
`max.edge.age`	Maximum edge age. Decrease to increase speed of change of graph topology. Default `200`
`train.online`	If used will run in online fashion. Default `FALSE`
`max.iter`	If training offline will stop if exceedes max.iter iterations. Default `200`
`dim`	Used for training online, specifies dataset example dimensionality
`min.improvement`	Used for offline (default) training. Controls stopping criterion, decrease if training stops too early. Default `1e-3`
`lambda`	New vertex is added every lambda iterations. Default 200
`verbosity`	How verbose should the process be, as integer from $[0, 6]$, default: `0`
`seed`	Seed for internal randomization
`value.range`	All example features should be in this range, required for optimized version of the algorithm. Default `(0,1)`

Examples

```
## Not run:
# Train online optimizedGNG. All values in this dataset are in the range (-4.3, 4.3)
X <- gng.preset.sphere(100)
gng <- OptimizedGNG(train.online = TRUE, value.range=c(min(X), max(X)), dim=3, max.nodes=20)
insertExamples(gng, X)
run(gng)
Sys.sleep(10)
pause(gng)

## End(Not run)
```

---

pause *pause*

Description

Pause algorithm

Usage

```
pause(object)
```

Arguments

object	GNG object

Examples

```
gng <- GNG(gng.preset.sphere(100))
pause(gng)
print(gng$isRunning())
```

---

plot.Rcpp_CecModel *plot*

Description

Plot clustering found on 2D plot coloring by cluster.

Usage

```
## S3 method for class 'Rcpp_CecModel'
plot(x, slice = c(), pca = FALSE,
  ellipses = FALSE, centers = FALSE, ...)
```

Arguments

x	CEC model object.
slice	List of dimentions chosen for display since plot is 2D.
pca	Apply PCA or not
ellipses	Outline clusters.
centers	Marks center of every cluster.
...	other arguments not used by this method.

Details

Plot CEC

Examples

```
## Not run:
plot(cec)
plot(cec, slice=c(1,3), ellipses=TRUE)
plot(cec, slice=c(1,2,3))
plot(cec, ellipses=TRUE, centers=FALSE)
plot(cec, pca=TRUE, ellipses=TRUE, centers=FALSE)

## End(Not run)
```

---

plot.Rcpp_GNGServer *plot GNG object*

Description

Plot resulting graph using igraph plotting

Usage

```
## S3 method for class 'Rcpp_GNGServer'
plot(x, vertex.color = gng.plot.color.cluster,
  layout = layout.fruchterman.reingold, mode = gng.plot.2d,
  vertex.size = 3, ...)
```

Arguments

x	GNG object
vertex.color	How to color vertexes. Possible values: "fast.cluster" (vertex color is set to fastgreedy.community clustering), "label" (rounds to integer label if present), list of integers (colors vertices according to provided list), "none" (every node is white),

layout	igraph layout to be used when plotting. Defaults to layout.fruchterman.reingold. Other good choice is using gng.plot.layout.v2d, which returns two first spatial coordinates.
mode	"2d" (igraph plot) "2d.errors" (igraph plot with mean error log plot)
vertex.size	Size of plotted vertices
...	other arguments not used by this method.

Details

Plot GNG

Note

If you want to "power-use" plotting and plot for instance a subgraph, you might be interested in exporting igraph with convertToIGraph function

Examples

```
## Not run:
gng <- GNG(scaled.wine)
# Plots igraph using first 2 coordinates and colors according to clusters
plot(gng, mode=gng.plot.2d.errors, layout=gng.plot.layout.v2d, vertex.color=gng.plot.color.cluster)

# For more possibilities see gng.plot.* constants

## End(Not run)
```

---

plot.Rcpp_SVMClient *Plot SVM object*

Description

Plots trained svm data and models disciriminative

Usage

```
## S3 method for class 'Rcpp_SVMClient'
plot(x, X = NULL, mode = "normal", cols = c(1,
  2), radius = 3, radius.max = 10, ...)
```

Arguments

x	Trained SVM object
X	Optional new data points to be predicted and plotted in one of the following formats: data.frame, data.matrix; default: NULL
mode	Which plotting mode to use as string, available are:

- `'normal'` - default mode, plots data in cols argument and a linear decision boundry in available
- `'pca'` - preforms PCA decomposition and draws data in a subspace of first 2 dimensions from the PCA
- `'contour'` - countour plot for non-linear kernels

`cols`	Data dimensions to be plotted as vector of length 2, default: `c(1,2)`
`radius`	Radius of the plotted data points as float, default: `3`
`radius.max`	Maximum radius of data points can be plotted, when model is trained with example weights as float, default: `10`
`...`	other arguments not used by this method.

Examples

```
## Not run:
# here we ause svm is a trained SVM model
plot(svm)
plot(svm, X=x, cols=c(1,3))
plot(svm, mode="pca", radius=5)

## End(Not run)
```

`predict` *predict*

Description

Classify a new point according to the model (returns index of cluster where given point belong to)

Usage

```
## S3 method for class 'Rcpp_CecModel'
predict(object, x, ...)
```

Arguments

`object`	Trained CEC model object.
`x`	Given point.
`...`	other arguments not used by this method.

predict.gng *predict*

Description

Retrieves prediction from trained GNG model

Usage

```
## S3 method for class 'Rcpp_GNGServer'
predict(object, x, ...)
```

Arguments

object	Trained model
x	Vector or matrix of examples
...	other arguments not used by this method

Details

Predict

Examples

```
gng <- GNG(gng.preset.sphere(100))
predict(gng, c(1,2,2))
```

predict.Rcpp_SVMClient *Predict using SVM object*

Description

Returns predicted classes or distance to discriminative for provided test examples.

Usage

```
## S3 method for class 'Rcpp_SVMClient'
predict(object, x_test, decision.function = FALSE,
  ...)
```

Arguments

object	Trained SVM object
x_test	Unlabeled data, in one of the following formats: data.frame, data.matrix, SparseM::matrix.csr, Matrix::Matrix, slam::simple_triplet_matrix
decision.function	Uf TRUE returns SVMs decision function (distance of a point from discriminant) instead of predicted labels, default: FALSE
...	other arguments not used by this method.

Examples

```
## Not run:
# firstly, SVM model needs to be trained
svm <- SVM(x, y, core="libsvm", kernel="linear", C=1)
# then we can use it to predict unknown samples
predcit(svm, x_test)

## End(Not run)
```

---

predictComponent *predictComponent*

Description

Finds connected component closest to given vector(s). On the first execution of function strongly connected components are calculated using igraph::cluster function.

Usage

```
predictComponent(object, x)
```

Arguments

object	GNG object
x	Can be either vector or data.frame.

Details

Find closest component

Examples

```
gng <- GNG(gng.preset.sphere(100))
# Find closest component to c(1,1,1)
predictComponent(gng,  c(1,1,1))
```

print.Rcpp_CecModel *print*

Description

Print basic information about clusters found. Presents a structure of the cec results object (clusters found)

Usage

```
## S3 method for class 'Rcpp_CecModel'
print(x, ...)
```

Arguments

x	CEC object model.
...	other arguments not used by this method.

Details

Print CEC

print.Rcpp_GNGServer *print*

Description

Print basic information about GNG object

Usage

```
## S3 method for class 'Rcpp_GNGServer'
print(x, ...)
```

Arguments

x	GNG object model.
...	other arguments not used by this method.

`print.Rcpp_SVMClient` *Print SVM object*

Description

Prints short summary of a trained model.

Usage

```
## S3 method for class 'Rcpp_SVMClient'
print(x, ...)
```

Arguments

`x`	Trained SVM object
`...`	other arguments not used by this method.

`Rcpp_CecConfiguration-class`
Class Rcpp_CecConfiguration.

Description

Class `Rcpp_CecConfiguration` defines a CEC model configuration class.

Methods

`setAlgorithm(...)` void setAlgorithm(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >)

`setCentroids(...)` void setCentroids(Rcpp::List)

`setEps(...)` void setEps(double)

`setFunction(...)` void setFunction(Rcpp::Function_Impl<Rcpp::PreserveStorage>)

`setItmax(...)` void setItmax(int)

`setLogCluster(...)` void setLogCluster(bool)

`setLogEnergy(...)` void setLogEnergy(bool)

`setMethodInit(...)` void setMethodInit(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >)

`setMethodType(...)` void setMethodType(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >)

`setMix(...)` void setMix(Rcpp::List)

`setNrOfClusters(...)` void setNrOfClusters(unsigned int)

`setNstart(...)` void setNstart(unsigned int)

`setR(...)` void setR(double)

`setSeed(...)` void setSeed(int)

Rcpp_CecModel-class *Class Rcpp_CecModel.*

Description

Class `Rcpp_CecModel` defines a CEC model class.

Methods

`centers()` std::__1::vector<arma::Row<double>, std::__1::allocator<arma::Row<double> > > centers() const

`clustering()` std::__1::vector<unsigned int, std::__1::allocator<unsigned int> > clustering() const

`covMatrix()` std::__1::vector<arma::Mat<double>, std::__1::allocator<arma::Mat<double> > > covMatrix() const

`energy()` double energy() const

`getDataset()` arma::Mat<double> getDataset()

`log.energy()` std::__1::list<double, std::__1::allocator<double> > log.energy() const

`log.iters()` unsigned int log.iters() const

`log.ncluster()` std::__1::list<unsigned int, std::__1::allocator<unsigned int> > log.ncluster() const

`runAll()` void runAll()

`runOneIteration()` void runOneIteration()

Rcpp_GNGServer-class *Class Rcpp_GNGServer.*

Description

Class `Rcpp_GNGServer` defines a GNGServer class.

Methods

`clustering()` Rcpp::NumericVector clustering()

`getCurrentIteration()` unsigned int getCurrentIteration() const

`getDatasetSize()` unsigned int getDatasetSize() const

`getErrorStatistics()` Rcpp::NumericVector getErrorStatistics()

`getMeanError()` double getMeanError()

`getNode(...)` Rcpp::List getNode(int)

`getNumberNodes()` unsigned int getNumberNodes() const

`hasStarted()` bool hasStarted() const

`insertExamples(...)` void insertExamples(Rcpp::Matrix<14, Rcpp::PreserveStorage>)

`insertLabeledExamples(...)` void insertLabeledExamples(Rcpp::Matrix<14, Rcpp::PreserveStorage>, Rcpp::NumericVector)

`isRunning()` bool isRunning() const

`nodeDistance(...)` double nodeDistance(int, int) const

`pause()` void pause()

`predict(...)` int predict(Rcpp::Vector<14, Rcpp::PreserveStorage>)

`run()` void run()

`setVerbosity(...)` void setVerbosity(int)

`terminate()` void terminate()

`Rcpp_SVMClient-class` *Class Rcpp_SVMClient.*

Description

Class `Rcpp_SVMClient` defines a SVM model class.

Methods

`alpha()` arma::Col<double> alpha()

`areExamplesWeighted()` bool areExamplesWeighted()

`bias()` double bias()

`C()` double C()

`cache()` double cache()

`coef0()` double coef0()

`core()` std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > core()

`degree()` int degree()

`gamma()` double gamma()

`getClassWeights()` arma::Col<double> getClassWeights()

`getEps()` double getEps()

`getExampleWeights()` arma::Col<double> getExampleWeights()

`getIterations()` int getIterations()

`getNumberClass()` int getNumberClass()

`getNumberSV()` int getNumberSV()

`getSV()` arma::SpMat<double> getSV()

`isProbability()` bool isProbability()

`isShrinking()` bool isShrinking()

`isSparse()` bool isSparse()

`kernel()` std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > kernel()

`prep()` std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > prep()

`setAlpha(...)` void setAlpha(arma::Col<double>)

`setBias(...)` void setBias(double)

`setC(...)` void setC(double)

`setCache(...)` void setCache(double)

`setCoef0(...)` void setCoef0(double)

`setDegree(...)` void setDegree(int)

`setEps(...)` void setEps(double)

`setGamma(...)` void setGamma(double)

`setKernel(...)` void setKernel(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >)

`setLibrary(...)` void setLibrary(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >)

`setPreprocess(...)` void setPreprocess(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >)

`setProbability(...)` void setProbability(int)

`setShrinking(...)` void setShrinking(int)

`setW(...)` void setW(arma::Col<double>)

`train()` void train()

`w()` arma::Col<double> w()

`run` *run*

Description

Run algorithm (in parallel)

Usage

```
run(object)
```

Arguments

`object`	GNG object

Examples

```
gng <- GNG(gng.preset.sphere(100))
run(gng)
print(isRunning(gng))
```

runAll *runAll*

Description

Starts whole algorithm again with same parameters

Usage

```
runAll(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
runAll(c)

## End(Not run)
```

runOneIteration *runOneIteration*

Description

runs one iteration of algorithm

Usage

```
runOneIteration(c)
```

Arguments

c	object Trained CEC model object.

Examples

```
## Not run:
runOneIteration(c)

## End(Not run)
```

summary.Rcpp_CecModel *summary*

Description

Print detailed information about CEC model object

Usage

```
## S3 method for class 'Rcpp_CecModel'
summary(object, ...)
```

Arguments

object	CEC model object.
...	other arguments not used by this method.

Details

Summary CEC

summary.Rcpp_GNGServer *summary*

Description

Print basic information about GNG object

Usage

```
## S3 method for class 'Rcpp_GNGServer'
summary(object, ...)
```

Arguments

object	GNG object model.
...	other arguments not used by this method.

Details

Summary of GNG object

summary.Rcpp_SVMClient *Summary of SVM object*

Description

Prints short summary of a trained model.

Usage

```
## S3 method for class 'Rcpp_SVMClient'
summary(object, ...)
```

Arguments

object	Trained SVM object
...	other arguments not used by this method.

SVM *Create SVM object*

Description

Create and train SVM model object.

Usage

```
SVM(x, ...)

## S3 method for class 'formula'
SVM(formula, data, ...)

## Default S3 method:
SVM(x, y, core = "libsvm", kernel = "linear",
  prep = "none", transductive.learning = FALSE,
  transductive.posratio = -1, C = 1, gamma = if (is.vector(x)) 1 else
  1/ncol(x), coef0 = 0, degree = 3, class.weights = NULL,
  example.weights = NULL, cache_size = 100, tol = 0.001, max.iter = -1,
  verbosity = 4, class.type = "one.versus.all", svm.options = "", ...)
```

Arguments

x	Training data without labels in one of the following formats: data.frame, data.matrix, SparseM::matrix.csr, Matrix::Matrix, slam::simple_triplet_matrix
...	other arguments not used by this method.
formula	Can be passed with data instead of x, y pair, formula needs to point to lables column, for example: target~.
data	Can be passed instead of x, y pair with formula to mark the labels column, supported formats are: data.frame, data.matrix
y	Labels in one of the followinf formts: factor, vector. Recommended type is factor
core	Support Vector Machine library to use in traning, available are: 'libsvm', 'svmlight'; default: 'libsvm'
kernel	Kernel type as string, available are: 'linear', 'poly', 'rbf', 'sigmoid'; default: 'linear' • linear: $x' * w$ • poly: $(gamma * x' * w + coef0)^{degree}$ • rbf: $exp(-gamma * \|x - w\|^2)$ • sigmoid: $tanh(gamma * x' * w + coef0)$
prep	Preprocess method as string, available are: 'none', '2e'; default: 'none'. For more information on 2eSVM see: http://www.sciencedirect.com/science/article/pii/S0957417414004138
transductive.learning	Option got SVM model to deduce missing labels from the dataset, default: FALSE NOTE: this feature is only available with svmlight library, missing labels are marked as 'TR', if none are found and transductive to TRUE, label 0 will be interpreted as missing
transductive.posratio	Fraction of unlabeled examples to be classified into the positive class as float from $[0, 1]$, default: the ratio of positive and negative examples in the training data
C	Cost/complexity parameter, default: 1
gamma	Parameter for poly, rbf and sigmoid kernels, default: 1/n_features
coef0	For poly and sigmoid kernels, default: 0
degree	For poly kernel, default: 3
class.weights	Named vector with weight fir each class, default: NULL
example.weights	Vector of the same length as training data with weights for each traning example, default: NULL NOTE: this feature is only supported with svmlight library
cache_size	Cache memory size in MB, default: 100
tol	Tolerance of termination criterion, default: 1e-3
max.iter	Depending on library:

- libsvm: number of iterations after which the training porcess is killed (it can end earlier is desired tolerance is met), default: `1e6`
- svmlight: number of iterations after which if there is no progress traning is killed, default: `-1` (no limit)

`verbosity` How verbose should the process be, as integer from $[1, 6]$, default: `4`

`class.type` Multiclass algorithm type as string, available are: `'one.versus.all'`, `'one.versus.one'`; default: `'one.versus.one'`

`svm.options` enables to pass all svmlight command lines arguments for more advanced options, for details see http://svmlight.joachims.org/

Value

SVM model object

Examples

```
## Not run:
# train SVM from data in x and labels in y
svm <- SVM(x, y, core="libsvm", kernel="linear", C=1)

# train SVM using a dataset with both data and lables and a formula pointing to labels
formula <- target ~ .
svm <- SVM(formula, data, core="svmlight", kernel="rbf", gamma=1e3)

# train a model with 2eSVM algorithm
data(svm_breast_cancer_dataset)
ds <- svm.breastcancer.dataset
svm.2e <- SVM(x=ds[,-1], y=ds[,1], core="libsvm", kernel="linear", prep = "2e", C=10);
# more at \url{http://r.gmum.net/samples/svm.2e.html}

# train SVM on a multiclass data set
data(iris)
# with "one vs rest" strategy
svm.ova <- SVM(Species ~ ., data=iris, class.type="one.versus.all", verbosity=0)
# or with "one vs one" strategy
svm.ovo <- SVM(x=iris[,1:4], y=iris[,5], class.type="one.versus.one", verbosity=0)

# we can use svmlights sample weighting feature, suppose we have weights vector
# with a weight for every sample in the traning data
weighted.svm <- SVM(formula=y~., data=df, core="svmlight", kernel="rbf", C=1.0,
                    gamma=0.5, example.weights=weights)

# svmlight alows us to determine missing labels from a dataset
# suppose we have a labels y with missing labels marked as zeros
svm.transduction <- SVM(x, y, transductive.learning=TRUE, core="svmlight")

# for more in-depth examples visit \url{http://r.gmum.net/getting_started.html}

## End(Not run)
```

svm.accuracy *Measure accuracy scoreof a prediction*

Description

Calculates accuracy of a prediction, returns precent of correctly predicted examples over all test examples.

Usage

```
svm.accuracy(prediction, target)
```

Arguments

prediction	factor or 1 dim vector with predicted classes
target	factor or 1 dim vector with true classes #'

Examples

```
## Not run:
# firstly, SVM model needs to be trained
svm <- SVM(x, y, core="libsvm", kernel="linear", C=1)
# then we can use it to predict unknown samples
p <- predcit(svm, x_test)
acc <- svm.accuracy(p, y)

## End(Not run)
```

svm.breastcancer.dataset *svm.breastcancer.dataset*

Description

UCI breast cancer dataset

svm.transduction *svm.transduction*

Description

Dataset used in transduction demo on website

terminate *terminate*

Description

Terminate algorithm

Usage

```
terminate(object)
```

Arguments

object	GNG object

Examples

```
gng <- GNG(gng.preset.sphere(100))
terminate(gng)
```

Tset *Tset*

Description

Simple dataset in the form of T letter

Index