Package ‘kernlab’

October 13, 2022

Version 0.9-31

Title Kernel-Based Machine Learning Lab

Description Kernel-based machine learning methods for classification,
regression, clustering, novelty detection, quantile regression
and dimensionality reduction. Among other methods 'kernlab'
includes Support Vector Machines, Spectral Clustering, Kernel
PCA, Gaussian Processes and a QP solver.

Depends R (>= 2.10)

Imports methods, stats, grDevices, graphics

LazyLoad Yes

License GPL-2

SystemRequirements C++11

NeedsCompilation yes

Author Alexandros Karatzoglou [aut, cre],
Alex Smola [aut],
Kurt Hornik [aut],
National ICT Australia (NICTA) [cph],
Michael A. Maniscalco [ctb, cph],
Choon Hui Teo [ctb]

Maintainer Alexandros Karatzoglou <`alexandros.karatzoglou@gmail.com`>

Repository CRAN

Date/Publication 2022-06-09 03:26:43 UTC

R topics documented:

`as.kernelMatrix` *Assing kernelMatrix class to matrix objects*

Description

`as.kernelMatrix` in package **kernlab** can be used to coerce the kernelMatrix class to matrix objects representing a kernel matrix. These matrices can then be used with the kernelMatrix interfaces which most of the functions in **kernlab** support.

Usage

```
## S4 method for signature 'matrix'
as.kernelMatrix(x, center = FALSE)
```

Arguments

`x`	matrix to be assigned the `kernelMatrix` class
`center`	center the kernel matrix in feature space (default: FALSE)

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`kernelMatrix`, `dots`

Examples

```
## Create toy data
x <- rbind(matrix(rnorm(10),,2),matrix(rnorm(10,mean=3),,2))
y <- matrix(c(rep(1,5),rep(-1,5)))

### Use as.kernelMatrix to label the cov. matrix as a kernel matrix
### which is eq. to using a linear kernel

K <- as.kernelMatrix(crossprod(t(x)))

K

svp2 <- ksvm(K, y, type="C-svc")

svp2
```

couple *Probabilities Coupling function*

Description

`couple` is used to link class-probability estimates produced by pairwise coupling in multi-class classification problems.

Usage

```
couple(probin, coupler = "minpair")
```

Arguments

`probin`	The pairwise coupled class-probability estimates
`coupler`	The type of coupler to use. Currently `minpar` and `pkpd` and `vote` are supported (see reference for more details). If `vote` is selected the returned value is a primitive estimate passed on given votes.

Details

As binary classification problems are much easier to solve many techniques exist to decompose multi-class classification problems into many binary classification problems (voting, error codes, etc.). Pairwise coupling (one against one) constructs a rule for discriminating between every pair of classes and then selecting the class with the most winning two-class decisions. By using Platt's probabilities output for SVM one can get a class probability for each of the $k(k-1)/2$ models created in the pairwise classification. The couple method implements various techniques to combine these probabilities.

Value

A matrix with the resulting probability estimates.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Ting-Fan Wu, Chih-Jen Lin, ruby C. Weng
Probability Estimates for Multi-class Classification by Pairwise Coupling
Neural Information Processing Symposium 2003
http://papers.neurips.cc/paper/2454-probability-estimates-for-multi-class-classification-by-pairwi
pdf

See Also

predict.ksvm, ksvm

Examples

```
## create artificial pairwise probabilities
pairs <- matrix(c(0.82,0.12,0.76,0.1,0.9,0.05),2)

couple(pairs)

couple(pairs, coupler="pkpd")

couple(pairs, coupler ="vote")
```

`csi` *Cholesky decomposition with Side Information*

Description

The `csi` function in **kernlab** is an implementation of an incomplete Cholesky decomposition algorithm which exploits side information (e.g., classification labels, regression responses) to compute a low rank decomposition of a kernel matrix from the data.

Usage

```
## S4 method for signature 'matrix'
csi(x, y, kernel="rbfdot", kpar=list(sigma=0.1), rank,
centering = TRUE, kappa = 0.99 ,delta = 40 ,tol = 1e-5)
```

Arguments

`x`	The data matrix indexed by row
`y`	the classification labels or regression responses. In classification y is a $m \times n$ matrix where m the number of data and n the number of classes y and y_i is 1 if the corresponding x belongs to class i.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class `kernel`, which computes the inner product in feature space between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

- `rbfdot` Radial Basis kernel function "Gaussian"
- `polydot` Polynomial kernel function
- `vanilladot` Linear kernel function
- `tanhdot` Hyperbolic tangent kernel function
- `laplacedot` Laplacian kernel function
- `besseldot` Bessel kernel function

- `anovadot` ANOVA RBF kernel function
- `splinedot` Spline kernel
- `stringdot` String kernel

The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`rank`	maximal rank of the computed kernel matrix
`centering`	if `TRUE` centering is performed (default: TRUE)
`kappa`	trade-off between approximation of K and prediction of Y (default: 0.99)
`delta`	number of columns of cholesky performed in advance (default: 40)
`tol`	minimum gain at each iteration (default: 1e-4)

Details

An incomplete cholesky decomposition calculates Z where $K = ZZ'$ K being the kernel matrix. Since the rank of a kernel matrix is usually low, Z tends to be smaller then the complete kernel matrix. The decomposed matrix can be used to create memory efficient kernel-based algorithms without the need to compute and store a complete kernel matrix in memory.
`csi` uses the class labels, or regression responses to compute a more appropriate approximation for the problem at hand considering the additional information from the response variable.

Value

An S4 object of class "csi" which is an extension of the class "matrix". The object is the decomposed kernel matrix along with the slots :

`pivots`	Indices on which pivots where done
`diagresidues`	Residuals left on the diagonal
`maxresiduals`	Residuals picked for pivoting
`predgain`	predicted gain before adding each column
`truegain`	actual gain after adding each column
`Q`	QR decomposition of the kernel matrix
`R`	QR decomposition of the kernel matrix

slots can be accessed either by `object@slot` or by accessor functions with the same name (e.g., `pivots(object)`)

Author(s)

Alexandros Karatzoglou (based on Matlab code by Francis Bach)
<alexandros.karatzoglou@ci.tuwien.ac.at>

References

Francis R. Bach, Michael I. Jordan
Predictive low-rank decomposition for kernel methods.
Proceedings of the Twenty-second International Conference on Machine Learning (ICML) 2005
http://www.di.ens.fr/~fbach/bach_jordan_csi.pdf

See Also

inchol, chol, csi-class

Examples

```
data(iris)

## create multidimensional y matrix
yind <- t(matrix(1:3,3,150))
ymat <- matrix(0, 150, 3)
ymat[yind==as.integer(iris[,5])] <- 1

datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
Z <- csi(datamatrix,ymat, kernel=rbf, rank = 30)
dim(Z)
pivots(Z)
# calculate kernel matrix
K <- crossprod(t(Z))
# difference between approximated and real kernel matrix
(K - kernelMatrix(kernel=rbf, datamatrix))[6,]
```

---

csi-class *Class "csi"*

Description

The reduced Cholesky decomposition object

Objects from the Class

Objects can be created by calls of the form `new("csi", ...)`. or by calling the `csi` function.

Slots

`.Data`**:** Object of class `"matrix"` contains the decomposed matrix

`pivots`**:** Object of class `"vector"` contains the pivots performed

`diagresidues`**:** Object of class `"vector"` contains the diagonial residues

`maxresiduals`**:** Object of class `"vector"` contains the maximum residues

predgain Object of class `"vector"` contains the predicted gain before adding each column

truegain Object of class `"vector"` contains the actual gain after adding each column

Q Object of class `"matrix"` contains Q from the QR decomposition of the kernel matrix

R Object of class `"matrix"` contains R from the QR decomposition of the kernel matrix

Extends

Class `"matrix"`, directly.

Methods

diagresidues `signature(object = "csi")`: returns the diagonial residues

maxresiduals `signature(object = "csi")`: returns the maximum residues

pivots `signature(object = "csi")`: returns the pivots performed

predgain `signature(object = "csi")`: returns the predicted gain before adding each column

truegain `signature(object = "csi")`: returns the actual gain after adding each column

Q `signature(object = "csi")`: returns Q from the QR decomposition of the kernel matrix

R `signature(object = "csi")`: returns R from the QR decomposition of the kernel matrix

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`csi`, `inchol-class`

Examples

```
data(iris)

## create multidimensional y matrix
yind <- t(matrix(1:3,3,150))
ymat <- matrix(0, 150, 3)
ymat[yind==as.integer(iris[,5])] <- 1

datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
```

```
Z <- csi(datamatrix,ymat, kernel=rbf, rank = 30)
dim(Z)
pivots(Z)
# calculate kernel matrix
K <- crossprod(t(Z))
# difference between approximated and real kernel matrix
(K - kernelMatrix(kernel=rbf, datamatrix))[6,]
```

---

dots *Kernel Functions*

Description

The kernel generating functions provided in kernlab.
The Gaussian RBF kernel $k(x, x') = \exp(-\sigma \|x - x'\|^2)$
The Polynomial kernel $k(x, x') = (scale < x, x' > +offset)^{degree}$
The Linear kernel $k(x, x') =< x, x' >$
The Hyperbolic tangent kernel $k(x, x') = \tanh(scale < x, x' > +offset)$
The Laplacian kernel $k(x, x') = \exp(-\sigma \|x - x'\|)$
The Bessel kernel $k(x, x') = (-Bessel_{(\nu+1)}^{n} \sigma \|x - x'\|^2)$
The ANOVA RBF kernel $k(x, x') = \sum_{1 \leq i_1 \ldots < i_D \leq N} \prod_{d=1}^{D} k(x_{id}, x'_{id})$ where k(x,x) is a Gaussian RBF kernel.
The Spline kernel $\prod_{d=1}^{D} 1 + x_i x_j + x_i x_j min(x_i, x_j) - \frac{x_i + x_j}{2} min(x_i, x_j)^2 + \frac{min(x_i, x_j)^3}{3}$ \ The String kernels (see `stringdot`.

Usage

```
rbfdot(sigma = 1)

polydot(degree = 1, scale = 1, offset = 1)

tanhdot(scale = 1, offset = 1)

vanilladot()

laplacedot(sigma = 1)

besseldot(sigma = 1, order = 1, degree = 1)

anovadot(sigma = 1, degree = 1)

splinedot()
```

Arguments

`sigma`	The inverse kernel width used by the Gaussian the Laplacian, the Bessel and the ANOVA kernel

`degree`	The degree of the polynomial, bessel or ANOVA kernel function. This has to be an positive integer.
`scale`	The scaling parameter of the polynomial and tangent kernel is a convenient way of normalizing patterns without the need to modify the data itself
`offset`	The offset used in a polynomial or hyperbolic tangent kernel
`order`	The order of the Bessel function to be used as a kernel

Details

The kernel generating functions are used to initialize a kernel function which calculates the dot (inner) product between two feature vectors in a Hilbert Space. These functions can be passed as a `kernel` argument on almost all functions in **kernlab**(e.g., `ksvm`, `kpca` etc).

Although using one of the existing kernel functions as a `kernel` argument in various functions in **kernlab** has the advantage that optimized code is used to calculate various kernel expressions, any other function implementing a dot product of class `kernel` can also be used as a kernel argument. This allows the user to use, test and develop special kernels for a given data set or algorithm. For details on the string kernels see `stringdot`.

Value

Return an S4 object of class `kernel` which extents the `function` class. The resulting function implements the given kernel calculating the inner (dot) product between two vectors.

`kpar`	a list containing the kernel parameters (hyperparameters) used.

The kernel parameters can be accessed by the `kpar` function.

Note

If the offset in the Polynomial kernel is set to $0$, we obtain homogeneous polynomial kernels, for positive values, we have inhomogeneous kernels. Note that for negative values the kernel does not satisfy Mercer's condition and thus the optimizers may fail.

In the Hyperbolic tangent kernel if the offset is negative the likelihood of obtaining a kernel matrix that is not positive definite is much higher (since then even some diagonal elements may be negative), hence if this kernel has to be used, the offset should always be positive. Note, however, that this is no guarantee that the kernel will be positive.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`stringdot`, `kernelMatrix` , `kernelMult`, `kernelPol`

Examples

```
rbfkernel <- rbfdot(sigma = 0.1)
rbfkernel

kpar(rbfkernel)

## create two vectors
x <- rnorm(10)
y <- rnorm(10)

## calculate dot product
rbfkernel(x,y)
```

---

gausspr *Gaussian processes for regression and classification*

Description

gausspr is an implementation of Gaussian processes for classification and regression.

Usage

```
## S4 method for signature 'formula'
gausspr(x, data=NULL, ..., subset, na.action = na.omit, scaled = TRUE)

## S4 method for signature 'vector'
gausspr(x,...)

## S4 method for signature 'matrix'
gausspr(x, y, scaled = TRUE, type= NULL, kernel="rbfdot",
          kpar="automatic", var=1, variance.model = FALSE, tol=0.0005,
          cross=0, fit=TRUE, ... , subset, na.action = na.omit)
```

Arguments

x	a symbolic description of the model to be fit or a matrix or vector when a formula interface is not used. When not using a formula x is a matrix or vector containing the variables in the model
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'gausspr' is called from.
y	a response vector with one label for each row/component of x. Can be either a factor (for classification tasks) or a numeric vector (for regression).

`type`	Type of problem. Either "classification" or "regression". Depending on whether `y` is a factor or not, the default setting for `type` is `classification` or `regression`, respectively, but can be overwritten by setting an explicit value.
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally (both `x` and `y` variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`var`	the initial noise variance, (only for regression) (default : 0.001)
`variance.model`	build model for variance or standard deviation estimation (only for regression) (default : FALSE)
`tol`	tolerance of termination criterion (default: 0.001)
`fit`	indicates whether the fitted values should be computed and included in the model or not (default: 'TRUE')
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the Mean Squared Error for regression

`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

A Gaussian process is specified by a mean and a covariance function. The mean is a function of x (which is often the zero function), and the covariance is a function $C(x, x')$ which expresses the expected covariance between the value of the function y at the points x and x'. The actual function $y(x)$ in any data modeling problem is assumed to be a single sample from this Gaussian distribution. Laplace approximation is used for the parameter estimation in gaussian processes for classification.

The predict function can return class probabilities for classification problems by setting the `type` parameter to "probabilities". For the regression setting the `type` parameter to "variance" or "sdeviation" returns the estimated variance or standard deviation at each predicted point.

Value

An S4 object of class "gausspr" containing the fitted model along with information. Accessor functions can be used to access the slots of the object which include :

`alpha`	The resulting model parameters
`error`	Training error (if fit == TRUE)

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

C. K. I. Williams and D. Barber
Bayesian classification with Gaussian processes.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(12):1342-1351, 1998
`https://homepages.inf.ed.ac.uk/ckiw/postscript/pami_final.ps.gz`

See Also

`predict.gausspr`, `rvm`, `ksvm`, `gausspr-class`, `lssvm`

Examples

```
# train model
data(iris)
test <- gausspr(Species~.,data=iris,var=2)
```

```
test
alpha(test)

# predict on the training set
predict(test,iris[,-5])
# class probabilities
predict(test, iris[,-5], type="probabilities")

# create regression data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.03)

# regression with gaussian processes
foo <- gausspr(x, y)
foo

# predict and plot
ytest <- predict(foo, x)
plot(x, y, type ="l")
lines(x, ytest, col="red")


#predict and variance
x = c(-4, -3, -2, -1,  0, 0.5, 1, 2)
y = c(-2,  0,  -0.5,1,  2, 1, 0, -1)
plot(x,y)
foo2 <- gausspr(x, y, variance.model = TRUE)
xtest <- seq(-4,2,0.2)
lines(xtest, predict(foo2, xtest))
lines(xtest,
      predict(foo2, xtest)+2*predict(foo2,xtest, type="sdeviation"),
      col="red")
lines(xtest,
      predict(foo2, xtest)-2*predict(foo2,xtest, type="sdeviation"),
      col="red")
```

`gausspr-class` *Class "gausspr"*

Description

The Gaussian Processes object class

Objects from the Class

Objects can be created by calls of the form `new("gausspr", ...)`. or by calling the `gausspr` function

Slots

`tol`**:** Object of class `"numeric"` contains tolerance of termination criteria

`kernelf`**:** Object of class `"kfunction"` contains the kernel function used

`kpar`**:** Object of class `"list"` contains the kernel parameter used

`kcall`**:** Object of class `"list"` contains the used function call

`type`**:** Object of class `"character"` contains type of problem

`terms`**:** Object of class `"ANY"` contains the terms representation of the symbolic model used (when using a formula)

`xmatrix`**:** Object of class `"input"` containing the data matrix used

`ymatrix`**:** Object of class `"output"` containing the response matrix

`fitted`**:** Object of class `"output"` containing the fitted values

`lev`**:** Object of class `"vector"` containing the levels of the response (in case of classification)

`nclass`**:** Object of class `"numeric"` containing the number of classes (in case of classification)

`alpha`**:** Object of class `"listI"` containing the computes alpha values

`alphaindex` Object of class `"list"` containing the indexes for the alphas in various classes (in multi-class problems).

`sol` Object of class `"matrix"` containing the solution to the Gaussian Process formulation, it is used to compute the variance in regression problems.

`scaling` Object of class `"ANY"` containing the scaling coefficients of the data (when case `scaled = TRUE` is used).

`nvar`**:** Object of class `"numeric"` containing the computed variance

`error`**:** Object of class `"numeric"` containing the training error

`cross`**:** Object of class `"numeric"` containing the cross validation error

`n.action`**:** Object of class `"ANY"` containing the action performed in NA

Methods

alpha `signature(object = "gausspr")`: returns the alpha vector

cross `signature(object = "gausspr")`: returns the cross validation error

error `signature(object = "gausspr")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values

kcall `signature(object = "gausspr")`: returns the call performed

kernelf `signature(object = "gausspr")`: returns the kernel function used

kpar `signature(object = "gausspr")`: returns the kernel parameter used

lev `signature(object = "gausspr")`: returns the response levels (in classification)

type `signature(object = "gausspr")`: returns the type of problem

xmatrix `signature(object = "gausspr")`: returns the data matrix used

ymatrix `signature(object = "gausspr")`: returns the response matrix used

scaling `signature(object = "gausspr")`: returns the scaling coefficients of the data (when `scaled = TRUE` is used)

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

See Also

gausspr, ksvm-class, vm-class

Examples

```
# train model
data(iris)
test <- gausspr(Species~.,data=iris,var=2)
test
alpha(test)
error(test)
lev(test)
```

---

inchol *Incomplete Cholesky decomposition*

Description

`inchol` computes the incomplete Cholesky decomposition of the kernel matrix from a data matrix.

Usage

```
inchol(x, kernel="rbfdot", kpar=list(sigma=0.1), tol = 0.001,
             maxiter = dim(x)[1], blocksize = 50, verbose = 0)
```

Arguments

`x`	The data matrix indexed by row
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class `kernel`, which computes the inner product in feature space between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function

	• `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`tol`	algorithm stops when remaining pivots bring less accuracy then `tol` (default: 0.001)
`maxiter`	maximum number of iterations and columns in Z
`blocksize`	add this many columns to matrix per iteration
`verbose`	print info on algorithm convergence

Details

An incomplete cholesky decomposition calculates Z where $K = ZZ'$ K being the kernel matrix. Since the rank of a kernel matrix is usually low, Z tends to be smaller then the complete kernel matrix. The decomposed matrix can be used to create memory efficient kernel-based algorithms without the need to compute and store a complete kernel matrix in memory.

Value

An S4 object of class "inchol" which is an extension of the class "matrix". The object is the decomposed kernel matrix along with the slots :

`pivots`	Indices on which pivots where done
`diagresidues`	Residuals left on the diagonal
`maxresiduals`	Residuals picked for pivoting

slots can be accessed either by `object@slot` or by accessor functions with the same name (e.g., `pivots(object)`)

Author(s)

Alexandros Karatzoglou (based on Matlab code by S.V.N. (Vishy) Vishwanathan and Alex Smola)
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Francis R. Bach, Michael I. Jordan
Kernel Independent Component Analysis
Journal of Machine Learning Research 3, 1-48
https://www.jmlr.org/papers/volume3/bach02a/bach02a.pdf

See Also

csi, inchol-class, chol

Examples

```
data(iris)
datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
Z <- inchol(datamatrix,kernel=rbf)
dim(Z)
pivots(Z)
# calculate kernel matrix
K <- crossprod(t(Z))
# difference between approximated and real kernel matrix
(K - kernelMatrix(kernel=rbf, datamatrix))[6,]
```

`inchol-class` *Class "inchol"*

Description

The reduced Cholesky decomposition object

Objects from the Class

Objects can be created by calls of the form `new("inchol", ...)`. or by calling the `inchol` function.

Slots

`.Data`: Object of class `"matrix"` contains the decomposed matrix

`pivots`: Object of class `"vector"` contains the pivots performed

`diagresidues`: Object of class `"vector"` contains the diagonial residues

`maxresiduals`: Object of class `"vector"` contains the maximum residues

Extends

Class `"matrix"`, directly.

Methods

diagresidues `signature(object = "inchol")`: returns the diagonial residues

maxresiduals `signature(object = "inchol")`: returns the maximum residues

pivots `signature(object = "inchol")`: returns the pivots performed

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`inchol`, `csi-class`, `csi`

Examples

```
data(iris)
datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
Z <- inchol(datamatrix,kernel=rbf)
dim(Z)
pivots(Z)
diagresidues(Z)
maxresiduals(Z)
```

`income` *Income Data*

Description

Customer Income Data from a marketing survey.

Usage

```
data(income)
```

Format

A data frame with 14 categorical variables (8993 observations).

Explanation of the variable names:

1	`INCOME`	annual income of household (Personal income if single)	ordinal
2	`SEX`	sex	nominal
3	`MARITAL.STATUS`	marital status	nominal

4	`AGE`	age	ordinal
5	`EDUCATION`	educational grade	ordinal
6	`OCCUPATION`	type of work	nominal
7	`AREA`	how long the interviewed person has lived in the San Francisco/Oakland/San Jose area	ordinal
8	`DUAL.INCOMES`	dual incomes (if married)	nominal
9	`HOUSEHOLD.SIZE`	persons living in the household	ordinal
10	`UNDER18`	persons in household under 18	ordinal
11	`HOUSEHOLDER`	householder status	nominal
12	`HOME.TYPE`	type of home	nominal
13	`ETHNIC.CLASS`	ethnic classification	nominal
14	`LANGUAGE`	language most often spoken at home	nominal

Details

A total of N=9409 questionnaires containing 502 questions were filled out by shopping mall customers in the San Francisco Bay area. The dataset is an extract from this survey. It consists of 14 demographic attributes. The dataset is a mixture of nominal and ordinal variables with a lot of missing data. The goal is to predict the Anual Income of Household from the other 13 demographics attributes.

Source

Impact Resources, Inc., Columbus, OH (1987).

`inlearn` *Onlearn object initialization*

Description

Online Kernel Algorithm object `onlearn` initialization function.

Usage

```
## S4 method for signature 'numeric'
inlearn(d, kernel = "rbfdot", kpar = list(sigma = 0.1),
        type = "novelty", buffersize = 1000)
```

Arguments

`d`	the dimensionality of the data to be learned
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

- `rbfdot` Radial Basis kernel function "Gaussian"
- `polydot` Polynomial kernel function
- `vanilladot` Linear kernel function
- `tanhdot` Hyperbolic tangent kernel function
- `laplacedot` Laplacian kernel function
- `besseldot` Bessel kernel function
- `anovadot` ANOVA RBF kernel function

The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar` the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. For valid parameters for existing kernels are :

- `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot".
- `degree, scale, offset` for the Polynomial kernel "polydot"
- `scale, offset` for the Hyperbolic tangent kernel function "tanhdot"
- `sigma, order, degree` for the Bessel kernel "besseldot".
- `sigma, degree` for the ANOVA kernel "anovadot".

Hyper-parameters for user defined kernels can be passed through the `kpar` parameter as well.

`type` the type of problem to be learned by the online algorithm : `classification`, `regression`, `novelty`

`buffersize` the size of the buffer to be used

Details

The `inlearn` is used to initialize a blank `onlearn` object.

Value

The function returns an `S4` object of class `onlearn` that can be used by the `onlearn` function.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`onlearn`, `onlearn-class`

Examples

```
## create toy data set
x <- rbind(matrix(rnorm(100),,2),matrix(rnorm(100)+3,,2))
y <- matrix(c(rep(1,50),rep(-1,50)),,1)

## initialize onlearn object
on <- inlearn(2, kernel = "rbfdot", kpar = list(sigma = 0.2),
              type = "classification")

## learn one data point at the time
for(i in sample(1:100,100))
on <- onlearn(on,x[i,],y[i],nu=0.03,lambda=0.1)

sign(predict(on,x))
```

---

ipop *Quadratic Programming Solver*

Description

ipop solves the quadratic programming problem :
$\min(c' * x + 1/2 * x' * H * x)$
subject to:
$b <= A * x <= b + r$
$l <= x <= u$

Usage

```
ipop(c, H, A, b, l, u, r, sigf = 7, maxiter = 40, margin = 0.05,
     bound = 10, verb = 0)
```

Arguments

c	Vector or one column matrix appearing in the quadratic function
H	square matrix appearing in the quadratic function, or the decomposed form Z of the H matrix where Z is a nxm matrix with $n > m$ and $ZZ' = H$.
A	Matrix defining the constrains under which we minimize the quadratic function
b	Vector or one column matrix defining the constrains
l	Lower bound vector or one column matrix
u	Upper bound vector or one column matrix
r	Vector or one column matrix defining constrains
sigf	Precision (default: 7 significant figures)
maxiter	Maximum number of iterations

`margin`	how close we get to the constrains
`bound`	Clipping bound for the variables
`verb`	Display convergence information during runtime

Details

ipop uses an interior point method to solve the quadratic programming problem.
The H matrix can also be provided in the decomposed form Z where $ZZ' = H$ in that case the Sherman Morrison Woodbury formula is used internally.

Value

An S4 object with the following slots

`primal`	Vector containing the primal solution of the quadratic problem
`dual`	The dual solution of the problem
`how`	Character string describing the type of convergence

all slots can be accessed through accessor functions (see example)

Author(s)

Alexandros Karatzoglou (based on Matlab code by Alex Smola)
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

R. J. Vanderbei
LOQO: An interior point code for quadratic programming
Optimization Methods and Software 11, 451-484, 1999
`https://vanderbei.princeton.edu/ps/loqo5.pdf`

See Also

`solve.QP`, `inchol`, `csi`

Examples

```
## solve the Support Vector Machine optimization problem
data(spam)

## sample a scaled part (500 points) of the spam data set
m <- 500
set <- sample(1:dim(spam)[1],m)
x <- scale(as.matrix(spam[,-58]))[set,]
y <- as.integer(spam[set,58])
y[y==2] <- -1

##set C parameter and kernel
C <- 5
```

```
rbf <- rbfdot(sigma = 0.1)

## create H matrix etc.
H <- kernelPol(rbf,x,,y)
c <- matrix(rep(-1,m))
A <- t(y)
b <- 0
l <- matrix(rep(0,m))
u <- matrix(rep(C,m))
r <- 0

sv <- ipop(c,H,A,b,l,u,r)
sv
dual(sv)
```

ipop-class *Class "ipop"*

Description

The quadratic problem solver class

Objects from the Class

Objects can be created by calls of the form `new("ipop", ...)`. or by calling the `ipop` function.

Slots

`primal`**:** Object of class `"vector"` the primal solution of the problem

`dual`**:** Object of class `"numeric"` the dual of the problem

`how`**:** Object of class `"character"` convergence information

Methods

primal Object of class `ipop`Return the primal of the problem

dual Object of class `ipop`Return the dual of the problem

how Object of class `ipop`Return information on convergence

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`ipop`

Examples

```
## solve the Support Vector Machine optimization problem
data(spam)

## sample a scaled part (300 points) of the spam data set
m <- 300
set <- sample(1:dim(spam)[1],m)
x <- scale(as.matrix(spam[,-58]))[set,]
y <- as.integer(spam[set,58])
y[y==2] <- -1

##set C parameter and kernel
C <- 5
rbf <- rbfdot(sigma = 0.1)

## create H matrix etc.
H <- kernelPol(rbf,x,,y)
c <- matrix(rep(-1,m))
A <- t(y)
b <- 0
l <- matrix(rep(0,m))
u <- matrix(rep(C,m))
r <- 0

sv <- ipop(c,H,A,b,l,u,r)
primal(sv)
dual(sv)
how(sv)
```

---

kcca *Kernel Canonical Correlation Analysis*

Description

Computes the canonical correlation analysis in feature space.

Usage

```
## S4 method for signature 'matrix'
kcca(x, y, kernel="rbfdot", kpar=list(sigma=0.1),
gamma = 0.1, ncomps = 10, ...)
```

Arguments

x	a matrix containing data index by row
y	a matrix containing data index by row

`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a inner product in feature space between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`gamma`	regularization parameter (default : 0.1)
`ncomps`	number of canonical components (default : 10)
`...`	additional parameters for the `kpca` function

Details

The kernel version of canonical correlation analysis. Kernel Canonical Correlation Analysis (KCCA) is a non-linear extension of CCA. Given two random variables, KCCA aims at extracting the information which is shared by the two random variables. More precisely given x and y the purpose of KCCA is to provide nonlinear mappings $f(x)$ and $g(y)$ such that their correlation is maximized.

Value

An S4 object containing the following slots:

`kcor`	Correlation coefficients in feature space
`xcoef`	estimated coefficients for the `x` variables in the feature space
`ycoef`	estimated coefficients for the `y` variables in the feature space

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

References

Malte Kuss, Thore Graepel
The Geometry Of Kernel Canonical Correlation Analysis
`https://www.microsoft.com/en-us/research/publication/the-geometry-of-kernel-canonical-correlation-a`

See Also

`cancor`, `kpca`, `kfa`, `kha`

Examples

```
## dummy data
x <- matrix(rnorm(30),15)
y <- matrix(rnorm(30),15)

kcca(x,y,ncomps=2)
```

`kcca-class` *Class "kcca"*

Description

The "kcca" class

Objects from the Class

Objects can be created by calls of the form `new("kcca", ...)`. or by the calling the `kcca` function.

Slots

`kcor`**:** Object of class `"vector"` describing the correlations

`xcoef`**:** Object of class `"matrix"` estimated coefficients for the `x` variables

`ycoef`**:** Object of class `"matrix"` estimated coefficients for the `y` variables

Methods

kcor `signature(object = "kcca")`: returns the correlations

xcoef `signature(object = "kcca")`: returns the estimated coefficients for the `x` variables

ycoef `signature(object = "kcca")`: returns the estimated coefficients for the `y` variables

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

See Also

`kcca`, `kpca-class`

Examples

```
## dummy data
x <- matrix(rnorm(30),15)
y <- matrix(rnorm(30),15)

kcca(x,y,ncomps=2)
```

---

kernel-class *Class "kernel" "rbfkernel" "polykernel", "tanhkernel", "vanillakernel"*

Description

The built-in kernel classes in **kernlab**

Objects from the Class

Objects can be created by calls of the form `new("rbfkernel")`, `new{"polykernel"}`, `new{"tanhkernel"}`, `new{"vanillakernel"}`, `new{"anovakernel"}`, `new{"besselkernel"}`, `new{"laplacekernel"}`, `new{"splinekernel"}`, `new{"stringkernel"}`

or by calling the `rbfdot`, `polydot`, `tanhdot`, `vanilladot`, `anovadot`, `besseldot`, `laplacedot`, `splinedot`, `stringdot` functions etc..

Slots

`.Data`**:** Object of class `"function"` containing the kernel function

`kpar`**:** Object of class `"list"` containing the kernel parameters

Extends

Class `"kernel"`, directly. Class `"function"`, by class `"kernel"`.

Methods

kernelMatrix `signature(kernel = "rbfkernel", x = "matrix")`: computes the kernel matrix

kernelMult `signature(kernel = "rbfkernel", x = "matrix")`: computes the quadratic kernel expression

kernelPol `signature(kernel = "rbfkernel", x = "matrix")`: computes the kernel expansion

kernelFast `signature(kernel = "rbfkernel", x = "matrix"),,a`: computes parts or the full kernel matrix, mainly used in kernel algorithms where columns of the kernel matrix are computed per invocation

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`dots`

Examples

```
rbfkernel <- rbfdot(sigma = 0.1)
rbfkernel
is(rbfkernel)
kpar(rbfkernel)
```

`kernelMatrix` *Kernel Matrix functions*

Description

`kernelMatrix` calculates the kernel matrix $K_{ij} = k(x_i, x_j)$ or $K_{ij} = k(x_i, y_j)$.
`kernelPol` computes the quadratic kernel expression $H = z_i z_j k(x_i, x_j)$, $H = z_i k_j k(x_i, y_j)$.
`kernelMult` calculates the kernel expansion $f(x_i) = \sum_{i=1}^{m} z_i k(x_i, x_j)$
`kernelFast` computes the kernel matrix, identical to `kernelMatrix`, except that it also requires the squared norm of the first argument as additional input, useful in iterative kernel matrix calculations.

Usage

```
## S4 method for signature 'kernel'
kernelMatrix(kernel, x, y = NULL)

## S4 method for signature 'kernel'
kernelPol(kernel, x, y = NULL, z, k = NULL)

## S4 method for signature 'kernel'
```

```
kernelMult(kernel, x, y = NULL, z, blocksize = 256)

## S4 method for signature 'kernel'
kernelFast(kernel, x, y, a)
```

Arguments

kernel	the kernel function to be used to calculate the kernel matrix. This has to be a function of class `kernel`, i.e. which can be generated either one of the build in kernel generating functions (e.g., `rbfdot` etc.) or a user defined function of class `kernel` taking two vector arguments and returning a scalar.
x	a data matrix to be used to calculate the kernel matrix, or a list of vector when a `stringkernel` is used
y	second data matrix to calculate the kernel matrix, or a list of vector when a `stringkernel` is used
z	a suitable vector or matrix
k	a suitable vector or matrix
a	the squared norm of `x`, e.g., `rowSums(x^2)`
blocksize	the kernel expansion computations are done block wise to avoid storing the kernel matrix into memory. `blocksize` defines the size of the computational blocks.

Details

Common functions used during kernel based computations.
The `kernel` parameter can be set to any function, of class kernel, which computes the inner product in feature space between two vector arguments. **kernlab** provides the most popular kernel functions which can be initialized by using the following functions:

- `rbfdot` Radial Basis kernel function
- `polydot` Polynomial kernel function
- `vanilladot` Linear kernel function
- `tanhdot` Hyperbolic tangent kernel function
- `laplacedot` Laplacian kernel function
- `besseldot` Bessel kernel function
- `anovadot` ANOVA RBF kernel function
- `splinedot` the Spline kernel

(see example.)

`kernelFast` is mainly used in situations where columns of the kernel matrix are computed per invocation. In these cases, evaluating the norm of each row-entry over and over again would cause significant computational overhead.

Value

kernelMatrix returns a symmetric diagonal semi-definite matrix.
kernelPol returns a matrix.
kernelMult usually returns a one-column matrix.

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

See Also

rbfdot, polydot, tanhdot, vanilladot

Examples

```
## use the spam data
data(spam)
dt <- as.matrix(spam[c(10:20,3000:3010),-58])

## initialize kernel function
rbf <- rbfdot(sigma = 0.05)
rbf

## calculate kernel matrix
kernelMatrix(rbf, dt)

yt <- as.matrix(as.integer(spam[c(10:20,3000:3010),58]))
yt[yt==2] <- -1

## calculate the quadratic kernel expression
kernelPol(rbf, dt, ,yt)

## calculate the kernel expansion
kernelMult(rbf, dt, ,yt)
```

---

kfa *Kernel Feature Analysis*

Description

The Kernel Feature Analysis algorithm is an algorithm for extracting structure from possibly high-dimensional data sets. Similar to kpca a new basis for the data is found. The data can then be projected on the new basis.

Usage

```
## S4 method for signature 'formula'
kfa(x, data = NULL, na.action = na.omit, ...)

## S4 method for signature 'matrix'
kfa(x, kernel = "rbfdot", kpar = list(sigma = 0.1),
   features = 0, subset = 59, normalize = TRUE, na.action = na.omit)
```

Arguments

`x`	The data matrix indexed by row or a formula describing the model. Note, that an intercept is always included, whether given in the formula or not.
`data`	an optional data frame containing the variables in the model (when using a formula).
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes an inner product in feature space between two vector arguments. **kernlab** provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`features`	Number of features (principal components) to return. (default: 0 , all)
`subset`	the number of features sampled (used) from the data set
`normalize`	normalize the feature selected (default: TRUE)

`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

Kernel Feature analysis is similar to Kernel PCA, but instead of extracting eigenvectors of the training dataset in feature space, it approximates the eigenvectors by selecting training patterns which are good basis vectors for the training set. It works by choosing a fixed size subset of the data set and scaling it to unit length (under the kernel). It then chooses the features that maximize the value of the inner product (kernel function) with the rest of the patterns.

Value

`kfa` returns an object of class `kfa` containing the features selected by the algorithm.

`xmatrix`	contains the features selected
`alpha`	contains the sparse alpha vector

The `predict` function can be used to embed new data points into to the selected feature base.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Alex J. Smola, Olvi L. Mangasarian and Bernhard Schoelkopf
Sparse Kernel Feature Analysis
Data Mining Institute Technical Report 99-04, October 1999
`ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-04.ps`

See Also

`kpca`, `kfa-class`

Examples

```
data(promotergene)
f <- kfa(~.,data=promotergene,features=2,kernel="rbfdot",
         kpar=list(sigma=0.01))
plot(predict(f,promotergene),col=as.numeric(promotergene[,1]))
```

`kfa-class` *Class "kfa"*

Description

The class of the object returned by the Kernel Feature Analysis `kfa` function

Objects from the Class

Objects can be created by calls of the form `new("kfa", ...)` or by calling the `kfa` method. The objects contain the features along with the alpha values.

Slots

`alpha`**:** Object of class `"matrix"` containing the alpha values

`alphaindex`**:** Object of class `"vector"` containing the indexes of the selected feature

`kernelf`**:** Object of class `"kfunction"` containing the kernel function used

`xmatrix`**:** Object of class `"matrix"` containing the selected features

`kcall`**:** Object of class `"call"` containing the `kfa` function call

`terms`**:** Object of class `"ANY"` containing the formula terms

Methods

alpha `signature(object = "kfa")`: returns the alpha values

alphaindex `signature(object = "kfa")`: returns the index of the selected features

kcall `signature(object = "kfa")`: returns the function call

kernelf `signature(object = "kfa")`: returns the kernel function used

predict `signature(object = "kfa")`: used to embed more data points to the feature base

xmatrix `signature(object = "kfa")`: returns the selected features.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`kfa`, `kpca-class`

Examples

```
data(promotergene)
f <- kfa(~.,data=promotergene)
```

`kha` *Kernel Principal Components Analysis*

Description

Kernel Hebbian Algorithm is a nonlinear iterative algorithm for principal component analysis.

Usage

```
## S4 method for signature 'formula'
kha(x, data = NULL, na.action, ...)

## S4 method for signature 'matrix'
kha(x, kernel = "rbfdot", kpar = list(sigma = 0.1), features = 5,
         eta = 0.005, th = 1e-4, maxiter = 10000, verbose = FALSE,
        na.action = na.omit, ...)
```

Arguments

`x` The data matrix indexed by row or a formula describing the model. Note, that an intercept is always included, whether given in the formula or not.

`data` an optional data frame containing the variables in the model (when using a formula).

`kernel` the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes the inner product in feature space between two vector arguments (see `kernels`). **kernlab** provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

- `rbfdot` Radial Basis kernel function "Gaussian"
- `polydot` Polynomial kernel function
- `vanilladot` Linear kernel function
- `tanhdot` Hyperbolic tangent kernel function
- `laplacedot` Laplacian kernel function
- `besseldot` Bessel kernel function
- `anovadot` ANOVA RBF kernel function
- `splinedot` Spline kernel

The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar` the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are :

- `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot".
- `degree, scale, offset` for the Polynomial kernel "polydot"

- `scale, offset` for the Hyperbolic tangent kernel function "tanhdot"
- `sigma, order, degree` for the Bessel kernel "besseldot".
- `sigma, degree` for the ANOVA kernel "anovadot".

Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.

`features`	Number of features (principal components) to return. (default: 5)
`eta`	The hebbian learning rate (default : 0.005)
`th`	the smallest value of the convergence step (default : 0.0001)
`maxiter`	the maximum number of iterations.
`verbose`	print convergence every 100 iterations. (default : FALSE)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

The original form of KPCA can only be used on small data sets since it requires the estimation of the eigenvectors of a full kernel matrix. The Kernel Hebbian Algorithm iteratively estimates the Kernel Principal Components with only linear order memory complexity. (see ref. for more details)

Value

An S4 object containing the principal component vectors along with the corresponding normalization values.

`pcv`	a matrix containing the principal component vectors (column wise)
`eig`	The normalization values
`xmatrix`	The original data matrix

all the slots of the object can be accessed by accessor functions.

Note

The predict function can be used to embed new data on the new space

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Kwang In Kim, M.O. Franz and B. Schölkopf
Kernel Hebbian Algorithm for Iterative Kernel Principal Component Analysis
Max-Planck-Institut für biologische Kybernetik, Tübingen (109)
`https://is.mpg.de/fileadmin/user_upload/files/publications/pdf2302.pdf`

See Also

`kpca`, `kfa`, `kcca`, `pca`

Examples

```
# another example using the iris
data(iris)
test <- sample(1:150,70)

kpc <- kha(~.,data=iris[-test,-5],kernel="rbfdot",
           kpar=list(sigma=0.2),features=2, eta=0.001, maxiter=65)

#print the principal component vectors
pcv(kpc)

#plot the data projection on the components
plot(predict(kpc,iris[,-5]),col=as.integer(iris[,5]),
     xlab="1st Principal Component",ylab="2nd Principal Component")
```

`kha-class` *Class "kha"*

Description

The Kernel Hebbian Algorithm class

Objects objects of class "kha"

Objects can be created by calls of the form `new("kha", ...)`. or by calling the `kha` function.

Slots

`pcv`: Object of class `"matrix"` containing the principal component vectors

`eig`: Object of class `"vector"` containing the corresponding normalization values

`eskm`: Object of class `"vector"` containing the kernel sum

`kernelf`: Object of class `"kfunction"` containing the kernel function used

`kpar`: Object of class `"list"` containing the kernel parameters used

`xmatrix`: Object of class `"matrix"` containing the data matrix used

`kcall`: Object of class `"ANY"` containing the function call

`n.action`: Object of class `"ANY"` containing the action performed on NA

Methods

eig `signature(object = "kha")`: returns the normalization values

kcall `signature(object = "kha")`: returns the performed call

kernelf `signature(object = "kha")`: returns the used kernel function

pcv `signature(object = "kha")`: returns the principal component vectors

eskm `signature(object = "kha")`: returns the kernel sum

predict `signature(object = "kha")`: embeds new data

xmatrix `signature(object = "kha")`: returns the used data matrix

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`kha`, `ksvm-class`, `kcca-class`

Examples

```
# another example using the iris
data(iris)
test <- sample(1:50,20)

kpc <- kha(~.,data=iris[-test,-5], kernel="rbfdot",
           kpar=list(sigma=0.2),features=2, eta=0.001, maxiter=65)

#print the principal component vectors
pcv(kpc)
kernelf(kpc)
eig(kpc)
```

`kkmeans` *Kernel k-means*

Description

A weighted kernel version of the famous k-means algorithm.

Usage

```
## S4 method for signature 'formula'
kkmeans(x, data = NULL, na.action = na.omit, ...)

## S4 method for signature 'matrix'
```

```
kkmeans(x, centers, kernel = "rbfdot", kpar = "automatic",
        alg="kkmeans", p=1, na.action = na.omit, ...)

## S4 method for signature 'kernelMatrix'
kkmeans(x, centers, ...)

## S4 method for signature 'list'
kkmeans(x, centers, kernel = "stringdot",
        kpar = list(length=4, lambda=0.5),
        alg ="kkmeans", p = 1, na.action = na.omit, ...)
```

Arguments

`x`	the matrix of data to be clustered, or a symbolic description of the model to be fit, or a kernel Matrix of class `kernelMatrix`, or a list of character vectors.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'kkmeans' is called from.
`centers`	Either the number of clusters or a matrix of initial cluster centers. If the first a random initial partitioning is used.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a inner product in feature space between two vector arguments (see `link{kernels}`). **kernlab** provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel "Gaussian" • `polydot` Polynomial kernel • `vanilladot` Linear kernel • `tanhdot` Hyperbolic tangent kernel • `laplacedot` Laplacian kernel • `besseldot` Bessel kernel • `anovadot` ANOVA RBF kernel • `splinedot` Spline kernel • `stringdot` String kernel Setting the kernel parameter to "matrix" treats `x` as a kernel matrix calling the `kernelMatrix` interface. The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	a character string or the list of hyper-parameters (kernel parameters). The default character string `"automatic"` uses a heuristic the determine a suitable value for the width parameter of the RBF kernel. A list can also be used containing the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot".

- `degree, scale, offset` for the Polynomial kernel "polydot"
- `scale, offset` for the Hyperbolic tangent kernel function "tanhdot"
- `sigma, order, degree` for the Bessel kernel "besseldot".
- `sigma, degree` for the ANOVA kernel "anovadot".
- `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized.

Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.

`alg`	the algorithm to use. Options currently include `kkmeans` and `kerninghan`.
`p`	a parameter used to keep the affinity matrix positive semidefinite
`na.action`	The action to perform on NA
`...`	additional parameters

Details

`kernel k-means` uses the 'kernel trick' (i.e. implicitly projecting all data into a non-linear feature space with the use of a kernel) in order to deal with one of the major drawbacks of `k-means` that is that it cannot capture clusters that are not linearly separable in input space.
The algorithm is implemented using the triangle inequality to avoid unnecessary and computational expensive distance calculations. This leads to significant speedup particularly on large data sets with a high number of clusters.
With a particular choice of weights this algorithm becomes equivalent to Kernighan-Lin, and the norm-cut graph partitioning algorithms.
The function also support input in the form of a kernel matrix or a list of characters for text clustering.
The data can be passed to the `kkmeans` function in a `matrix` or a `data.frame`, in addition `kkmeans` also supports input in the form of a kernel matrix of class `kernelMatrix` or as a list of character vectors where a string kernel has to be used.

Value

An S4 object of class `specc` which extends the class `vector` containing integers indicating the cluster to which each point is allocated. The following slots contain useful information

`centers`	A matrix of cluster centers.
`size`	The number of point in each cluster
`withinss`	The within-cluster sum of squares for each cluster
`kernelf`	The kernel function used

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Inderjit Dhillon, Yuqiang Guan, Brian Kulis
A Unified view of Kernel k-means, Spectral Clustering and Graph Partitioning
UTCS Technical Report
https://people.bu.edu/bkulis/pubs/spectral_techreport.pdf

See Also

specc, kpca, kcca

Examples

```
## Cluster the iris data set.
data(iris)

sc <- kkmeans(as.matrix(iris[,-5]), centers=3)

sc
centers(sc)
size(sc)
withinss(sc)
```

---

kmmd *Kernel Maximum Mean Discrepancy.*

Description

The Kernel Maximum Mean Discrepancy kmmd performs a non-parametric distribution test.

Usage

```
## S4 method for signature 'matrix'
kmmd(x, y, kernel="rbfdot",kpar="automatic", alpha = 0.05,
     asymptotic = FALSE, replace = TRUE, ntimes = 150, frac = 1, ...)

## S4 method for signature 'kernelMatrix'
kmmd(x, y, Kxy, alpha = 0.05,
     asymptotic = FALSE, replace = TRUE, ntimes = 100, frac = 1, ...)

## S4 method for signature 'list'
kmmd(x, y, kernel="stringdot",
     kpar = list(type = "spectrum", length = 4), alpha = 0.05,
     asymptotic = FALSE, replace = TRUE, ntimes = 150, frac = 1, ...)
```

Arguments

`x`	data values, in a `matrix`, `list`, or `kernelMatrix`
`y`	data values, in a `matrix`, `list`, or `kernelMatrix`
`Kxy`	`kernlMatrix` between x and y values (only for the kernelMatrix interface)
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. `kernlab` provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel • `stringdot` String kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". • `lenght, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized. Hyper-parameters for user defined kernels can be passed through the `kpar` parameter as well. In the case of a Radial Basis kernel function (Gaussian) kpar can also be set to the string "automatic" which uses the heuristics in 'sigest' to calculate a good 'sigma' value for the Gaussian RBF or Laplace kernel, from the data. (default = "automatic").
`alpha`	the confidence level of the test (default: 0.05)
`asymptotic`	calculate the bounds asymptotically (suitable for smaller datasets) (default: FALSE)
`replace`	use replace when sampling for computing the asymptotic bounds (default : TRUE)
`ntimes`	number of times repeating the sampling procedure (default : 150)
`frac`	fraction of points to sample (frac : 1)
`...`	additional parameters.

Details

kmmd calculates the kernel maximum mean discrepancy for samples from two distributions and conducts a test as to whether the samples are from different distributions with level alpha.

Value

An S4 object of class kmmd containing the results of whether the H0 hypothesis is rejected or not. H0 being that the samples x and y come from the same distribution. The object contains the following slots :

H0	is H0 rejected (logical)
AsympH0	is H0 rejected according to the asymptotic bound (logical)
kernelf	the kernel function used.
mmdstats	the test statistics (vector of two)
Radbound	the Rademacher bound
Asymbound	the asymptotic bound

see kmmd-class for more details.

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

References

Gretton, A., K. Borgwardt, M. Rasch, B. Schoelkopf and A. Smola
A Kernel Method for the Two-Sample-Problem
Neural Information Processing Systems 2006, Vancouver
http://papers.nips.cc/paper/3110-a-kernel-method-for-the-two-sample-problem.pdf

See Also

ksvm

Examples

```
# create data
x <- matrix(runif(300),100)
y <- matrix(runif(300)+1,100)


mmdo <- kmmd(x, y)

mmdo
```

`kmmd-class` *Class "kqr"*

Description

The Kernel Maximum Mean Discrepancy object class

Objects from the Class

Objects can be created by calls of the form `new("kmmd", ...)`. or by calling the `kmmd` function

Slots

`kernelf`**:** Object of class `"kfunction"` contains the kernel function used

`xmatrix`**:** Object of class `"kernelMatrix"` containing the data used

H0 Object of class `"logical"` contains value of : is H0 rejected (logical)

`AsympH0` Object of class `"logical"` contains value : is H0 rejected according to the asymptotic bound (logical)

`mmdstats` Object of class `"vector"` contains the test statistics (vector of two)

`Radbound` Object of class `"numeric"` contains the Rademacher bound

`Asymbound` Object of class `"numeric"` contains the asymptotic bound

Methods

kernelf `signature(object = "kmmd")`: returns the kernel function used

H0 `signature(object = "kmmd")`: returns the value of H0 being rejected

AsympH0 `signature(object = "kmmd")`: returns the value of H0 being rejected according to the asymptotic bound

mmdstats `signature(object = "kmmd")`: returns the values of the mmd statistics

Radbound `signature(object = "kmmd")`: returns the value of the Rademacher bound

Asymbound `signature(object = "kmmd")`: returns the value of the asymptotic bound

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`kmmd`,

Examples

```
# create data
x <- matrix(runif(300),100)
y <- matrix(runif(300)+1,100)


mmdo <- kmmd(x, y)

H0(mmdo)
```

`kpca` *Kernel Principal Components Analysis*

Description

Kernel Principal Components Analysis is a nonlinear form of principal component analysis.

Usage

```
## S4 method for signature 'formula'
kpca(x, data = NULL, na.action, ...)

## S4 method for signature 'matrix'
kpca(x, kernel = "rbfdot", kpar = list(sigma = 0.1),
    features = 0, th = 1e-4, na.action = na.omit, ...)

## S4 method for signature 'kernelMatrix'
kpca(x, features = 0, th = 1e-4, ...)

## S4 method for signature 'list'
kpca(x, kernel = "stringdot", kpar = list(length = 4, lambda = 0.5),
    features = 0, th = 1e-4, na.action = na.omit, ...)
```

Arguments

`x`	the data matrix indexed by row or a formula describing the model, or a kernel Matrix of class `kernelMatrix`, or a list of character vectors
`data`	an optional data frame containing the variables in the model (when using a formula).
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function

- `vanilladot` Linear kernel function
- `tanhdot` Hyperbolic tangent kernel function
- `laplacedot` Laplacian kernel function
- `besseldot` Bessel kernel function
- `anovadot` ANOVA RBF kernel function
- `splinedot` Spline kernel

The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are :

- `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot".
- `degree, scale, offset` for the Polynomial kernel "polydot"
- `scale, offset` for the Hyperbolic tangent kernel function "tanhdot"
- `sigma, order, degree` for the Bessel kernel "besseldot".
- `sigma, degree` for the ANOVA kernel "anovadot".

Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.

`features`	Number of features (principal components) to return. (default: 0 , all)
`th`	the value of the eigenvalue under which principal components are ignored (only valid when features = 0). (default : 0.0001)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

Using kernel functions one can efficiently compute principal components in high-dimensional feature spaces, related to input space by some non-linear map.
The data can be passed to the `kpca` function in a `matrix` or a `data.frame`, in addition `kpca` also supports input in the form of a kernel matrix of class `kernelMatrix` or as a list of character vectors where a string kernel has to be used.

Value

An S4 object containing the principal component vectors along with the corresponding eigenvalues.

`pcv`	a matrix containing the principal component vectors (column wise)
`eig`	The corresponding eigenvalues
`rotated`	The original data projected (rotated) on the principal components
`xmatrix`	The original data matrix

all the slots of the object can be accessed by accessor functions.

Note

The predict function can be used to embed new data on the new space

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Schoelkopf B., A. Smola, K.-R. Mueller :
Nonlinear component analysis as a kernel eigenvalue problem
Neural Computation 10, 1299-1319
`http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.1366`

See Also

`kcca`, `pca`

Examples

```
# another example using the iris
data(iris)
test <- sample(1:150,20)

kpc <- kpca(~.,data=iris[-test,-5],kernel="rbfdot",
            kpar=list(sigma=0.2),features=2)

#print the principal component vectors
pcv(kpc)

#plot the data projection on the components
plot(rotated(kpc),col=as.integer(iris[-test,5]),
     xlab="1st Principal Component",ylab="2nd Principal Component")

#embed remaining points
emb <- predict(kpc,iris[test,-5])
points(emb,col=as.integer(iris[test,5]))
```

`kpca-class` *Class "kpca"*

Description

The Kernel Principal Components Analysis class

Objects of class "kpca"

Objects can be created by calls of the form `new("kpca", ...)`. or by calling the `kpca` function.

Slots

`pcv`: Object of class `"matrix"` containing the principal component vectors

`eig`: Object of class `"vector"` containing the corresponding eigenvalues

`rotated`: Object of class `"matrix"` containing the projection of the data on the principal components

`kernelf`: Object of class `"function"` containing the kernel function used

`kpar`: Object of class `"list"` containing the kernel parameters used

`xmatrix`: Object of class `"matrix"` containing the data matrix used

`kcall`: Object of class `"ANY"` containing the function call

`n.action`: Object of class `"ANY"` containing the action performed on NA

Methods

eig `signature(object = "kpca")`: returns the eigenvalues

kcall `signature(object = "kpca")`: returns the performed call

kernelf `signature(object = "kpca")`: returns the used kernel function

pcv `signature(object = "kpca")`: returns the principal component vectors

predict `signature(object = "kpca")`: embeds new data

rotated `signature(object = "kpca")`: returns the projected data

xmatrix `signature(object = "kpca")`: returns the used data matrix

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`ksvm-class`, `kcca-class`

Examples

```
# another example using the iris
data(iris)
test <- sample(1:50,20)

kpc <- kpca(~.,data=iris[-test,-5],kernel="rbfdot",
            kpar=list(sigma=0.2),features=2)

#print the principal component vectors
pcv(kpc)
rotated(kpc)
kernelf(kpc)
eig(kpc)
```

kqr *Kernel Quantile Regression.*

Description

The Kernel Quantile Regression algorithm `kqr` performs non-parametric Quantile Regression.

Usage

```
## S4 method for signature 'formula'
kqr(x, data=NULL, ..., subset, na.action = na.omit, scaled = TRUE)

## S4 method for signature 'vector'
kqr(x,...)

## S4 method for signature 'matrix'
kqr(x, y, scaled = TRUE, tau = 0.5, C = 0.1, kernel = "rbfdot",
    kpar = "automatic", reduced = FALSE, rank = dim(x)[1]/6,
    fit = TRUE, cross = 0, na.action = na.omit)

## S4 method for signature 'kernelMatrix'
kqr(x, y, tau = 0.5, C = 0.1, fit = TRUE, cross = 0)

## S4 method for signature 'list'
kqr(x, y, tau = 0.5, C = 0.1, kernel = "strigdot",
    kpar= list(length=4, C=0.5), fit = TRUE, cross = 0)
```

Arguments

`x`	e data or a symbolic description of the model to be fit. When not using a formula x can be a matrix or vector containing the training data or a kernel matrix of class `kernelMatrix` of the training data or a list of character vectors (for use with the string kernel). Note, that the intercept is always excluded, whether given in the formula or not.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which `kqr` is called from.
`y`	a numeric vector or a column matrix containing the response.
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally (both `x` and `y` variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions. (default: TRUE)
`tau`	the quantile to be estimated, this is generally a number strictly between 0 and 1. For 0.5 the median is calculated. (default: 0.5)

`C` the cost regularization parameter. This parameter controls the smoothness of the fitted function, essentially higher values for C lead to less smooth functions.(default: 1)

`kernel` the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. `kernlab` provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

- `rbfdot` Radial Basis kernel function "Gaussian"
- `polydot` Polynomial kernel function
- `vanilladot` Linear kernel function
- `tanhdot` Hyperbolic tangent kernel function
- `laplacedot` Laplacian kernel function
- `besseldot` Bessel kernel function
- `anovadot` ANOVA RBF kernel function
- `splinedot` Spline kernel
- `stringdot` String kernel

The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar` the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are :

- `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot".
- `degree, scale, offset` for the Polynomial kernel "polydot"
- `scale, offset` for the Hyperbolic tangent kernel function "tanhdot"
- `sigma, order, degree` for the Bessel kernel "besseldot".
- `sigma, degree` for the ANOVA kernel "anovadot".
- `lenght, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized.

Hyper-parameters for user defined kernels can be passed through the `kpar` parameter as well. In the case of a Radial Basis kernel function (Gaussian) kpar can also be set to the string "automatic" which uses the heuristics in 'sigest' to calculate a good 'sigma' value for the Gaussian RBF or Laplace kernel, from the data. (default = "automatic").

`reduced` use an incomplete cholesky decomposition to calculate a decomposed form Z of the kernel Matrix K (where $K = ZZ'$) and perform the calculations with Z. This might be useful when using `kqr` with large datasets since normally an n times n kernel matrix would be computed. Setting `reduced` to `TRUE` makes use of `csi` to compute a decomposed form instead and thus only a $n \times m$ matrix where $m < n$ and n the sample size is stored in memory (default: FALSE)

`rank` the rank m of the decomposed matrix calculated when using an incomplete cholesky decomposition. This parameter is only taken into account when `reduced` is `TRUE`(default : dim(x)[1]/6)

`fit`	indicates whether the fitted values should be computed and included in the model or not (default: 'TRUE')
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the Pinball loss and the for quantile regression
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters.

Details

In quantile regression a function is fitted to the data so that it satisfies the property that a portion tau of the data $y|n$ is below the estimate. While the error bars of many regression problems can be viewed as such estimates quantile regression estimates this quantity directly. Kernel quantile regression is similar to nu-Support Vector Regression in that it minimizes a regularized loss function in RKHS. The difference between nu-SVR and kernel quantile regression is in the type of loss function used which in the case of quantile regression is the pinball loss (see reference for details.). Minimizing the regularized loss boils down to a quadratic problem which is solved using an interior point QP solver `ipop` implemented in `kernlab`.

Value

An S4 object of class `kqr` containing the fitted model along with information.Accessor functions can be used to access the slots of the object which include :

`alpha`	The resulting model parameters which can be also accessed by `coef`.
`kernelf`	the kernel function used.
`error`	Training error (if fit == TRUE)

see `kqr-class` for more details.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Ichiro Takeuchi, Quoc V. Le, Timothy D. Sears, Alexander J. Smola
Nonparametric Quantile Estimation
Journal of Machine Learning Research 7,2006,1231-1264
`https://www.jmlr.org/papers/volume7/takeuchi06a/takeuchi06a.pdf`

See Also

predict.kqr, kqr-class, ipop, rvm, ksvm

Examples

```
# create data
x <- sort(runif(300))
y <- sin(pi*x) + rnorm(300,0,sd=exp(sin(2*pi*x)))

# first calculate the median
qrm <- kqr(x, y, tau = 0.5, C=0.15)

# predict and plot
plot(x, y)
ytest <- predict(qrm, x)
lines(x, ytest, col="blue")

# calculate 0.9 quantile
qrm <- kqr(x, y, tau = 0.9, kernel = "rbfdot",
           kpar= list(sigma=10), C=0.15)
ytest <- predict(qrm, x)
lines(x, ytest, col="red")

# calculate 0.1 quantile
qrm <- kqr(x, y, tau = 0.1,C=0.15)
ytest <- predict(qrm, x)
lines(x, ytest, col="green")

# print first 10 model coefficients
coef(qrm)[1:10]
```

---

kqr-class *Class "kqr"*

Description

The Kernel Quantile Regression object class

Objects from the Class

Objects can be created by calls of the form `new("kqr", ...)`. or by calling the `kqr` function

Slots

`kernelf`: Object of class "kfunction" contains the kernel function used

`kpar`: Object of class "list" contains the kernel parameter used

`coef`: Object of class "ANY" containing the model parameters

`param`: Object of class "list" contains the cost parameter C and tau parameter used `kcall`**:** Object of class `"list"` contains the used function call

`terms`**:** Object of class `"ANY"` contains the terms representation of the symbolic model used (when using a formula)

`xmatrix`**:** Object of class `"input"` containing the data matrix used

`ymatrix`**:** Object of class `"output"` containing the response matrix

`fitted`**:** Object of class `"output"` containing the fitted values

`alpha`**:** Object of class `"listI"` containing the computes alpha values

`b`**:** Object of class `"numeric"` containing the offset of the model.

`scaling` Object of class `"ANY"` containing the scaling coefficients of the data (when case `scaled = TRUE` is used).

`error`**:** Object of class `"numeric"` containing the training error

`cross`**:** Object of class `"numeric"` containing the cross validation error

`n.action`**:** Object of class `"ANY"` containing the action performed in NA

`nclass`**:** Inherited from class `vm`, not used in kqr

`lev`**:** Inherited from class `vm`, not used in kqr

`type`**:** Inherited from class `vm`, not used in kqr

Methods

coef `signature(object = "kqr")`: returns the coefficients (alpha) of the model

alpha `signature(object = "kqr")`: returns the alpha vector (identical to `coef`)

b `signature(object = "kqr")`: returns the offset beta of the model.

cross `signature(object = "kqr")`: returns the cross validation error

error `signature(object = "kqr")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values

kcall `signature(object = "kqr")`: returns the call performed

kernelf `signature(object = "kqr")`: returns the kernel function used

kpar `signature(object = "kqr")`: returns the kernel parameter used

param `signature(object = "kqr")`: returns the cost regularization parameter C and tau used

xmatrix `signature(object = "kqr")`: returns the data matrix used

ymatrix `signature(object = "kqr")`: returns the response matrix used

scaling `signature(object = "kqr")`: returns the scaling coefficients of the data (when `scaled = TRUE` is used)

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`kqr`, `vm-class`, `ksvm-class`

Examples

```
# create data
x <- sort(runif(300))
y <- sin(pi*x) + rnorm(300,0,sd=exp(sin(2*pi*x)))

# first calculate the median
qrm <- kqr(x, y, tau = 0.5, C=0.15)

# predict and plot
plot(x, y)
ytest <- predict(qrm, x)
lines(x, ytest, col="blue")

# calculate 0.9 quantile
qrm <- kqr(x, y, tau = 0.9, kernel = "rbfdot",
           kpar = list(sigma = 10), C = 0.15)
ytest <- predict(qrm, x)
lines(x, ytest, col="red")

# print model coefficients and other information
coef(qrm)
b(qrm)
error(qrm)
kernelf(qrm)
```

---

ksvm *Support Vector Machines*

Description

Support Vector Machines are an excellent tool for classification, novelty detection, and regression. ksvm supports the well known C-svc, nu-svc, (classification) one-class-svc (novelty) eps-svr, nu-svr (regression) formulations along with native multi-class classification formulations and the bound-constraint SVM formulations.
ksvm also supports class-probabilities output and confidence intervals for regression.

Usage

```
## S4 method for signature 'formula'
ksvm(x, data = NULL, ..., subset, na.action = na.omit, scaled = TRUE)

## S4 method for signature 'vector'
ksvm(x, ...)

## S4 method for signature 'matrix'
ksvm(x, y = NULL, scaled = TRUE, type = NULL,
     kernel ="rbfdot", kpar = "automatic",
     C = 1, nu = 0.2, epsilon = 0.1, prob.model = FALSE,
     class.weights = NULL, cross = 0, fit = TRUE, cache = 40,
     tol = 0.001, shrinking = TRUE, ...,
     subset, na.action = na.omit)

## S4 method for signature 'kernelMatrix'
ksvm(x, y = NULL, type = NULL,
     C = 1, nu = 0.2, epsilon = 0.1, prob.model = FALSE,
     class.weights = NULL, cross = 0, fit = TRUE, cache = 40,
     tol = 0.001, shrinking = TRUE, ...)

## S4 method for signature 'list'
ksvm(x, y = NULL, type = NULL,
     kernel = "stringdot", kpar = list(length = 4, lambda = 0.5),
     C = 1, nu = 0.2, epsilon = 0.1, prob.model = FALSE,
     class.weights = NULL, cross = 0, fit = TRUE, cache = 40,
     tol = 0.001, shrinking = TRUE, ...,
     na.action = na.omit)
```

Arguments

`x`	a symbolic description of the model to be fit. When not using a formula x can be a matrix or vector containing the training data or a kernel matrix of class `kernelMatrix` of the training data or a list of character vectors (for use with the string kernel). Note, that the intercept is always excluded, whether given in the formula or not.
`data`	an optional data frame containing the training data, when using a formula. By default the data is taken from the environment which 'ksvm' is called from.
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally (both `x` and `y` variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.
`type`	`ksvm` can be used for classification , for regression, or for novelty detection. Depending on whether `y` is a factor or not, the default setting for `type` is `C-svc` or `eps-svr`, respectively, but can be overwritten by setting an explicit value. Valid options are: • `C-svc` C classification • `nu-svc` nu classification • `C-bsvc` bound-constraint svm classification • `spoc-svc` Crammer, Singer native multi-class • `kbb-svc` Weston, Watkins native multi-class • `one-svc` novelty detection

- `eps-svr` epsilon regression
- `nu-svr` nu regression
- `eps-bsvr` bound-constraint svm regression

`kernel` the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes the inner product in feature space between two vector arguments (see `kernels`).
kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

- `rbfdot` Radial Basis kernel "Gaussian"
- `polydot` Polynomial kernel
- `vanilladot` Linear kernel
- `tanhdot` Hyperbolic tangent kernel
- `laplacedot` Laplacian kernel
- `besseldot` Bessel kernel
- `anovadot` ANOVA RBF kernel
- `splinedot` Spline kernel
- `stringdot` String kernel

Setting the kernel parameter to "matrix" treats `x` as a kernel matrix calling the `kernelMatrix` interface.

The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar` the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. For valid parameters for existing kernels are :

- `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot".
- `degree, scale, offset` for the Polynomial kernel "polydot"
- `scale, offset` for the Hyperbolic tangent kernel function "tanhdot"
- `sigma, order, degree` for the Bessel kernel "besseldot".
- `sigma, degree` for the ANOVA kernel "anovadot".
- `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized.

Hyper-parameters for user defined kernels can be passed through the kpar parameter as well. In the case of a Radial Basis kernel function (Gaussian) kpar can also be set to the string "automatic" which uses the heuristics in `sigest` to calculate a good `sigma` value for the Gaussian RBF or Laplace kernel, from the data. (default = "automatic").

`C` cost of constraints violation (default: 1) this is the 'C'-constant of the regularization term in the Lagrange formulation.

`nu`	parameter needed for `nu-svc`, `one-svc`, and `nu-svr`. The `nu` parameter sets the upper bound on the training error and the lower bound on the fraction of data points to become Support Vectors (default: 0.2).
`epsilon`	epsilon in the insensitive-loss function used for `eps-svr`, `nu-svr` and `eps-bsvm` (default: 0.1)
`prob.model`	if set to `TRUE` builds a model for calculating class probabilities or in case of regression, calculates the scaling parameter of the Laplacian distribution fitted on the residuals. Fitting is done on output data created by performing a 3-fold cross-validation on the training data. For details see references. (default: `FALSE`)
`class.weights`	a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named.
`cache`	cache memory in MB (default 40)
`tol`	tolerance of termination criterion (default: 0.001)
`shrinking`	option whether to use the shrinking-heuristics (default: `TRUE`)
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression
`fit`	indicates whether the fitted values should be computed and included in the model or not (default: `TRUE`)
`...`	additional parameters for the low level fitting function
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)

Details

`ksvm` uses John Platt's SMO algorithm for solving the SVM QP problem an most SVM formulations. On the `spoc-svc`, `kbb-svc`, `C-bsvc` and `eps-bsvr` formulations a chunking algorithm based on the TRON QP solver is used.
For multiclass-classification with k classes, $k > 2$, `ksvm` uses the 'one-against-one'-approach, in which $k(k-1)/2$ binary classifiers are trained; the appropriate class is found by a voting scheme, The `spoc-svc` and the `kbb-svc` formulations deal with the multiclass-classification problems by solving a single quadratic problem involving all the classes.
If the predictor variables include factors, the formula interface must be used to get a correct model matrix.
In classification when `prob.model` is `TRUE` a 3-fold cross validation is performed on the data and a sigmoid function is fitted on the resulting decision values f. The data can be passed to the `ksvm` function in a `matrix` or a `data.frame`, in addition `ksvm` also supports input in the form of a kernel matrix of class `kernelMatrix` or as a list of character vectors where a string kernel has to be used.
The `plot` function for binary classification `ksvm` objects displays a contour plot of the decision values with the corresponding support vectors highlighted.

The predict function can return class probabilities for classification problems by setting the `type` parameter to "probabilities".
The problem of model selection is partially addressed by an empirical observation for the RBF kernels (Gaussian , Laplace) where the optimal values of the $sigma$ width parameter are shown to lie in between the 0.1 and 0.9 quantile of the $\|x - x'\|$ statistics. When using an RBF kernel and setting `kpar` to "automatic", `ksvm` uses the `sigest` function to estimate the quantiles and uses the median of the values.

Value

An S4 object of class `"ksvm"` containing the fitted model, Accessor functions can be used to access the slots of the object (see examples) which include:

`alpha`	The resulting support vectors, (alpha vector) (possibly scaled).
`alphaindex`	The index of the resulting support vectors in the data matrix. Note that this index refers to the pre-processed data (after the possible effect of `na.omit` and `subset`)
`coef`	The corresponding coefficients times the training labels.
`b`	The negative intercept.
`nSV`	The number of Support Vectors
`obj`	The value of the objective function. In case of one-against-one classification this is a vector of values
`error`	Training error
`cross`	Cross validation error, (when cross > 0)
`prob.model`	Contains the width of the Laplacian fitted on the residuals in case of regression, or the parameters of the sigmoid fitted on the decision values in case of classification.

Note

Data is scaled internally by default, usually yielding better results.

Author(s)

Alexandros Karatzoglou (SMO optimizers in C++ by Chih-Chung Chang & Chih-Jen Lin)
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

- Chang Chih-Chung, Lin Chih-Jen
  LIBSVM: a library for Support Vector Machines
  https://www.csie.ntu.edu.tw/~cjlin/libsvm/
- Chih-Wei Hsu, Chih-Jen Lin
  BSVM https://www.csie.ntu.edu.tw/~cjlin/bsvm/

- J. Platt
  Probabilistic outputs for support vector machines and comparison to regularized likelihood methods
  Advances in Large Margin Classifiers, A. Smola, P. Bartlett, B. Schoelkopf and D. Schuurmans, Eds. Cambridge, MA: MIT Press, 2000.
  `http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1639`
- H.-T. Lin, C.-J. Lin and R. C. Weng
  A note on Platt's probabilistic outputs for support vector machines
  `https://www.csie.ntu.edu.tw/~htlin/paper/doc/plattprob.pdf`
- C.-W. Hsu and C.-J. Lin
  A comparison on methods for multi-class support vector machines
  IEEE Transactions on Neural Networks, 13(2002) 415-425.
  `https://www.csie.ntu.edu.tw/~cjlin/papers/multisvm.ps.gz`
- K. Crammer, Y. Singer
  On the learnability and design of output codes for multiclass prolems
  Computational Learning Theory, 35-46, 2000.
  `http://www.learningtheory.org/colt2000/papers/CrammerSinger.pdf`
- J. Weston, C. Watkins
  Multi-class support vector machines
  In M. Verleysen, Proceedings of ESANN99 Brussels, 1999
  `http://citeseer.ist.psu.edu/8884.html`

See Also

`predict.ksvm`, `ksvm-class`, `couple`

Examples

```
## simple example using the spam data set
data(spam)

## create test and training set
index <- sample(1:dim(spam)[1])
spamtrain <- spam[index[1:floor(dim(spam)[1]/2)], ]
spamtest <- spam[index[((ceiling(dim(spam)[1]/2)) + 1):dim(spam)[1]], ]

## train a support vector machine
filter <- ksvm(type~.,data=spamtrain,kernel="rbfdot",
               kpar=list(sigma=0.05),C=5,cross=3)
filter

## predict mail type on the test set
mailtype <- predict(filter,spamtest[,-58])

## Check results
table(mailtype,spamtest[,58])


## Another example with the famous iris data
```

```
data(iris)

## Create a kernel function using the build in rbfdot function
rbf <- rbfdot(sigma=0.1)
rbf

## train a bound constraint support vector machine
irismodel <- ksvm(Species~.,data=iris,type="C-bsvc",
                  kernel=rbf,C=10,prob.model=TRUE)

irismodel

## get fitted values
fitted(irismodel)

## Test on the training set with probabilities as output
predict(irismodel, iris[,-5], type="probabilities")


## Demo of the plot function
x <- rbind(matrix(rnorm(120),,2),matrix(rnorm(120,mean=3),,2))
y <- matrix(c(rep(1,60),rep(-1,60)))

svp <- ksvm(x,y,type="C-svc")
plot(svp,data=x)


### Use kernelMatrix
K <- as.kernelMatrix(crossprod(t(x)))

svp2 <- ksvm(K, y, type="C-svc")

svp2

# test data
xtest <- rbind(matrix(rnorm(20),,2),matrix(rnorm(20,mean=3),,2))
# test kernel matrix i.e. inner/kernel product of test data with
# Support Vectors

Ktest <- as.kernelMatrix(crossprod(t(xtest),t(x[SVindex(svp2), ])))

predict(svp2, Ktest)


#### Use custom kernel

k <- function(x,y) {(sum(x*y) +1)*exp(-0.001*sum((x-y)^2))}
class(k) <- "kernel"

data(promotergene)

## train svm using custom kernel
gene <- ksvm(Class~.,data=promotergene[c(1:20, 80:100),],kernel=k,
             C=5,cross=5)

gene


#### Use text with string kernels
data(reuters)
is(reuters)
tsv <- ksvm(reuters,rlabels,kernel="stringdot",
            kpar=list(length=5),cross=3,C=10)
tsv


## regression
# create data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.03)

# train support vector machine
regm <- ksvm(x,y,epsilon=0.01,kpar=list(sigma=16),cross=3)
plot(x,y,type="l")
lines(x,predict(regm,x),col="red")
```

---

ksvm-class *Class "ksvm"*

Description

An S4 class containing the output (model) of the ksvm Support Vector Machines function

Objects from the Class

Objects can be created by calls of the form new("ksvm", ...) or by calls to the ksvm function.

Slots

type: Object of class "character" containing the support vector machine type ("C-svc", "nu-svc", "C-bsvc", "spoc-svc", "one-svc", "eps-svr", "nu-svr", "eps-bsvr")

param: Object of class "list" containing the Support Vector Machine parameters (C, nu, epsilon)

kernelf: Object of class "function" containing the kernel function

kpar: Object of class "list" containing the kernel function parameters (hyperparameters)

kcall: Object of class "ANY" containing the ksvm function call

scaling: Object of class "ANY" containing the scaling information performed on the data

terms: Object of class "ANY" containing the terms representation of the symbolic model used (when using a formula) `xmatrix`**:** Object of class `"input"` (`"list"` for multiclass problems or `"matrix"` for binary classification and regression problems) containing the support vectors calculated from the data matrix used during computations (possibly scaled and without NA). In the case of multi-class classification each list entry contains the support vectors from each binary classification problem from the one-against-one method.

`ymatrix`**:** Object of class `"output"` the response `"matrix"` or `"factor"` or `"vector"` or `"logical"`

`fitted`**:** Object of class `"output"` with the fitted values, predictions using the training set.

`lev`**:** Object of class `"vector"` with the levels of the response (in the case of classification)

`prob.model`**:** Object of class `"list"` with the class prob. model

`prior`**:** Object of class `"list"` with the prior of the training set

`nclass`**:** Object of class `"numeric"` containing the number of classes (in the case of classification)

`alpha`**:** Object of class `"listI"` containing the resulting alpha vector (`"list"` or `"matrix"` in case of multiclass classification) (support vectors)

`coef`**:** Object of class `"ANY"` containing the resulting coefficients

`alphaindex`**:** Object of class `"list"` containing

`b`**:** Object of class `"numeric"` containing the resulting offset

`SVindex`**:** Object of class `"vector"` containing the indexes of the support vectors

`nSV`**:** Object of class `"numeric"` containing the number of support vectors

`obj`**:** Object of class `vector` containing the value of the objective function. When using one-against-one in multiclass classification this is a vector.

`error`**:** Object of class `"numeric"` containing the training error

`cross`**:** Object of class `"numeric"` containing the cross-validation error

`n.action`**:** Object of class `"ANY"` containing the action performed for NA

Methods

SVindex `signature(object = "ksvm")`: return the indexes of support vectors

alpha `signature(object = "ksvm")`: returns the complete 5 alpha vector (wit zero values)

alphaindex `signature(object = "ksvm")`: returns the indexes of non-zero alphas (support vectors)

cross `signature(object = "ksvm")`: returns the cross-validation error

error `signature(object = "ksvm")`: returns the training error

obj `signature(object = "ksvm")`: returns the value of the objective function

fitted `signature(object = "vm")`: returns the fitted values (predict on training set)

kernelf `signature(object = "ksvm")`: returns the kernel function

kpar `signature(object = "ksvm")`: returns the kernel parameters (hyperparameters)

lev `signature(object = "ksvm")`: returns the levels in case of classification

prob.model `signature(object="ksvm")`: returns class prob. model values

param `signature(object="ksvm")`: returns the parameters of the SVM in a list (C, epsilon, nu etc.)

prior `signature(object="ksvm")`: returns the prior of the training set

kcall `signature(object="ksvm")`: returns the `ksvm` function call

scaling `signature(object = "ksvm")`: returns the scaling values

show `signature(object = "ksvm")`: prints the object information

type `signature(object = "ksvm")`: returns the problem type

xmatrix `signature(object = "ksvm")`: returns the data matrix used

ymatrix `signature(object = "ksvm")`: returns the response vector

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzolgou@ci.tuwien.ac.at>`

See Also

`ksvm`, `rvm-class`, `gausspr-class`

Examples

```
## simple example using the promotergene data set
data(promotergene)

## train a support vector machine
gene <- ksvm(Class~.,data=promotergene,kernel="rbfdot",
             kpar=list(sigma=0.015),C=50,cross=4)
gene

# the kernel  function
kernelf(gene)
# the alpha values
alpha(gene)
# the coefficients
coef(gene)
# the fitted values
fitted(gene)
# the cross validation error
cross(gene)
```

`lssvm` *Least Squares Support Vector Machine*

Description

The `lssvm` function is an implementation of the Least Squares SVM. `lssvm` includes a reduced version of Least Squares SVM using a decomposition of the kernel matrix which is calculated by the `csi` function.

Usage

```
## S4 method for signature 'formula'
lssvm(x, data=NULL, ..., subset, na.action = na.omit, scaled = TRUE)

## S4 method for signature 'vector'
lssvm(x, ...)

## S4 method for signature 'matrix'
lssvm(x, y, scaled = TRUE, kernel = "rbfdot", kpar = "automatic",
      type = NULL, tau = 0.01, reduced = TRUE, tol = 0.0001,
      rank = floor(dim(x)[1]/3), delta = 40, cross = 0, fit = TRUE,
      ..., subset, na.action = na.omit)

## S4 method for signature 'kernelMatrix'
lssvm(x, y, type = NULL, tau = 0.01,
      tol = 0.0001, rank = floor(dim(x)[1]/3), delta = 40, cross = 0,
      fit = TRUE, ...)

## S4 method for signature 'list'
lssvm(x, y, scaled = TRUE,
      kernel = "stringdot", kpar = list(length=4, lambda = 0.5),
      type = NULL, tau = 0.01, reduced = TRUE, tol = 0.0001,
      rank = floor(dim(x)[1]/3), delta = 40, cross = 0, fit = TRUE,
      ..., subset)
```

Arguments

`x`	a symbolic description of the model to be fit, a matrix or vector containing the training data when a formula interface is not used or a `kernelMatrix` or a list of character vectors.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'lssvm' is called from.
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for classification or regression - currently nor supported -).
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally to zero mean and unit variance. The center and scale values are returned and used for later predictions.
`type`	Type of problem. Either "classification" or "regression". Depending on whether `y` is a factor or not, the default setting for `type` is "classification" or "regression" respectively, but can be overwritten by setting an explicit value. (regression is currently not supported)
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector

	arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel "Gaussian" • `polydot` Polynomial kernel • `vanilladot` Linear kernel • `tanhdot` Hyperbolic tangent kernel • `laplacedot` Laplacian kernel • `besseldot` Bessel kernel • `anovadot` ANOVA RBF kernel • `splinedot` Spline kernel • `stringdot` String kernel Setting the kernel parameter to "matrix" treats `x` as a kernel matrix calling the `kernelMatrix` interface. The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. For valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". • `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized. Hyper-parameters for user defined kernels can be passed through the kpar parameter as well. `kpar` can also be set to the string "automatic" which uses the heuristics in `sigest` to calculate a good `sigma` value for the Gaussian RBF or Laplace kernel, from the data. (default = "automatic").
`tau`	the regularization parameter (default 0.01)
`reduced`	if set to `FALSE` the full linear problem of the lssvm is solved, when `TRUE` a reduced method using `csi` is used.
`rank`	the maximal rank of the decomposed kernel matrix, see `csi`
`delta`	number of columns of cholesky performed in advance, see `csi` (default 40)
`tol`	tolerance of termination criterion for the `csi` function, lower tolerance leads to more precise approximation but may increase the training time and the decomposed matrix size (default: 0.0001)

`fit`	indicates whether the fitted values should be computed and included in the model or not (default: 'TRUE')
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the Mean Squared Error for regression
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

Least Squares Support Vector Machines are reformulation to the standard SVMs that lead to solving linear KKT systems. The algorithm is based on the minimization of a classical penalized least-squares cost function. The current implementation approximates the kernel matrix by an incomplete Cholesky factorization obtained by the `csi` function, thus the solution is an approximation to the exact solution of the lssvm optimization problem. The quality of the solution depends on the approximation and can be influenced by the "rank" , "delta", and "tol" parameters.

Value

An S4 object of class `"lssvm"` containing the fitted model, Accessor functions can be used to access the slots of the object (see examples) which include:

`alpha`	the parameters of the `"lssvm"`
`coef`	the model coefficients (identical to alpha)
`b`	the model offset.
`xmatrix`	the training data used by the model

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

J. A. K. Suykens and J. Vandewalle
Least Squares Support Vector Machine Classifiers
Neural Processing Letters vol. 9, issue 3, June 1999

See Also

`ksvm`, `gausspr`, `csi`

Examples

```
## simple example
data(iris)

lir <- lssvm(Species~.,data=iris)

lir

lirr <- lssvm(Species~.,data= iris, reduced = FALSE)

lirr

## Using the kernelMatrix interface

iris <- unique(iris)

rbf <- rbfdot(0.5)

k <- kernelMatrix(rbf, as.matrix(iris[,-5]))

klir <- lssvm(k, iris[, 5])

klir

pre <- predict(klir, k)
```

`lssvm-class` *Class "lssvm"*

Description

The Gaussian Processes object

Objects from the Class

Objects can be created by calls of the form `new("lssvm", ...)`. or by calling the `lssvm` function

Slots

`kernelf`**:** Object of class `"kfunction"` contains the kernel function used

`kpar`**:** Object of class `"list"` contains the kernel parameter used

`param`**:** Object of class `"list"` contains the regularization parameter used.

`kcall`**:** Object of class `"call"` contains the used function call

`type`**:** Object of class `"character"` contains type of problem

`coef`**:** Object of class `"ANY"` contains the model parameter

`terms`**:** Object of class `"ANY"` contains the terms representation of the symbolic model used (when using a formula) `xmatrix`**:** Object of class `"matrix"` containing the data matrix used

`ymatrix`**:** Object of class `"output"` containing the response matrix

`fitted`**:** Object of class `"output"` containing the fitted values

`b`**:** Object of class `"numeric"` containing the offset

`lev`**:** Object of class `"vector"` containing the levels of the response (in case of classification)

`scaling`**:** Object of class `"ANY"` containing the scaling information performed on the data

`nclass`**:** Object of class `"numeric"` containing the number of classes (in case of classification)

`alpha`**:** Object of class `"listI"` containing the computes alpha values

`alphaindex` Object of class `"list"` containing the indexes for the alphas in various classes (in multi-class problems).

`error`**:** Object of class `"numeric"` containing the training error

`cross`**:** Object of class `"numeric"` containing the cross validation error

`n.action`**:** Object of class `"ANY"` containing the action performed in NA

`nSV`**:** Object of class `"numeric"` containing the number of model parameters

Methods

alpha `signature(object = "lssvm")`: returns the alpha vector

cross `signature(object = "lssvm")`: returns the cross validation error

error `signature(object = "lssvm")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values

kcall `signature(object = "lssvm")`: returns the call performed

kernelf `signature(object = "lssvm")`: returns the kernel function used

kpar `signature(object = "lssvm")`: returns the kernel parameter used

param `signature(object = "lssvm")`: returns the regularization parameter used

lev `signature(object = "lssvm")`: returns the response levels (in classification)

type `signature(object = "lssvm")`: returns the type of problem

scaling `signature(object = "ksvm")`: returns the scaling values

xmatrix `signature(object = "lssvm")`: returns the data matrix used

ymatrix `signature(object = "lssvm")`: returns the response matrix used

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`lssvm`, `ksvm-class`

Examples

```
# train model
data(iris)
test <- lssvm(Species~.,data=iris,var=2)
test
alpha(test)
error(test)
lev(test)
```

`musk` *Musk data set*

Description

This dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks.

Usage

```
data(musk)
```

Format

A data frame with 476 observations on the following 167 variables.

Variables 1-162 are "distance features" along rays. The distances are measured in hundredths of Angstroms. The distances may be negative or positive, since they are actually measured relative to an origin placed along each ray. The origin was defined by a "consensus musk" surface that is no longer used. Hence, any experiments with the data should treat these feature values as lying on an arbitrary continuous scale. In particular, the algorithm should not make any use of the zero point or the sign of each feature value.

Variable 163 is the distance of the oxygen atom in the molecule to a designated point in 3-space. This is also called OXY-DIS.

Variable 164 is the X-displacement from the designated point.

Variable 165 is the Y-displacement from the designated point.

Variable 166 is the Z-displacement from the designated point.

Class: 0 for non-musk, and 1 for musk

Source

UCI Machine Learning data repository

Examples

```
data(musk)

muskm <- ksvm(Class~.,data=musk,kernel="rbfdot",C=1000)

muskm
```

`onlearn` *Kernel Online Learning algorithms*

Description

Online Kernel-based Learning algorithms for classification, novelty detection, and regression.

Usage

```
## S4 method for signature 'onlearn'
onlearn(obj, x, y = NULL, nu = 0.2, lambda = 1e-04)
```

Arguments

`obj`	`obj` an object of class `onlearn` created by the initialization function `inlearn` containing the kernel to be used during learning and the parameters of the learned model
`x`	vector or matrix containing the data. Factors have to be numerically coded. If `x` is a matrix the code is run internally one sample at the time.
`y`	the class label in case of classification. Only binary classification is supported and class labels have to be -1 or +1.
`nu`	the parameter similarly to the `nu` parameter in SVM bounds the training error.
`lambda`	the learning rate

Details

The online algorithms are based on a simple stochastic gradient descent method in feature space. The state of the algorithm is stored in an object of class `onlearn` and has to be passed to the function at each iteration.

Value

The function returns an `S4` object of class `onlearn` containing the model parameters and the last fitted value which can be retrieved by the accessor method `fit`. The value returned in the classification and novelty detection problem is the decision function value phi. The accessor methods `alpha` returns the model parameters.

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

References

Kivinen J. Smola A.J. Williamson R.C.
Online Learning with Kernels
IEEE Transactions on Signal Processing vol. 52, Issue 8, 2004
https://alex.smola.org/papers/2004/KivSmoWil04.pdf

See Also

inlearn

Examples

```
## create toy data set
x <- rbind(matrix(rnorm(100),,2),matrix(rnorm(100)+3,,2))
y <- matrix(c(rep(1,50),rep(-1,50)),,1)

## initialize onlearn object
on <- inlearn(2,kernel="rbfdot",kpar=list(sigma=0.2),
              type="classification")

ind <- sample(1:100,100)
## learn one data point at the time
for(i in ind)
on <- onlearn(on,x[i,],y[i],nu=0.03,lambda=0.1)

## or learn all the data
on <- onlearn(on,x[ind,],y[ind],nu=0.03,lambda=0.1)

sign(predict(on,x))
```

onlearn-class *Class "onlearn"*

Description

The class of objects used by the Kernel-based Online learning algorithms

Objects from the Class

Objects can be created by calls of the form new("onlearn", ...). or by calls to the function inlearn.

Slots

`kernelf`**:** Object of class `"function"` containing the used kernel function

`buffer`**:** Object of class `"numeric"` containing the size of the buffer

`kpar`**:** Object of class `"list"` containing the hyperparameters of the kernel function.

`xmatrix`**:** Object of class `"matrix"` containing the data points (similar to support vectors)

`fit`**:** Object of class `"numeric"` containing the decision function value of the last data point

`onstart`**:** Object of class `"numeric"` used for indexing

`onstop`**:** Object of class `"numeric"` used for indexing

`alpha`**:** Object of class `"ANY"` containing the model parameters

`rho`**:** Object of class `"numeric"` containing model parameter

`b`**:** Object of class `"numeric"` containing the offset

`pattern`**:** Object of class `"factor"` used for dealing with factors

`type`**:** Object of class `"character"` containing the problem type (classification, regression, or novelty

Methods

alpha `signature(object = "onlearn")`: returns the model parameters

b `signature(object = "onlearn")`: returns the offset

buffer `signature(object = "onlearn")`: returns the buffer size

fit `signature(object = "onlearn")`: returns the last decision function value

kernelf `signature(object = "onlearn")`: return the kernel function used

kpar `signature(object = "onlearn")`: returns the hyper-parameters used

onlearn `signature(obj = "onlearn")`: the learning function

predict `signature(object = "onlearn")`: the predict function

rho `signature(object = "onlearn")`: returns model parameter

show `signature(object = "onlearn")`: show function

type `signature(object = "onlearn")`: returns the type of problem

xmatrix `signature(object = "onlearn")`: returns the stored data points

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`onlearn`, `inlearn`

Examples

```
## create toy data set
x <- rbind(matrix(rnorm(100),,2),matrix(rnorm(100)+3,,2))
y <- matrix(c(rep(1,50),rep(-1,50)),,1)

## initialize onlearn object
on <- inlearn(2,kernel="rbfdot",kpar=list(sigma=0.2),
              type="classification")

## learn one data point at the time
for(i in sample(1:100,100))
on <- onlearn(on,x[i,],y[i],nu=0.03,lambda=0.1)

sign(predict(on,x))
```

---

plot *plot method for support vector object*

Description

Plot a binary classification support vector machine object. The `plot` function returns a contour plot of the decision values.

Usage

```
## S4 method for signature 'ksvm'
plot(object, data=NULL, grid = 50, slice = list())
```

Arguments

`object`	a `ksvm` classification object created by the `ksvm` function
`data`	a data frame or matrix containing data to be plotted
`grid`	granularity for the contour plot.
`slice`	a list of named numeric values for the dimensions held constant (only needed if more than two variables are used). Dimensions not specified are fixed at 0.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`ksvm`

Examples

```
## Demo of the plot function
x <- rbind(matrix(rnorm(120),,2),matrix(rnorm(120,mean=3),,2))
y <- matrix(c(rep(1,60),rep(-1,60)))

svp <- ksvm(x,y,type="C-svc")
plot(svp,data=x)
```

---

prc-class *Class "prc"*

Description

Principal Components Class

Objects of class "prc"

Objects from the class cannot be created directly but only contained in other classes.

Slots

`pcv`: Object of class `"matrix"` containing the principal component vectors

`eig`: Object of class `"vector"` containing the corresponding eigenvalues

`kernelf`: Object of class `"kfunction"` containing the kernel function used

`kpar`: Object of class `"list"` containing the kernel parameters used

`xmatrix`: Object of class `"input"` containing the data matrix used

`kcall`: Object of class `"ANY"` containing the function call

`n.action`: Object of class `"ANY"` containing the action performed on NA

Methods

eig `signature(object = "prc")`: returns the eigenvalues

kcall `signature(object = "prc")`: returns the performed call

kernelf `signature(object = "prc")`: returns the used kernel function

pcv `signature(object = "prc")`: returns the principal component vectors

predict `signature(object = "prc")`: embeds new data

xmatrix `signature(object = "prc")`: returns the used data matrix

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`kpca-class`,`kha-class`, `kfa-class`

predict.gausspr — *predict method for Gaussian Processes object*

Description

Prediction of test data using Gaussian Processes

Usage

```
## S4 method for signature 'gausspr'
predict(object, newdata, type = "response", coupler = "minpair")
```

Arguments

object	an S4 object of class gausspr created by the gausspr function
newdata	a data frame or matrix containing new data
type	one of response, probabilities indicating the type of output: predicted values or matrix of class probabilities
coupler	Coupling method used in the multiclass case, can be one of minpair or pkpd (see reference for more details).

Value

response	predicted classes (the classes with majority vote) or the response value in regression.
probabilities	matrix of class probabilities (one column for each class and one row for each input).

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

References

- C. K. I. Williams and D. Barber
  Bayesian classification with Gaussian processes.
  IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(12):1342-1351, 1998
  https://homepages.inf.ed.ac.uk/ckiw/postscript/pami_final.ps.gz
- T.F. Wu, C.J. Lin, R.C. Weng.
  Probability estimates for Multi-class Classification by Pairwise Coupling
  https://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf

Examples

```
## example using the promotergene data set
data(promotergene)

## create test and training set
ind <- sample(1:dim(promotergene)[1],20)
genetrain <- promotergene[-ind, ]
genetest <- promotergene[ind, ]

## train a support vector machine
gene <- gausspr(Class~.,data=genetrain,kernel="rbfdot",
                kpar=list(sigma=0.015))
gene

## predict gene type probabilities on the test set
genetype <- predict(gene,genetest,type="probabilities")
genetype
```

`predict.kqr` *Predict method for kernel Quantile Regression object*

Description

Prediction of test data for kernel quantile regression

Usage

```
## S4 method for signature 'kqr'
predict(object, newdata)
```

Arguments

`object`	an S4 object of class `kqr` created by the `kqr` function
`newdata`	a data frame, matrix, or kernelMatrix containing new data

Value

The value of the quantile given by the computed `kqr` model in a vector of length equal to the the rows of `newdata`.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

Examples

```
# create data
x <- sort(runif(300))
y <- sin(pi*x) + rnorm(300,0,sd=exp(sin(2*pi*x)))

# first calculate the median
qrm <- kqr(x, y, tau = 0.5, C=0.15)

# predict and plot
plot(x, y)
ytest <- predict(qrm, x)
lines(x, ytest, col="blue")

# calculate 0.9 quantile
qrm <- kqr(x, y, tau = 0.9, kernel = "rbfdot",
           kpar= list(sigma=10), C=0.15)
ytest <- predict(qrm, x)
lines(x, ytest, col="red")
```

`predict.ksvm` *predict method for support vector object*

Description

Prediction of test data using support vector machines

Usage

```
## S4 method for signature 'ksvm'
predict(object, newdata, type = "response", coupler = "minpair")
```

Arguments

`object`	an S4 object of class `ksvm` created by the `ksvm` function
`newdata`	a data frame or matrix containing new data
`type`	one of `response`, `probabilities` ,`votes`, `decision` indicating the type of output: predicted values, matrix of class probabilities, matrix of vote counts, or matrix of decision values.
`coupler`	Coupling method used in the multiclass case, can be one of `minpair` or `pkpd` (see reference for more details).

Value

If `type(object)` is `C-svc`, `nu-svc`, `C-bsvm` or `spoc-svc` the vector returned depends on the argument `type`:

`response`	predicted classes (the classes with majority vote).

`probabilities`	matrix of class probabilities (one column for each class and one row for each input).
`votes`	matrix of vote counts (one column for each class and one row for each new input)

If `type(object)` is `eps-svr`, `eps-bsvr` or `nu-svr` a vector of predicted values is returned. If `type(object)` is `one-classification` a vector of logical values is returned.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

- T.F. Wu, C.J. Lin, R.C. Weng.
  Probability estimates for Multi-class Classification by Pairwise Coupling
  `https://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf`
- H.T. Lin, C.J. Lin, R.C. Weng
  A note on Platt's probabilistic outputs for support vector machines
  `https://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.pdf`

Examples

```
## example using the promotergene data set
data(promotergene)

## create test and training set
ind <- sample(1:dim(promotergene)[1],20)
genetrain <- promotergene[-ind, ]
genetest <- promotergene[ind, ]

## train a support vector machine
gene <- ksvm(Class~.,data=genetrain,kernel="rbfdot",
             kpar=list(sigma=0.015),C=70,cross=4,prob.model=TRUE)
gene

## predict gene type probabilities on the test set
genetype <- predict(gene,genetest,type="probabilities")
genetype
```

`promotergene` *E. coli promoter gene sequences (DNA)*

Description

Promoters have a region where a protein (RNA polymerase) must make contact and the helical DNA sequence must have a valid conformation so that the two pieces of the contact region spatially align. The data contains DNA sequences of promoters and non-promoters.

Usage

```
data(promotergene)
```

Format

A data frame with 106 observations and 58 variables. The first variable `Class` is a factor with levels `+` for a promoter gene and `-` for a non-promoter gene. The remaining 57 variables `V2` to `V58` are factors describing the sequence. The DNA bases are coded as follows: `a` adenine `c` cytosine `g` guanine `t` thymine

Source

UCI Machine Learning data repository
https://archive.ics.uci.edu/ml/machine-learning-databases/molecular-biology/promoter-gene-sequences

References

Towell, G., Shavlik, J. and Noordewier, M.
Refinement of Approximate Domain Theories by Knowledge-Based Artificial Neural Networks.
In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)

Examples

```
data(promotergene)

## Create classification model using Gaussian Processes

prom <- gausspr(Class~.,data=promotergene,kernel="rbfdot",
                kpar=list(sigma=0.02),cross=4)
prom

## Create model using Support Vector Machines

promsv <- ksvm(Class~.,data=promotergene,kernel="laplacedot",
               kpar="automatic",C=60,cross=4)
promsv
```

`ranking` *Ranking*

Description

A universal ranking algorithm which assigns importance/ranking to data points given a query.

Usage

```
## S4 method for signature 'matrix'
ranking(x, y,
        kernel ="rbfdot", kpar = list(sigma = 1),
        scale = FALSE, alpha = 0.99, iterations = 600,
        edgegraph = FALSE, convergence = FALSE ,...)

## S4 method for signature 'kernelMatrix'
ranking(x, y,
        alpha = 0.99, iterations = 600, convergence = FALSE,...)

## S4 method for signature 'list'
ranking(x, y,
        kernel = "stringdot", kpar = list(length = 4, lambda = 0.5),
        alpha = 0.99, iterations = 600, convergence = FALSE, ...)
```

Arguments

`x`	a matrix containing the data to be ranked, or the kernel matrix of data to be ranked or a list of character vectors
`y`	The index of the query point in the data matrix or a vector of length equal to the rows of the data matrix having a one at the index of the query points index and zero at all the other points.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. For valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot"

- `sigma, order, degree` for the Bessel kernel "besseldot".
- `sigma, degree` for the ANOVA kernel "anovadot".

Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.

`scale`	If TRUE the data matrix columns are scaled to zero mean and unit variance.
`alpha`	The `alpha` parameter takes values between 0 and 1 and is used to control the authoritative scores received from the unlabeled points. For 0 no global structure is found the algorithm ranks the points similarly to the original distance metric.
`iterations`	Maximum number of iterations
`edgegraph`	Construct edgegraph (only supported with the RBF kernel)
`convergence`	Include convergence matrix in results
`...`	Additional arguments

Details

A simple universal ranking algorithm which exploits the intrinsic global geometric structure of the data. In many real world applications this should be superior to a local method in which the data are simply ranked by pairwise Euclidean distances. Firstly a weighted network is defined on the data and an authoritative score is assigned to each query. The query points act as source nodes that continually pump their authoritative scores to the remaining points via the weighted network and the remaining points further spread the scores they received to their neighbors. This spreading process is repeated until convergence and the points are ranked according to their score at the end of the iterations.

Value

An S4 object of class `ranking` which extends the `matrix` class. The first column of the returned matrix contains the original index of the points in the data matrix the second column contains the final score received by each point and the third column the ranking of the point. The object contains the following slots :

`edgegraph`	Containing the edgegraph of the data points.
`convergence`	Containing the convergence matrix

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

D. Zhou, J. Weston, A. Gretton, O. Bousquet, B. Schoelkopf
Ranking on Data Manifolds
Advances in Neural Information Processing Systems 16.
MIT Press Cambridge Mass. 2004
`http://papers.neurips.cc/paper/2447-ranking-on-data-manifolds.pdf`

See Also

ranking-class, specc

Examples

```
data(spirals)

## create data from spirals
ran <- spirals[rowSums(abs(spirals) < 0.55) == 2,]

## rank points according to similarity to the most upper left point
ranked <- ranking(ran, 54, kernel = "rbfdot",
                  kpar = list(sigma = 100), edgegraph = TRUE)
ranked[54, 2] <- max(ranked[-54, 2])
c<-1:86
op <- par(mfrow = c(1, 2),pty="s")
plot(ran)
plot(ran, cex=c[ranked[,3]]/40)
```

`ranking-class` *Class "ranking"*

Description

Object of the class `"ranking"` are created from the `ranking` function and extend the class `matrix`

Objects from the Class

Objects can be created by calls of the form `new("ranking", ...)`.

Slots

`.Data`**:** Object of class `"matrix"` containing the data ranking and scores

`convergence`**:** Object of class `"matrix"` containing the convergence matrix

`edgegraph`**:** Object of class `"matrix"` containing the edgegraph

Extends

Class `"matrix"`, directly.

Methods

show `signature(object = "ranking")`: displays the ranking score matrix

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

ranking

Examples

```
data(spirals)

## create data set to be ranked
ran<-spirals[rowSums(abs(spirals)<0.55)==2,]

## rank points according to "relevance" to point 54 (up left)
ranked<-ranking(ran,54,kernel="rbfdot",
                kpar=list(sigma=100),edgegraph=TRUE)

ranked
edgegraph(ranked)[1:10,1:10]
```

---

reuters *Reuters Text Data*

Description

A small sample from the Reuters news data set.

Usage

```
data(reuters)
```

Format

A list of 40 text documents along with the labels. `reuters` contains the text documents and `rlabels` the labels in a vector.

Details

This dataset contains a list of 40 text documents along with the labels. The data consist out of 20 documents from the `acq` category and 20 documents from the crude category. The labels are stored in `rlabels`

Source

Reuters

rvm *Relevance Vector Machine*

Description

The Relevance Vector Machine is a Bayesian model for regression and classification of identical functional form to the support vector machine. The `rvm` function currently supports only regression.

Usage

```
## S4 method for signature 'formula'
rvm(x, data=NULL, ..., subset, na.action = na.omit)

## S4 method for signature 'vector'
rvm(x, ...)

## S4 method for signature 'matrix'
rvm(x, y, type="regression",
    kernel="rbfdot", kpar="automatic",
    alpha= ncol(as.matrix(x)), var=0.1, var.fix=FALSE, iterations=100,
    verbosity = 0, tol = .Machine$double.eps, minmaxdiff = 1e-3,
    cross = 0, fit = TRUE, ... , subset, na.action = na.omit)

## S4 method for signature 'list'
rvm(x, y, type = "regression",
    kernel = "stringdot", kpar = list(length = 4, lambda = 0.5),
    alpha = 5, var = 0.1, var.fix = FALSE, iterations = 100,
    verbosity = 0, tol = .Machine$double.eps, minmaxdiff = 1e-3,
    cross = 0, fit = TRUE, ..., subset, na.action = na.omit)
```

Arguments

`x`	a symbolic description of the model to be fit. When not using a formula x can be a matrix or vector containing the training data or a kernel matrix of class `kernelMatrix` of the training data or a list of character vectors (for use with the string kernel). Note, that the intercept is always excluded, whether given in the formula or not.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'rvm' is called from.
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).
`type`	`rvm` can only be used for regression at the moment.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

- `rbfdot` Radial Basis kernel "Gaussian"
- `polydot` Polynomial kernel
- `vanilladot` Linear kernel
- `tanhdot` Hyperbolic tangent kernel
- `laplacedot` Laplacian kernel
- `besseldot` Bessel kernel
- `anovadot` ANOVA RBF kernel
- `splinedot` Spline kernel
- `stringdot` String kernel

The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. For valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". • `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized. Hyper-parameters for user defined kernels can be passed through the kpar parameter as well. In the case of a Radial Basis kernel function (Gaussian) kpar can also be set to the string "automatic" which uses the heuristics in `sigest` to calculate a good `sigma` value for the Gaussian RBF or Laplace kernel, from the data. (default = "automatic").
`alpha`	The initial alpha vector. Can be either a vector of length equal to the number of data points or a single number.
`var`	the initial noise variance
`var.fix`	Keep noise variance fix during iterations (default: FALSE)
`iterations`	Number of iterations allowed (default: 100)
`tol`	tolerance of termination criterion
`minmaxdiff`	termination criteria. Stop when max difference is equal to this parameter (default:1e-3)
`verbosity`	print information on algorithm convergence (default = FALSE)
`fit`	indicates whether the fitted values should be computed and included in the model or not (default: TRUE)
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the Mean Squared Error for regression

`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

The Relevance Vector Machine typically leads to sparser models then the SVM. It also performs better in many cases (specially in regression).

Value

An S4 object of class "rvm" containing the fitted model. Accessor functions can be used to access the slots of the object which include :

`alpha`	The resulting relevance vectors
`alphaindex`	The index of the resulting relevance vectors in the data matrix
`nRV`	Number of relevance vectors
`RVindex`	The indexes of the relevance vectors
`error`	Training error (if `fit` = `TRUE`)

...

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Tipping, M. E.
Sparse Bayesian learning and the relevance vector machine
Journal of Machine Learning Research 1, 211-244
`https://www.jmlr.org/papers/volume1/tipping01a/tipping01a.pdf`

See Also

`ksvm`

Examples

```
# create data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.05)

# train relevance vector machine
```

```
foo <- rvm(x, y)
foo
# print relevance vectors
alpha(foo)
RVindex(foo)

# predict and plot
ytest <- predict(foo, x)
plot(x, y, type ="l")
lines(x, ytest, col="red")
```

---

rvm-class *Class "rvm"*

Description

Relevance Vector Machine Class

Objects from the Class

Objects can be created by calls of the form `new("rvm", ...)`. or by calling the `rvm` function.

Slots

`tol`: Object of class `"numeric"` contains tolerance of termination criteria used.

`kernelf`: Object of class `"kfunction"` contains the kernel function used

`kpar`: Object of class `"list"` contains the hyperparameter used

`kcall`: Object of class `"call"` contains the function call

`type`: Object of class `"character"` contains type of problem

`terms`: Object of class `"ANY"` containing the terms representation of the symbolic model used (when using a formula interface)

`xmatrix`: Object of class `"matrix"` contains the data matrix used during computation

`ymatrix`: Object of class `"output"` contains the response matrix

`fitted`: Object of class `"output"` with the fitted values, (predict on training set).

`lev`: Object of class `"vector"` contains the levels of the response (in classification)

`nclass`: Object of class `"numeric"` contains the number of classes (in classification)

`alpha`: Object of class `"listI"` containing the the resulting alpha vector

`coef`: Object of class `"ANY"` containing the the resulting model parameters

`nvar`: Object of class `"numeric"` containing the calculated variance (in case of regression)

`mlike`: Object of class `"numeric"` containing the computed maximum likelihood

`RVindex`: Object of class `"vector"` containing the indexes of the resulting relevance vectors

`nRV`: Object of class `"numeric"` containing the number of relevance vectors

`cross`: Object of class `"numeric"` containing the resulting cross validation error

`error`: Object of class `"numeric"` containing the training error

`n.action`: Object of class `"ANY"` containing the action performed on NA

Methods

RVindex `signature(object = "rvm")`: returns the index of the relevance vectors

alpha `signature(object = "rvm")`: returns the resulting alpha vector

cross `signature(object = "rvm")`: returns the resulting cross validation error

error `signature(object = "rvm")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values

kcall `signature(object = "rvm")`: returns the function call

kernelf `signature(object = "rvm")`: returns the used kernel function

kpar `signature(object = "rvm")`: returns the parameters of the kernel function

lev `signature(object = "rvm")`: returns the levels of the response (in classification)

mlike `signature(object = "rvm")`: returns the estimated maximum likelihood

nvar `signature(object = "rvm")`: returns the calculated variance (in regression)

type `signature(object = "rvm")`: returns the type of problem

xmatrix `signature(object = "rvm")`: returns the data matrix used during computation

ymatrix `signature(object = "rvm")`: returns the used response

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`rvm`, `ksvm-class`

Examples

```
# create data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.05)

# train relevance vector machine
foo <- rvm(x, y)
foo

alpha(foo)
RVindex(foo)
fitted(foo)
kernelf(foo)
nvar(foo)

## show slots
slotNames(foo)
```

sigest — *Hyperparameter estimation for the Gaussian Radial Basis kernel*

Description

Given a range of values for the "sigma" inverse width parameter in the Gaussian Radial Basis kernel for use with Support Vector Machines. The estimation is based on the data to be used.

Usage

```
## S4 method for signature 'formula'
sigest(x, data=NULL, frac = 0.5, na.action = na.omit, scaled = TRUE)
## S4 method for signature 'matrix'
sigest(x, frac = 0.5, scaled = TRUE, na.action = na.omit)
```

Arguments

`x`	a symbolic description of the model upon the estimation is based. When not using a formula x is a matrix or vector containing the data
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'ksvm' is called from.
`frac`	Fraction of data to use for estimation. By default a quarter of the data is used to estimate the range of the sigma hyperparameter.
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally to zero mean and unit variance (since this the default action in `ksvm` as well). The center and scale values are returned and used for later predictions.
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)

Details

`sigest` estimates the range of values for the sigma parameter which would return good results when used with a Support Vector Machine (`ksvm`). The estimation is based upon the 0.1 and 0.9 quantile of $\|x - x'\|^2$. Basically any value in between those two bounds will produce good results.

Value

Returns a vector of length 3 defining the range (0.1 quantile, median and 0.9 quantile) of the sigma hyperparameter.

Author(s)

Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

References

B. Caputo, K. Sim, F. Furesjo, A. Smola,
Appearance-based object recognition using SVMs: which kernel should I use?
Proc of NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision, Whistler, 2002.

See Also

ksvm

Examples

```
## estimate good sigma values for promotergene
data(promotergene)
srange <- sigest(Class~.,data = promotergene)
srange

s <- srange[2]
s
## create test and training set
ind <- sample(1:dim(promotergene)[1],20)
genetrain <- promotergene[-ind, ]
genetest <- promotergene[ind, ]

## train a support vector machine
gene <- ksvm(Class~.,data=genetrain,kernel="rbfdot",
             kpar=list(sigma = s),C=50,cross=3)
gene

## predict gene type on the test set
promoter <- predict(gene,genetest[,-1])

## Check results
table(promoter,genetest[,1])
```

---

spam *Spam E-mail Database*

Description

A data set collected at Hewlett-Packard Labs, that classifies 4601 e-mails as spam or non-spam. In addition to this class label there are 57 variables indicating the frequency of certain words and characters in the e-mail.

Usage

```
data(spam)
```

Format

A data frame with 4601 observations and 58 variables.

The first 48 variables contain the frequency of the variable name (e.g., business) in the e-mail. If the variable name starts with num (e.g., num650) the it indicates the frequency of the corresponding number (e.g., 650). The variables 49-54 indicate the frequency of the characters ';', '(', '[', '!', '$', and '#'. The variables 55-57 contain the average, longest and total run-length of capital letters. Variable 58 indicates the type of the mail and is either `"nonspam"` or `"spam"`, i.e. unsolicited commercial e-mail.

Details

The data set contains 2788 e-mails classified as `"nonspam"` and 1813 classified as `"spam"`.

The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, pornography... This collection of spam e-mails came from the collectors' postmaster and individuals who had filed spam. The collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators of non-spam. These are useful when constructing a personalized spam filter. One would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter.

Source

- Creators: Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt at Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304
- Donor: George Forman (gforman at nospam hpl.hp.com) 650-857-7835

These data have been taken from the UCI Repository Of Machine Learning Databases at `http://www.ics.uci.edu/~mlearn/MLRepository.html`

References

T. Hastie, R. Tibshirani, J.H. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

specc *Spectral Clustering*

Description

A spectral clustering algorithm. Clustering is performed by embedding the data into the subspace of the eigenvectors of an affinity matrix.

Usage

```
## S4 method for signature 'formula'
specc(x, data = NULL, na.action = na.omit, ...)

## S4 method for signature 'matrix'
specc(x, centers,
      kernel = "rbfdot", kpar = "automatic",
      nystrom.red = FALSE, nystrom.sample = dim(x)[1]/6,
      iterations = 200, mod.sample = 0.75, na.action = na.omit, ...)

## S4 method for signature 'kernelMatrix'
specc(x, centers, nystrom.red = FALSE, iterations = 200, ...)

## S4 method for signature 'list'
specc(x, centers,
      kernel = "stringdot", kpar = list(length=4, lambda=0.5),
      nystrom.red = FALSE, nystrom.sample = length(x)/6,
      iterations = 200, mod.sample = 0.75, na.action = na.omit, ...)
```

Arguments

`x`	the matrix of data to be clustered, or a symbolic description of the model to be fit, or a kernel Matrix of class `kernelMatrix`, or a list of character vectors.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'specc' is called from.
`centers`	Either the number of clusters or a set of initial cluster centers. If the first, a random set of rows in the eigenvectors matrix are chosen as the initial centers.
`kernel`	the kernel function used in computing the affinity matrix. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel • `stringdot` String kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	a character string or the list of hyper-parameters (kernel parameters). The default character string `"automatic"` uses a heuristic to determine a suitable value for the width parameter of the RBF kernel. The second option `"local"` (local

	scaling) uses a more advanced heuristic and sets a width parameter for every point in the data set. This is particularly useful when the data incorporates multiple scales. A list can also be used containing the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". • `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized. Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`nystrom.red`	use nystrom method to calculate eigenvectors. When `TRUE` a sample of the dataset is used to calculate the eigenvalues, thus only a nxm matrix where n the sample size is stored in memory (default: `FALSE`
`nystrom.sample`	number of data points to use for estimating the eigenvalues when using the nystrom method. (default : dim(x)[1]/6)
`mod.sample`	proportion of data to use when estimating sigma (default: 0.75)
`iterations`	the maximum number of iterations allowed.
`na.action`	the action to perform on NA
`...`	additional parameters

Details

Spectral clustering works by embedding the data points of the partitioning problem into the subspace of the k largest eigenvectors of a normalized affinity/kernel matrix. Using a simple clustering method like `kmeans` on the embedded points usually leads to good performance. It can be shown that spectral clustering methods boil down to graph partitioning.
The data can be passed to the `specc` function in a `matrix` or a `data.frame`, in addition `specc` also supports input in the form of a kernel matrix of class `kernelMatrix` or as a list of character vectors where a string kernel has to be used.

Value

An S4 object of class `specc` which extends the class `vector` containing integers indicating the cluster to which each point is allocated. The following slots contain useful information

`centers`	A matrix of cluster centers.
`size`	The number of point in each cluster
`withinss`	The within-cluster sum of squares for each cluster
`kernelf`	The kernel function used

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

References

Andrew Y. Ng, Michael I. Jordan, Yair Weiss
On Spectral Clustering: Analysis and an Algorithm
Neural Information Processing Symposium 2001
http://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm.pdf

See Also

`kkmeans`, `kpca`, `kcca`

Examples

```
## Cluster the spirals data set.
data(spirals)

sc <- specc(spirals, centers=2)

sc
centers(sc)
size(sc)
withinss(sc)

plot(spirals, col=sc)
```

`specc-class` *Class "specc"*

Description

The Spectral Clustering Class

Objects from the Class

Objects can be created by calls of the form `new("specc", ...)`. or by calling the function `specc`.

Slots

`.Data`**:** Object of class `"vector"` containing the cluster assignments

`centers`**:** Object of class `"matrix"` containing the cluster centers

`size`**:** Object of class `"vector"` containing the number of points in each cluster

`withinss`**:** Object of class `"vector"` containing the within-cluster sum of squares for each cluster

`kernelf` Object of class `kernel` containing the used kernel function.

Methods

centers `signature(object = "specc")`: returns the cluster centers

withinss `signature(object = "specc")`: returns the within-cluster sum of squares for each cluster

size `signature(object = "specc")`: returns the number of points in each cluster

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`specc`, `kpca-class`

Examples

```
## Cluster the spirals data set.
data(spirals)

sc <- specc(spirals, centers=2)

centers(sc)
size(sc)
```

`spirals` *Spirals Dataset*

Description

A toy data set representing two spirals with Gaussian noise. The data was created with the `mlbench.spirals` function in `mlbench`.

Usage

```
data(spirals)
```

Format

A matrix with 300 observations and 2 variables.

Examples

```
data(spirals)
plot(spirals)
```

`stringdot` *String Kernel Functions*

Description

String kernels.

Usage

```
stringdot(length = 4, lambda = 1.1, type = "spectrum", normalized = TRUE)
```

Arguments

`length`	The length of the substrings considered
`lambda`	The decay factor
`type`	Type of string kernel, currently the following kernels are supported : `spectrum` the kernel considers only matching substring of exactly length n (also know as string kernel). Each such matching substring is given a constant weight. The length parameter in this kernel has to be $length > 1$. `boundrange` this kernel (also known as boundrange) considers only matching substrings of length less than or equal to a given number N. This type of string kernel requires a length parameter $length > 1$ `constant` The kernel considers all matching substrings and assigns constant weight (e.g. 1) to each of them. This `constant` kernel does not require any additional parameter. `exponential` Exponential Decay kernel where the substring weight decays as the matching substring gets longer. The kernel requires a decay factor $\lambda > 1$ `string` essentially identical to the spectrum kernel, only computed using a more conventional way. `fullstring` essentially identical to the boundrange kernel only computed in a more conventional way.
`normalized`	normalize string kernel values, (default: `TRUE`)

Details

The kernel generating functions are used to initialize a kernel function which calculates the dot (inner) product between two feature vectors in a Hilbert Space. These functions or their function generating names can be passed as a `kernel` argument on almost all functions in **kernlab**(e.g., `ksvm`, `kpca` etc.).

The string kernels calculate similarities between two strings (e.g. texts or sequences) by matching the common substring in the strings. Different types of string kernel exists and are mainly distinguished by how the matching is performed i.e. some string kernels count the exact matchings of n characters (spectrum kernel) between the strings, others allow gaps (mismatch kernel) etc.

Value

Returns an S4 object of class `stringkernel` which extents the `function` class. The resulting function implements the given kernel calculating the inner (dot) product between two character vectors.

`kpar`	a list containing the kernel parameters (hyperparameters) used.

The kernel parameters can be accessed by the `kpar` function.

Note

The `spectrum` and `boundrange` kernel are faster and more efficient implementations of the `string` and `fullstring` kernels which will be still included in `kernlab` for the next two versions.

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzoglou@ci.tuwien.ac.at>`

See Also

`dots` , `kernelMatrix` , `kernelMult`, `kernelPol`

Examples

```
sk <- stringdot(type="string", length=5)

sk
```

`ticdata` *The Insurance Company Data*

Description

This data set used in the CoIL 2000 Challenge contains information on customers of an insurance company. The data consists of 86 variables and includes product usage data and socio-demographic data derived from zip area codes. The data was collected to answer the following question: Can you predict who would be interested in buying a caravan insurance policy and give an explanation why ?

Usage

```
data(ticdata)
```

Format

ticdata: Dataset to train and validate prediction models and build a description (9822 customer records). Each record consists of 86 attributes, containing sociodemographic data (attribute 1-43) and product ownership (attributes 44-86). The sociodemographic data is derived from zip codes. All customers living in areas with the same zip code have the same sociodemographic attributes. Attribute 86, `CARAVAN:Number of mobile home policies`, is the target variable.

Data Format

1	`STYPE`	Customer Subtype
2	`MAANTHUI`	Number of houses 1 - 10
3	`MGEMOMV`	Avg size household 1 - 6
4	`MGEMLEEF`	Average age
5	`MOSHOOFD`	Customer main type
6	`MGODRK`	Roman catholic
7	`MGODPR`	Protestant ...
8	`MGODOV`	Other religion
9	`MGODGE`	No religion
10	`MRELGE`	Married
11	`MRELSA`	Living together
12	`MRELOV`	Other relation
13	`MFALLEEN`	Singles
14	`MFGEKIND`	Household without children
15	`MFWEKIND`	Household with children
16	`MOPLHOOG`	High level education
17	`MOPLMIDD`	Medium level education
18	`MOPLLAAG`	Lower level education
19	`MBERHOOG`	High status
20	`MBERZELF`	Entrepreneur
21	`MBERBOER`	Farmer
22	`MBERMIDD`	Middle management
23	`MBERARBG`	Skilled labourers
24	`MBERARBO`	Unskilled labourers
25	`MSKA`	Social class A
26	`MSKB1`	Social class B1
27	`MSKB2`	Social class B2
28	`MSKC`	Social class C
29	`MSKD`	Social class D
30	`MHHUUR`	Rented house
31	`MHKOOP`	Home owners
32	`MAUT1`	1 car
33	`MAUT2`	2 cars
34	`MAUT0`	No car
35	`MZFONDS`	National Health Service
36	`MZPART`	Private health insurance

37	MINKM30	Income >30.000
38	MINK3045	Income 30-45.000
39	MINK4575	Income 45-75.000
40	MINK7512	Income 75-122.000
41	MINK123M	Income <123.000
42	MINKGEM	Average income
43	MKOOPKLA	Purchasing power class
44	PWAPART	Contribution private third party insurance
45	PWABEDR	Contribution third party insurance (firms)
46	PWALAND	Contribution third party insurance (agriculture)
47	PPERSAUT	Contribution car policies
48	PBESAUT	Contribution delivery van policies
49	PMOTSCO	Contribution motorcycle/scooter policies
50	PVRAAUT	Contribution lorry policies
51	PAANHANG	Contribution trailer policies
52	PTRACTOR	Contribution tractor policies
53	PWERKT	Contribution agricultural machines policies
54	PBROM	Contribution moped policies
55	PLEVEN	Contribution life insurances
56	PPERSONG	Contribution private accident insurance policies
57	PGEZONG	Contribution family accidents insurance policies
58	PWAOREG	Contribution disability insurance policies
59	PBRAND	Contribution fire policies
60	PZEILPL	Contribution surfboard policies
61	PPLEZIER	Contribution boat policies
62	PFIETS	Contribution bicycle policies
63	PINBOED	Contribution property insurance policies
64	PBYSTAND	Contribution social security insurance policies
65	AWAPART	Number of private third party insurance 1 - 12
66	AWABEDR	Number of third party insurance (firms) ...
67	AWALAND	Number of third party insurance (agriculture)
68	APERSAUT	Number of car policies
69	ABESAUT	Number of delivery van policies
70	AMOTSCO	Number of motorcycle/scooter policies
71	AVRAAUT	Number of lorry policies
72	AAANHANG	Number of trailer policies
73	ATRACTOR	Number of tractor policies
74	AWERKT	Number of agricultural machines policies
75	ABROM	Number of moped policies
76	ALEVEN	Number of life insurances
77	APERSONG	Number of private accident insurance policies
78	AGEZONG	Number of family accidents insurance policies
79	AWAOREG	Number of disability insurance policies
80	ABRAND	Number of fire policies
81	AZEILPL	Number of surfboard policies
82	APLEZIER	Number of boat policies
83	AFIETS	Number of bicycle policies
84	AINBOED	Number of property insurance policies

85	ABYSTAND	Number of social security insurance policies
86	CARAVAN	Number of mobile home policies 0 - 1

Note: All the variables starting with M are zipcode variables. They give information on the distribution of that variable, e.g., Rented house, in the zipcode area of the customer.

Details

Information about the insurance company customers consists of 86 variables and includes product usage data and socio-demographic data derived from zip area codes. The data was supplied by the Dutch data mining company Sentient Machine Research and is based on a real world business problem. The training set contains over 5000 descriptions of customers, including the information of whether or not they have a caravan insurance policy. The test set contains 4000 customers. The test and data set are merged in the ticdata set. More information about the data set and the CoIL 2000 Challenge along with publications based on the data set can be found at `http://www.liacs.nl/~putten/library/cc2000/`.

Source

- UCI KDD Archive:`http://kdd.ics.uci.edu`
- Donor: Sentient Machine Research
  Peter van der Putten
  Sentient Machine Research
  Baarsjesweg 224
  1058 AA Amsterdam
  The Netherlands
  +31 20 6186927
  pvdputten@hotmail.com, putten@liacs.nl

References

Peter van der Putten, Michel de Ruiter, Maarten van Someren *CoIL Challenge 2000 Tasks and Results: Predicting and Explaining Caravan Policy Ownership*
`http://www.liacs.nl/~putten/library/cc2000/`

`vm-class` *Class "vm"*

Description

An S4 VIRTUAL class used as a base for the various vector machine classes in **kernlab**

Objects from the Class

Objects from the class cannot be created directly but only contained in other classes.

Slots

`alpha`: Object of class `"listI"` containing the resulting alpha vector (list in case of multiclass classification) (support vectors)

`type`: Object of class `"character"` containing the vector machine type e.g., ("C-svc", "nu-svc", "C-bsvc", "spoc-svc", "one-svc", "eps-svr", "nu-svr", "eps-bsvr")

`kernelf`: Object of class `"function"` containing the kernel function

`kpar`: Object of class `"list"` containing the kernel function parameters (hyperparameters)

`kcall`: Object of class `"call"` containing the function call

`terms`: Object of class `"ANY"` containing the terms representation of the symbolic model used (when using a formula)

`xmatrix`: Object of class `"input"` the data matrix used during computations (support vectors) (possibly scaled and without NA)

`ymatrix`: Object of class `"output"` the response matrix/vector

`fitted`: Object of class `"output"` with the fitted values, predictions using the training set.

`lev`: Object of class `"vector"` with the levels of the response (in the case of classification)

`nclass`: Object of class `"numeric"` containing the number of classes (in the case of classification)

`error`: Object of class `"vector"` containing the training error

`cross`: Object of class `"vector"` containing the cross-validation error

`n.action`: Object of class `"ANY"` containing the action performed for NA

Methods

alpha `signature(object = "vm")`: returns the complete alpha vector (wit zero values)

cross `signature(object = "vm")`: returns the cross-validation error

error `signature(object = "vm")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values (predict on training set)

kernelf `signature(object = "vm")`: returns the kernel function

kpar `signature(object = "vm")`: returns the kernel parameters (hyperparameters)

lev `signature(object = "vm")`: returns the levels in case of classification

kcall `signature(object="vm")`: returns the function call

type `signature(object = "vm")`: returns the problem type

xmatrix `signature(object = "vm")`: returns the data matrix used(support vectors)

ymatrix `signature(object = "vm")`: returns the response vector

Author(s)

Alexandros Karatzoglou
`<alexandros.karatzolgou@ci.tuwien.ac.at>`

See Also

`ksvm-class`, `rvm-class`, `gausspr-class`

Index