Package ‘KERE’

October 12, 2022

Title  Expectile Regression in Reproducing Kernel Hilbert Space
Version  1.0.0
Date  2015-8-26
Author  Yi Yang, Teng Zhang, Hui Zou
Maintainer  Yi Yang <yiyang@umn.edu>
Depends  methods
Description  An efficient algorithm inspired by majorization-minimization principle for solving the entire solution path of a flexible nonparametric expectile regression estimator constructed in a reproducing kernel Hilbert space.
License  GPL-2
Repository  CRAN
Date/Publication  2015-08-28 00:35:09
NeedsCompilation  yes

R topics documented:

  as.kernelMatrix  .......................................................... 2
cv.KERE  ................................................................. 3
dots  ................................................................. 5
KERE  .............................................................. 7
kernel-class  .................................................. 9
kernelMatrix  .................................................. 10
plot.KERE  .................................................. 12
predict.KERE  .................................................. 13

Index  16
as.kernelMatrix

Assigning kernelMatrix class to matrix objects

Description

as.kernelMatrix in package KERE 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 KERE 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
Cross-validation for KERE

Description

Does k-fold cross-validation for KERE, produces a plot, and returns a value for lambda.

Usage

```r
## S3 method for class 'KERE'
cv(x, y, kern, lambda = NULL, nfolds = 5, foldid, omega = 0.5, ...)
```

Arguments

- `x`: matrix of predictors, of dimension $N \times p$; each row is an observation vector.
- `y`: response variable.
- `kern`: the built-in kernel classes in KERE. The `kern` parameter can be set to any function, of class kernel, which computes the inner product in feature space between two vector arguments. KERE 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.

Objects can be created by calling the `rbfdot`, `polydot`, `tanhdot`, `vanilladot`, `anovadot`, `besseldot`, `laplacedot`, `splinedot` functions etc. (see example.)

- `lambda`: a user supplied lambda sequence. It is better to supply a decreasing sequence of lambda values, if not, the program will sort user-defined lambda sequence in decreasing order automatically.

- `nfolds`: number of folds - default is 5. Although `nfolds` can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is `nfolds=3`.

- `foldid`: an optional vector of values between 1 and `nfold` identifying what fold each observation is in. If supplied, `nfold` can be missing.

- `omega`: the parameter $\omega$ in the expectile regression model. The value must be in (0,1). Default is 0.5.

- `...`: other arguments that can be passed to KERE.
Details

The function runs KERE \( n_{folds} + 1 \) times; the first to get the \( \lambda \) sequence, and then the remainder to compute the fit with each of the folds omitted. The average error and standard deviation over the folds are computed.

Value

an object of class \textit{cv.KERE} is returned, which is a list with the ingredients of the cross-validation fit.

- \texttt{lambda} the values of \( \lambda \) used in the fits.
- \texttt{cvm} the mean cross-validated error - a vector of length \( \text{length}(\lambda) \).
- \texttt{cvsd} estimate of standard error of \texttt{cvm}.
- \texttt{cvupper} upper curve = \texttt{cvm}+\texttt{cvsd}.
- \texttt{cvlo} lower curve = \texttt{cvm}-\texttt{cvsd}.
- \texttt{name} a character string "Expectile Loss"
- \texttt{lambda.min} the optimal value of \( \lambda \) that gives minimum cross validation error \texttt{cvm}.
- \texttt{cvm.min} the minimum cross validation error \texttt{cvm}.

Author(s)

Yi Yang, Teng Zhang and Hui Zou
Maintainer: Yi Yang <yiyang@umn.edu>

References


Examples

```r
N <- 200
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- 3*runif(N)
SNR <- 10 # signal-to-noise ratio
Y <- X1**1.5 + 2 * (X2**.5) + X1*X3
sigma <- sqrt(var(Y)/SNR)
Y <- Y + X2*rnorm(N,0,sigma)
X <- cbind(X1,X2,X3)

# set gaussian kernel
kern <- rbfdot(sigma=0.1)

# define lambda sequence
lambda <- exp(seq(log(0.5),log(0.01),len=10))

cv.KERE(x=X, y=Y, kern, lambda = lambda, nfolds = 5, omega = 0.5)
```
Kernel Functions

Description

The kernel generating functions provided in KERE.

The Gaussian RBF kernel
\[ k(x, x') = \exp(-\sigma \|x - x'\|^2) \]

The Polynomial kernel
\[ k(x, x') = (\text{scale} \times x, x') + \text{offset} \]

The Linear kernel
\[ k(x, x') = \langle x, x' \rangle \]

The Hyperbolic tangent kernel
\[ k(x, x') = \tanh(\text{scale} \times x, x') + \text{offset} \]

The Laplacian kernel
\[ k(x, x') = \exp(-\sigma \|x - x'\|^2) \]

The Bessel kernel
\[ k(x, x') = -\text{Bessel}_{\nu+1}(\sigma) \times \langle x, x' \rangle \]

The ANOVA RBF kernel
\[ k(x, x') = \sum_{1 \leq i_1 < \cdots < i_D \leq N} \prod_{d=1}^{D} k(x_{i_d}, x'_{i_d}) \text{ where } k(x, x) \text{ is a Gaussian RBF kernel.} \]

The Spline kernel
\[ \prod_{d=1}^{D} 1 + x_i x_j + x_i x_j \min(x_i, x_j) - \frac{x_i + x_j}{2} \min(x_i, x_j)^2 + \frac{\min(x_i, x_j)^3}{3} \]

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 a 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 KERE (e.g., ksvm, kpca etc).

Although using one of the existing kernel functions as a kernel argument in various functions in KERE 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.

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

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)

KERE

Fits the regularization paths for the kernel expectile regression.

Description
Fits a regularization path for the kernel expectile regression at a sequence of regularization parameters lambda.

Usage
KERE(x, y, kern, lambda = NULL, eps = 1e-08, maxit = 1e4, omega = 0.5, gamma = 1e-06, option = c("fast", "normal"))

Arguments
- x: matrix of predictors, of dimension $N \times p$; each row is an observation vector.
- y: response variable.
- kern: the built-in kernel classes in KERE. The kern parameter can be set to any function, of class kernel, which computes the inner product in feature space between two vector arguments. KERE provides the most popular kernel functions which can be initialized by using the following functions:
  - rbf: Radial Basis kernel function,
  - poly: Polynomial kernel function,
  - vanilla: Linear kernel function,
  - tanh: Hyperbolic tangent kernel function,
  - laplace: Laplacian kernel function,
  - bessel: Bessel kernel function,
  - anova: ANOVA RBF kernel function,
  - spline: the Spline kernel.
- lambda: a user supplied lambda sequence. It is better to supply a decreasing sequence of lambda values, if not, the program will sort user-defined lambda sequence in decreasing order automatically.
- eps: convergence threshold for majorization minimization algorithm. Each majorization descent loop continues until the relative change in any coefficient $||\alpha(new) - \alpha(old)||_2^2/||\alpha(old)||_2^2$ is less than eps. Defaults value is 1e-8.
- maxit: maximum number of loop iterations allowed at fixed lambda value. Default is 1e4. If models do not converge, consider increasing maxit.
- omega: the parameter $\omega$ in the expectile regression model. The value must be in (0,1). Default is 0.5.
gamma a scalar number. If it is specified, the number will be added to each diagonal element of the kernel matrix as perturbation. The default is 1e-06.

option users can choose which method to use to update the inverse matrix in the MM algorithm. "fast" uses a trick described in Yang, Zhang and Zou (2015) to update estimates for each lambda. "normal" uses a naive way for the computation.

Details

Note that the objective function in KERE is

\[ \text{Loss}(y - \alpha_0 - K \alpha)) + \lambda \alpha^T K \alpha, \]

where the \( \alpha_0 \) is the intercept, \( \alpha \) is the solution vector, and \( K \) is the kernel matrix with \( K_{ij} = K(x_i, x_j) \). Users can specify the kernel function to use, options include Radial Basis kernel, Polynomial kernel, Linear kernel, Hyperbolic tangent kernel, Laplacian kernel, Bessel kernel, ANOVA RBF kernel, the Spline kernel. Users can also tweak the penalty by choosing different lambda.

For computing speed reason, if models are not converging or running slow, consider increasing eps before increasing maxit.

Value

An object with S3 class KERE.

- call the call that produced this object.
- alpha a nrow(x)*length(lambda) matrix of coefficients. Each column is a solution vector corresponding to a lambda value in the lambda sequence.
- lambda the actual sequence of lambda values used.
- npass total number of loop iterations corresponding to each lambda value.
- jerr error flag, for warnings and errors, 0 if no error.

Author(s)

Yi Yang, Teng Zhang and Hui Zou
Maintainer: Yi Yang <yiyang@umn.edu>

References


Examples

```r
# create data
N <- 200
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- 3*runif(N)
SNR <- 10  # signal-to-noise ratio
Y <- X1**1.5 + 2 * (X2**.5) + X1*X3
```
```r
sigma <- sqrt(var(Y)/SNR)
Y <- Y + X2*rnorm(N,0,sigma)
X <- cbind(X1,X2,X3)

# set gaussian kernel
kern <- rbfdot(sigma=0.1)

# define lambda sequence
lambda <- exp(seq(log(0.5),log(0.01),len=10))

# run KERE
m1 <- KERE(x=X, y=Y, kern=kern, lambda = lambda, omega = 0.5)

# plot the solution paths
plot(m1)
```

---

**kernel-class**

Class "kernel" "rbfkernel" "polykernel", "tanhkernel", "vanillakernel"

---

**Description**

The built-in kernel classes in KERE

**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")` or by calling the `rbfdot`, `polydot`, `tanhdot`, `vanilladot`, `anovadot`, `besseldot`, `laplacedot`, `splinedot` 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"): computes parts or the full kernel matrix, mainly used in kernel algorithms where columns of the kernel matrix are computed per invocation
kernelMatrix

Author(s)
Alexandros Karatzoglou
<alexandros.karatzoglou@ci.tuwien.ac.at>

See Also
dots

Examples

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

### kernelMatrix

<table>
<thead>
<tr>
<th>kernelMatrix</th>
<th>Kernel Matrix functions</th>
</tr>
</thead>
</table>

#### 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 $H = z_i z_j k(x_i, x_j)$, $H = z_i k(x_i, y_j)$.

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

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

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

## S4 method for signature 'kernel'
kernellMult(kernel, x, y = NULL, z, blocksize = 256)

## S4 method for signature 'kernel'
kernellFast(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., rbf dot etc.) or a user defined function of class kernel taking two vector arguments and returning a scalar.
**kernelMatrix**

- **x**: a data matrix to be used to calculate the kernel matrix.
- **y**: second data matrix to calculate the kernel matrix.
- **z**: a suitable vector or matrix
- **k**: a suitable vector or matrix
- **a**: the squared norm of \( x \), e.g., \( \text{rowSums}(x^2) \)
- **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. KERE 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

```r
# use the spam data
x <- matrix(rnorm(10*10),10,10)

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

# calculate kernel matrix
kernelMatrix(rbf, x)

y <- matrix(rnorm(10*1),10,1)

# calculate the quadratic kernel expression
kernelPol(rbf, x, ,y)

# calculate the kernel expansion
kernelMult(rbf, x, ,y)
```

plot.KERE

---

`plot.KERE`  
`Plot coefficients from a "KERE" object`

Description

Produces a coefficient profile plot of the coefficient paths for a fitted KERE object.

Usage

```r
## S3 method for class 'KERE'
plot(x, ...)
```

Arguments

- `x` : fitted KERE model.
- `...` : other graphical parameters to plot.

Details

A coefficient profile plot is produced. The x-axis is $\log(\lambda)$. The y-axis is the value of fitted $\alpha$'s.

Author(s)

Yi Yang, Teng Zhang and Hui Zou
Maintainer: Yi Yang <yiyang@umn.edu>
References


Examples

```r
# create data
N <- 200
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- 3*runif(N)
SNR <- 10 # signal-to-noise ratio
Y <- X1**1.5 + 2 * (X2**.5) + X1*X3
sigma <- sqrt(var(Y)/SNR)
Y <- Y + X2*rnorm(N,0,sigma)
X <- cbind(X1,X2,X3)

# set gaussian kernel
kern <- rbfdot(sigma=0.1)

# define lambda sequence
lambda <- exp(seq(log(0.5),log(0.01),len=10))

# run KERE
m1 <- KERE(x=X, y=Y, kern=kern, lambda = lambda, omega = 0.5)

# plot the solution paths
plot(m1)
```

`predict.KERE`  
make predictions from a "KERE" object.

Description

Similar to other predict methods, this function predicts fitted values and class labels from a fitted KERE object.

Usage

```r
## S3 method for class 'KERE'
predict(object, kern, x, newx,...)
```

Arguments

- `object`: fitted KERE model object.
- `kern`: the built-in kernel classes in KERE. Objects can be created by calling the rbfdot, polydot, tanhdot, vanilladot, anovadot, besseldot, laplacedot, splinedot functions etc. (see example.)
predict.KERE

x

the original design matrix for training KERE.

newx

matrix of new values for x at which predictions are to be made. NOTE: newx must be a matrix with each row as an observation. predict function does not accept a vector or other formats of newx.

...

other parameters to predict function.

Details

The fitted $\alpha_0 + K * \alpha$ at newx is returned as a size nrow(newx) x length(lambda) matrix for various lambda values where the KERE model was fitted.

Value

The fitted $\alpha_0 + K * \alpha$ is returned as a size nrow(newx) x length(lambda) matrix. The row represents the index for observations of newx. The column represents the index for the lambda sequence.

Author(s)

Yi Yang, Teng Zhang and Hui Zou
Maintainer: Yi Yang <yiyang@umn.edu>

References


Examples

```r
# create data
N <- 100
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- 3*runif(N)
SNR <- 10 # signal-to-noise ratio
Y <- X1^1.5 + 2 * (X2^0.5) + X1*X3
sigma <- sqrt(var(Y)/SNR)
Y <- Y + X2*rnorm(N,0,sigma)
X <- cbind(X1,X2,X3)

# set gaussian kernel
kern <- rbfdot(sigma=0.1)

# define lambda sequence
lambda <- exp(seq(log(0.5),log(0.01),len=10))

# run KERE
m1 <- KERE(x=X, y=Y, kern=kern, lambda = lambda, omega = 0.5)

# create newx for prediction
N1 <- 5
X1 <- runif(N1)
```
X2 <- 2*runif(N1)
X3 <- 3*runif(N1)
newx <- cbind(X1,X2,X3)

# make prediction
p1 <- predict.KERE(m1, kern, X, newx)
p1
Index

* algebra
  * kernelMatrix, 10
* array
  * kernelMatrix, 10
* classes
  * kernel-class, 9
* methods
  * as.kernelMatrix, 2
* models
  * cv.KERE, 3
  * KERE, 7
  * plot.KERE, 12
  * predict.KERE, 13
* regression
  * cv.KERE, 3
  * KERE, 7
  * plot.KERE, 12
  * predict.KERE, 13
* symbolmath
  * dots, 5
  * anovadot (dots), 5
  * anovakernel-class (kernel-class), 9
  * as.kernelMatrix, 2
  * as.kernelMatrix, matrix-method
    * (as.kernelMatrix), 2
  * as.kernelMatrix-methods
    * (as.kernelMatrix), 2
  * besseldot (dots), 5
  * besselkernel-class (kernel-class), 9
  * cv.KERE, 3, 4
  * dots, 2, 5, 10
  * fourierdot (dots), 5
  * fourierkernel-class (kernel-class), 9
  * KERE, 4, 7, 8, 12, 13
  * kernel-class, 9
  * kernelFast (kernelMatrix), 10
  * kernelFast, anovakernel-method
    (kernelMatrix), 10
  * kernelFast, besselkernel-method
    (kernelMatrix), 10
  * kernelFast, kernel-method
    (kernelMatrix), 10
  * kernelFast, laplacekernel-method
    (kernelMatrix), 10
  * kernelFast, polykernel-method
    (kernelMatrix), 10
  * kernelFast, rbfkernel-method
    (kernelMatrix), 10
  * kernelFast, splinekernel-method
    (kernelMatrix), 10
  * kernelFast, tanhkernel-method
    (kernelMatrix), 10
  * kernelFast, vanillakernel-method
    (kernelMatrix), 10
  * kernelMatrix, 2, 6, 10
  * kernelMatrix, anovakernel-method
    (kernelMatrix), 10
  * kernelMatrix, besselkernel-method
    (kernelMatrix), 10
  * kernelMatrix, kernel-method
    (kernelMatrix), 10
  * kernelMatrix, laplacekernel-method
    (kernelMatrix), 10
  * kernelMatrix, polykernel-method
    (kernelMatrix), 10
  * kernelMatrix, rbfkernel-method
    (kernelMatrix), 10
  * kernelMatrix, splinekernel-method
    (kernelMatrix), 10
  * kernelMatrix, tanhkernel-method
    (kernelMatrix), 10
  * kernelMatrix, vanillakernel-method
    (kernelMatrix), 10
  * kernelMatrix-class (as.kernelMatrix), 2
INDEX

kernelMult, 6
kernelMult (kernelMatrix), 10
kernelMult, anovakernel, ANY-method
(kernelMatrix), 10
kernelMult, besselkernel, ANY-method
(kernelMatrix), 10
kernelMult, character, kernelMatrix-method
(kernelMatrix), 10
kernelMult, kernel-method
(kernelMatrix), 10
kernelMult, laplacekernel, ANY-method
(kernelMatrix), 10
kernelMult, polykernel, ANY-method
(kernelMatrix), 10
kernelMult, rbfkernel, ANY-method
(kernelMatrix), 10
kernelMult, splinekernel, ANY-method
(kernelMatrix), 10
kernelMult, tanhkernel, ANY-method
(kernelMatrix), 10
kernelMult, vanillakernel, ANY-method
(kernelMatrix), 10

kernelPol, 6
kernelPol (kernelMatrix), 10
kernelPol, anovakernel-method
(kernelMatrix), 10
kernelPol, besselkernel-method
(kernelMatrix), 10
kernelPol, kernel-method (kernelMatrix),
10
kernelPol, laplacekernel-method
(kernelMatrix), 10
kernelPol, polykernel-method
(kernelMatrix), 10
kernelPol, rbfkernel-method
(kernelMatrix), 10
kernelPol, splinekernel-method
(kernelMatrix), 10
kernelPol, tanhkernel-method
(kernelMatrix), 10
kernelPol, vanillakernel-method
(kernelMatrix), 10
kernels (dots), 5
kfunction (dots), 5
kfunction-class (kernel-class), 9
kpar (dots), 5
kpar, kernel-method (kernel-class), 9
laplacedot (dots), 5

laplacekernel-class (kernel-class), 9
plot.KERE, 12
polydot, 11
polydot (dots), 5
polykernel-class (kernel-class), 9
predict.KERE, 13
rbdot, 11
rbdot (dots), 5
rbfkernel-class (kernel-class), 9
show, kernel-method (dots), 5
spliedot (dots), 5
splinekernel-class (kernel-class), 9
tanhdot, 11
tanhdot (dots), 5
tanhkernel-class (kernel-class), 9
vanilladot, 11
vanilladot (dots), 5
vanillakernel-class (kernel-class), 9