Package ‘ktweedie’

October 15, 2023

Title 'Tweedie' Compound Poisson Model in the Reproducing Kernel Hilbert Space

Version 1.0.3

Date 2023-10-14

Description Kernel-based 'Tweedie' compound Poisson gamma model using high-dimensional predictors for the analyses of zero-inflated response variables. The package features built-in estimation, prediction and cross-validation tools and supports choice of different kernel functions. For more details, please see Yi Lian, Archer Yi Yang, Boxiang Wang, Peng Shi & Robert William Platt (2023) <doi:10.1080/00401706.2022.2156615>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Depends methods, R (>= 3.5.0)

Suggests knitr, rmarkdown, tweedie

LazyData true

VignetteBuilder knitr

NeedsCompilation yes

Author Yi Lian [aut, cre],
Archer Yi Yang [aut, cph],
Boxiang Wang [aut],
Peng Shi [aut],
Robert W. Platt [aut]

Maintainer Yi Lian <yi.lian@mail.mcgill.ca>

Repository CRAN

Date/Publication 2023-10-14 22:32:43 UTC

R topics documented:

as.kernelMatrix *Assing 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
```

`dat` *A demo dataset*

Description

A simulated dataset with covariate matrix `x` of size 30 x 5 and an outcome vector `y` of length 30.

Usage

```
data(dat)
```

Format

A list with 2 items:

x Covariate matrix

y Outcome vector

Details

`x` is generated from standard normal distribution. `y` is generated from Tweedie distribution with mean equal to exp(sin(x) %*% (6, -4, 0, 0, 0)). Only the first two variables are associated with the outcome.

`dots` *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 $k(x,x') = (scale <x, x'> + offset)^{degree}$
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 $\prod_{d=1}^D 1 + x_i x_j + x_i x_j min(x_i, x_j) - \frac{x_i + x_j}{2} min(x_i,x_j)^2 + \frac{min(x_i,x_j)^3}{3}$ \

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 **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)
```

`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"),,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.
`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., `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

```
## 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)
```

`ktd_cv` *Cross validation for tuning the regularization coefficient in the kernel Tweedie model*

Description

`ktd_cv()` performs cross-validation to determine the optimal regularization coefficient of the `ktweedie` model.

Usage

```
ktd_cv(x, y, kern, lambda, nfolds = 5, rho = 1.5, loss = "LL", ...)
```

Arguments

`x`	Covariate matrix.
`y`	Outcome vector (e.g. insurance cost).
`kern`	Choice of kernel. See `dots` for details on supported kernel functions.
`lambda`	A vector of candidate regularization coefficients used in cross-validation.
`nfolds`	Number of folds in cross-validation. Default is 5.
`rho`	The power parameter of the Tweedie model. Default is 1.5 and can take any real value between 1 and 2.
`loss`	Criterion used in cross-validation. "LL" for log likelihood, "RMSE" for root mean squared error, "MAD" for mean absolute difference. Default is "LL".
`...`	Optional arguments to be passed to `ktd_estimate()`.

Details

`ktd_cv()` is a built-in wrapper for cross-validation for the choice of regularization coefficient.

Value

A list of two items.

1. LL or RMSE or MAD: a vector of validation error based on the user-specified `loss`, named by the corresponding `lambda` values;
2. Best_lambda: the `lambda` value in the pair that generates the best loss;

See Also

`ktd_cv2d`, `ktd_estimate`, `ktd_predict`

Examples

```
# Provide a sequence of candidate values to the argument lambda.
# ktd_cv() will perform cross-validation to determine which is the best.
( cv1d <- ktd_cv(x = dat$x, y = dat$y,
                 kern = rbfdot(sigma = 1e-8),
                 lambda = 10^(-8:-1),
                 nfolds = 5) )
```

ktd_cv2d *Cross validation for jointly tuning the regularization coefficient and kernel parameter in the Kernel Tweedie Model*

Description

`ktd_cv2d()` performs 2-dimensional random search from user-specified ranges to determine the optimal pair of regularization coefficient and kernel parameter of the `ktweedie` model.

Usage

```
ktd_cv2d(
  x,
  y,
  kernfunc,
  lambda,
  sigma,
  ncoefs,
  nfolds = 5,
  rho = 1.5,
  loss = "LL",
  ...
)
```

Arguments

`x`	Covariate matrix.
`y`	Outcome vector (e.g. insurance cost).
`kernfunc`	Choice of kernel function. See dots for details on supported kernel functions.
`lambda`	A vector of length two indicating the lower and upper bound from which candidate regularization coefficient values are sampled uniformly on the log scale.
`sigma`	A vector of length two indicating the lower and upper bound from which candidate kernel parameter values are sampled uniformly on the log scale.
`ncoefs`	The number of candidate `lambda` and `sigma` pairs to be evaluated.
`nfolds`	Number of folds in cross-validation. Default is 5.
`rho`	The power parameter of the Tweedie model. Default is 1.5 and can take any real value between 1 and 2.
`loss`	Criterion used in cross-validation. "LL" for log likelihood, "RMSE" for root mean squared error, "MAD" for mean absolute difference. Default is "LL".
`...`	Optional arguments to be passed to `ktd_estimate()`.

Details

`ktd_cv2d()` is a built-in wrapper for 2D random search for the regularization coefficient and kernel parameter. For kernel functions with greater than one parameters, `ktd_cv2d()` supports the tuning of the first one.

Value

A list of three items.

1. LL or RMSE or MAD: a vector of validation error based on the user-specified `loss`, named by the corresponding `lambda` and `sigma` values;
2. Best_lambda: the `lambda` value in the pair that generates the best loss;
3. Best_sigma: the `sigma` value in the pair that generates the best loss.

See Also

`ktd_cv`, `ktd_estimate`, `ktd_predict`

Examples

```
### Cross-validation
# Provide the kernel function name (e.g. rbfdot) to the argument kernfunc,
# NOT the kernel function object, e.g. rbfdot(sigma = 1).
# Provide ranges where the candidate lambdas and sigmas are drawn from
# to the arguments lambda and sigma.
# The number of pairs of candidates to select from is specified by ncoefs.
( cv2d <- ktd_cv2d(x = dat$x, y = dat$y,
                   kernfunc = rbfdot,
                   lambda = c(1e-3, 1e0),
                   sigma = c(1e-3, 1e0),
                   ncoefs = 10) )
### Followed by fitting
fit <- ktd_estimate(x = dat$x, y = dat$y,
                    kern = rbfdot(sigma = cv2d$Best_sigma),
                    lam1 = cv2d$Best_lambda)
```

`ktd_estimate` *Estimate kernel Tweedie model coefficients*

Description

`ktd_estimate()` estimates the coefficients of the kernel Tweedie model `ktweedie` and the sparse kernel Tweedie model `sktweedie`. The log of the expected Tweedie mean is modeled by a function in the reproducing kernel Hilbert space. The `sktweedie` has an integrated feature selection component that induces sparsity by applying weights on the features and penalizing the weights.

Usage

```
ktd_estimate(
  x,
  y,
  kern,
  lam1,
  rho = 1.5,
  ftol = 1e-08,
  partol = 1e-08,
  abstol = 0,
  maxit = 1e+06,
  sparsity = FALSE,
  lam2 = 0,
  innerpartol = 1e-06,
  innermaxit = 1e+06,
  verbose = FALSE
)
```

Arguments

`x`	Covariate matrix.
`y`	Outcome vector (e.g. insurance cost).
`kern`	Choice of kernel. See `dots` for details on supported kernel functions.
`lam1`	A vector of regularization coefficients.
`rho`	The power parameter of the Tweedie model. Default is 1.5 and can take any real value between 1 and 2.
`ftol`	Stopping criterion based on objective function value. Default is 1e-8. See Details.
`partol`	Stopping criterion based on the coefficient values. Default is 1e-8. See Details.
`abstol`	Stopping criterion based on absolute value of the objective function. Default is 0.
`maxit`	Maximum number of iterations.
`sparsity`	Logical If true, the `sktweedie` model with variable selection will be used. Default is false, for the `ktweedie` model.
`lam2`	Regularization coefficient for the sparsity-inducing penalty in the `sktweedie` model.
`innerpartol`	Stopping criterion for the inner loops that update kernel parameters and weights based on the coefficient values. See Details.
`innermaxit`	Maximum number of iterations for the inner loops that update kernel parameters and variable weights. See Details.
`verbose`	Logical indicating whether to show details of each update.

Details

`ktd_estimate()` stops when the absolute difference between the objective function values of the last two updates is smaller than `ftol`, or the sum of absolute differences between the coefficients of the last two updates is smaller than `partol`, or the objective function values is below `abstol`, before `maxit` is reached. For the `sktweedie` model, there are inner loops for the update of kernel regression coefficients and regularization weights. The `innerpartol` and `innermaxit` arguments are the counterparts of `partol` and `maxit` for the inner loops.

Value

A list of three items.

1. `estimates`: a list containing the final objective function values and kernel Tweedie regression coefficients for each `lam1`.
2. `data`: stores the inputs, including the predictor matrix, the kernel function used in the fitting and `lam1`.
3. `sparsity`: a logical variable indicating whether the `ktweedie` or `sktweedie` is fitted.

See Also

`ktd_cv`, `ktd_cv2d`, `ktd_predict`, `rbfdot`

Examples

```
###### ktweedie ######
# Provide a sequence of candidate values to the argument lam1.
# Provide a kernel object to the argument kern.
lam1.seq <- c(1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1)
fit.ktd <- ktd_estimate(x = dat$x, y = dat$y,
                        kern = rbfdot(sigma = 1e-8),
                        lam1 = lam1.seq)
###### sktweedie ######
# Set sparsity to TRUE and a lam2 to control the level of sparsity
# Decrease lam2 if "WARNING: All weights are zero..."
fit.sktd <- ktd_estimate(x = dat$x,
                         y = dat$y,
                         kern = rbfdot(sigma = 0.1),
                         lam1 = 5,
                         sparsity = TRUE,
                         lam2 = 1)
# variables with fitted weight equal to 0 are not selected
```

`ktd_predict` *Predict outcome using fitted kernel Tweedie model*

Description

`ktd_predict()` predicts the outcome with fitted `ktweedie` or `sktweedie` model at the user supplied new data.

Usage

```
ktd_predict(model, newdata, which.lam1 = 1, type = "link")
```

Arguments

`model`	Fitted model from `ktd_estimate`
`newdata`	New x matrix for the prediction. If not provided, it will be the x matrix used to fit `model`.
`which.lam1`	The index of the `lam1` in `model` used in the prediction. Default is 1.
`type`	The type of prediction to be made - "`link`" for the linear predictor and "`response`" for the predicted outcome. Default is "`link`".

Details

`ktd_predict()` uses the fitted model from `ktd_estimate` to estimate the mean outcome for new data points.

Value

A list named `prediction` containing the vector of predicted outcomes.

See Also

`ktd_estimate`, `ktd_cv`, `ktd_cv2d`

Examples

```
# Fit a ktweedie model
fit <- ktd_estimate(x = dat$x, y = dat$y,
                    kern = rbfdot(sigma = 1e-6),
                    lam1 = 10^(-5:1))
# Generate newx at which predictions are to be made.
# The newdata should have the same dimension as the original trainig data.
newx <- matrix(rnorm(10 * ncol(dat$x)), nrow = 10)
pred <- ktd_predict(model = fit, newdata = newx,
                    which.lam1 = 3, type = "link")
```

Index