Package ‘iprior’

April 2, 2024

Title Regression Modelling using I-Priors
Version 0.7.4
Encoding UTF-8
Description Provides methods to perform and analyse I-prior regression models. Estimation is done either via direct optimisation of the log-likelihood or an EM algorithm.

URL https://github.com/haziqj/iprior
BugReports https://github.com/haziqj/iprior/issues
License GPL (>= 3.0)
Language en-GB
Depends R (>= 3.2.5)
Imports doSNOW, foreach, ggplot2, mvtorm, Rcpc (>= 0.12.5), reshape2, scales
LinkingTo Rcpc, RcppEigen
Suggests caret, knitr, MASS, R.rsp, rmarkdown, testthat
LazyData yes
VignetteBuilder knitr, R.rsp
RoxxygenNote 7.2.3
NeedsCompilation yes
Author Haziq Jamil [aut, cre]
Maintainer Haziq Jamil <haziq.jamil@gmail.com>
Repository CRAN
Date/Publication 2024-04-01 23:10:07 UTC

R topics documented:

  Accessors .......................................................... 2
  as.time ............................................................. 4
  check_theta ...................................................... 5
## Accessors

Accessor functions for `ipriorMod` objects.

### Description

Accessor functions for `ipriorMod` objects.

### Usage

```r
get_intercept(object)
get_y(object)
get_size(object, units = "kB", standard = "SI")
get_hyp(object)
get_lambda(object)
get_psi(object)
get_lengthscale(object)
```
**Accessors**

- get_hurst(object)
- get_offset(object)
- get_degree(object)
- get_se(object)
- get_kernels(object)
- get_kern_matrix(object, theta = NULL, newdata)
- get_prederror(object, error.type = c("RMSE", "MSE"))
- get_estl(object)
- get_method(object)
- get_convergence(object)
- get_niter(object)
- get_time(object)
- get_theta(object)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An ipriorMod object.</td>
</tr>
<tr>
<td>units</td>
<td>Units for object size.</td>
</tr>
<tr>
<td>standard</td>
<td>Standard for object size.</td>
</tr>
<tr>
<td>theta</td>
<td>(Optional) Value of hyperparameters to evaluate the kernel matrix.</td>
</tr>
<tr>
<td>newdata</td>
<td>(Optional) If not supplied, then a square, symmetric kernel matrix is returned using the data as input points. Otherwise, the kernel matrix is evaluated with respect to this set of data as well. It must be a list of vectors/matrices with similar dimensions to the original data.</td>
</tr>
<tr>
<td>error.type</td>
<td>(Optional) Report the mean squared error of prediction (&quot;MSE&quot;), or the root mean squared error of prediction (&quot;RMSE&quot;)</td>
</tr>
</tbody>
</table>

**Functions**

- get_intercept(): Obtain the intercept.
- get_y(): Obtain the response variables.
- get_size(): Obtain the object size of the I-prior model.
- get_hyp(): Obtain the hyperparameters of the model (both estimated and fixed ones).
• get_lambda(): Obtain the scale parameters used.
• get_psi(): Obtain the error precision.
• get_lengthscale(): Obtain the lengthscale for the SE kernels used.
• get_hurst(): Obtain the Hurst coefficient of the fBm kernels used.
• get_offset(): Obtain the offset parameters for the polynomial kernels used.
• get_degree(): Obtain the degree of the polynomial kernels used.
• get_se(): Obtain the standard errors of the estimated hyperparameters.
• get_kernels(): Obtain the kernels used.
• get_kern_matrix(): Obtain the kernel matrix of the I-prior model.
• get_prederror(): Obtain the training mean squared error.
• get_estl(): Obtain information on which hyperparameters were estimated and which were fixed.
• get_method(): Obtain the estimation method used.
• get_convergence(): Obtain the convergence information.
• get_niter(): Obtain the number of iterations performed.
• get_time(): Obtain the time taken to complete the estimation procedure.
• get_theta(): Extract the theta value at convergence. Note that this is on an unrestricted scale (see the vignette for details).

---

**as.time**

*Convert difftime class into time class*

**Description**

Convert difftime class into time class

**Usage**

```r
as.time(x)
```

**Arguments**

- `x` A difftime object.

**Value**

A time object which contains the time difference and units.
check_theta

Check the structure of the hyperparameters of an I-prior model

Usage
check_theta(object)

Arguments
object An ipriorMod object or an ipriorKernel object.

Value
A printout of the structure of the hyperparameters.

decimal_place

Cut a numeric vector to a certain number of decimal places

Usage
decimal_place(x, k = 2)
dec_plac(x, k = 2)

Arguments
x A numeric vector.
k The number of decimal places.

Value
A character vector with the correct number of decimal places.

Examples
decimal_place(pi, 3)
decimal_place(c(exp(1), pi, sqrt(2)), 4)
eigenCpp  
Eigen decomposition of a matrix in C++.

**Description**
Returns the eigenvalues and eigenvectors of a matrix X.

**Usage**
eigenCpp(X)

**Arguments**
- X  A symmetric, positive-definite matrix

**Details**
A fast implementation of eigen for symmetric, positive-definite matrices. This helps speed up the I-prior EM algorithm.

---

fastSquare  
Multiplying a symmetric matrix by itself in C++.

**Description**
Returns the square of a symmetric matrix X.

**Usage**
fastSquare(X)

**Arguments**
- X  A symmetric matrix

**Details**
A fast implementation of $X^2$ for symmetric matrices. This helps speed up the I-prior EM algorithm.
fastVDiag

Computing a quadratic matrix form in C++.

Description

Returns XdiagyXT.

Usage

fastVDiag(X, y)

Arguments

X A symmetric, square matrix of dimension n by n
y A vector of length n

Details

A fast implementation of XdiagyXT. This helps speed up the I-prior EM algorithm.

gen_multilevel

Generate simulated data for multilevel models

Description

Generate simulated data for multilevel models

Usage

gen_multilevel(
  n = 25,
  m = 6,
  sigma_e = 2,
  sigma_u0 = 2,
  sigma_u1 = 2,
  sigma_u01 = -2,
  beta0 = 0,
  beta1 = 2,
  x.jitter = 0.5,
  seed = NULL
)
Arguments

- **n**  
  Sample size. Input either a single number for a balanced data set, or a vector of length \( m \) indicating the sample size in each group.

- **m**  
  Number of groups/levels.

- **sigma_e**  
  The standard deviation of the errors.

- **sigma_u0**  
  The standard deviation of the random intercept.

- **sigma_u1**  
  The standard deviation of the random slopes.

- **sigma_u01**  
  The covariance of between the random intercept and the random slope.

- **beta0**  
  The mean of the random intercept.

- **beta1**  
  The mean of the random slope.

- **x.jitter**  
  A small amount of jitter is added to the \( X \) variables generated from a normal distribution with mean zero and standard deviation equal to \( x.jitter \).

- **seed**  
  (Optional) Random seed.

Value

A dataframe containing the response variable \( y \), the unidimensional explanatory variables \( X \), and the levels/groups (factors).

Examples

```r
gen_multilevel()
```

Description

Generate simulated data for smoothing models

Usage

```r
gen_smooth(n = 150, xlim = c(0.2, 4.6), x.jitter = 0.65, seed = NULL)
```

Arguments

- **n**  
  Sample size.

- **xlim**  
  Limits of the \( X \) variables to generate from.

- **x.jitter**  
  A small amount of jitter is added to the \( X \) variables generated from a normal distribution with mean zero and standard deviation equal to \( x.jitter \).

- **seed**  
  (Optional) Random seed.
**Value**

A dataframe containing the response variable y and unidimensional explanatory variable X.

**Examples**

```r
gen_smooth(10)
```

---

**gg_colour_hue**

*Emulate ggplot2 default colour palette*

**Description**

Emulate ggplot2 default colour palette. ipriorColPal and ggColPal are DEPRECATED.

**Usage**

```r
gg_colour_hue(x, h = c(0, 360) + 15, c = 100, l = 65)
gg_color_hue(x, h = c(0, 360) + 15, c = 100, l = 65)
gg_col_hue(x, h = c(0, 360) + 15, c = 100, l = 65)
ipriorColPal(x)
```

**Arguments**

- `x` The number of colours required.
- `h` Range of hues to use, in [0, 360].
- `c` Chroma (intensity of colour), maximum value varies depending on combination of hue and luminance.
- `l` Luminance (lightness), in [0, 100].

**Details**

This is the default colour scale for categorical variables in ggplot2. It maps each level to an evenly spaced hue on the colour wheel. It does not generate colour-blind safe palettes.

ipriorColPal() used to provide the colour palette for the iprior package, but this has been changed ggplot2’s colour palette instead.
Description

A national longitudinal survey of students from public and private high schools in the United States, with information such as students’ cognitive and non-cognitive skills, high school experiences, work experiences and future plans collected.

Usage

hsb

Format

A data frame of 7185 observations on 3 variables.

mathach  Math achievement.

ses  Socio-Economic status.

schoolid  Categorical variable indicating the school the student went to. Treated as factor.

Source

High School and Beyond, 1980: A Longitudinal Survey of Students in the United States (ICPSR 7896)

References


Examples

data(hsb)
str(hsb)
hsbsmall

Description

Smaller subset of hsb.

Usage

hsbsmall

Format

A data frame of 661 observations on 3 variables.

mathach  Math achievement.

ses  Socio-Economic status.

schoolid  Categorical variable indicating the school the student went to. Treated as factor.

Details

A random subset of size 16 out of the original 160 groups.

Examples

data(hsbsmall)
str(hsbsmall)

iprior

Fit an I-prior regression model

Description

A function to perform regression using I-priors. The I-prior model parameters may be estimated in a number of ways: direct minimisation of the marginal deviance, EM algorithm, fixed hyperparameters, or using a Nystrom kernel approximation.
Usage

## Default S3 method:
iprior(
y,
..., 
kernl = "linear",
method = "direct",
control = list(),
interactions = NULL,
est.lambda = TRUE,
est.hurst = FALSE,
est.lengthscale = FALSE,
est.offset = FALSE,
est.psi = TRUE,
fixed.hyp = NULL,
lambda = 1,
psi = 1,
nystrom = FALSE,
nys.seed = NULL,
model = list(),
train.samp,
test.samp,
intercept
)

## S3 method for class 'formula'
iprior(
formula,
data,
kernl = "linear",
one.lam = FALSE,
method = "direct",
control = list(),
est.lambda = TRUE,
est.hurst = FALSE,
est.lengthscale = FALSE,
est.offset = FALSE,
est.psi = TRUE,
fixed.hyp = NULL,
lambda = 1,
psi = 1,
nystrom = FALSE,
nys.seed = NULL,
model = list(),
train.samp,
test.samp,
intercept,
...
### S3 method for class 'ipriorKernel'

```r
iprior(object, method = "direct", control = list(), ...)
```

### S3 method for class 'ipriorMod'

```r
iprior(object, method = NULL, control = list(), iter.update = 100, ...)
```

#### Arguments

- **y**
  - Vector of response variables
- **...**
  - Only used when fitting using non-formula, enter the variables (vectors or matrices) separated by commas.
- **kernel**
  - Character vector indicating the type of kernel for the variables. Available choices are:
    - "linear" - (default) for the linear kernel
    - "canonical" - alternative name for "linear"
    - "fbm", "fbm,0.5" - for the fBm kernel with Hurst coefficient 0.5 (default)
    - "se", "se,1" - for the SE kernel with lengthscale 1 (default)
    - "poly", "poly2", "poly2,0" - for the polynomial kernel of degree 2 with offset 0 (default)
    - "pearson" - for the Pearson kernel

  The kernel argument can also be a vector of length equal to the number of variables, therefore it is possible to specify different kernels for each variables. Note that factor type variables are assigned the Pearson kernel by default, and that non-factor types can be forced to use the Pearson kernel (not recommended).

- **method**
  - The estimation method. One of:
    - "direct" - for the direct minimisation of the marginal deviance using `optim()`'s L-BFGS method
    - "em" - for the EM algorithm
    - "mixed" - combination of the direct and EM methods
    - "fixed" - for just obtaining the posterior regression function with fixed hyperparameters (default method when setting `fixed.hyp = TRUE`)
    - "canonical" - an efficient estimation method which takes advantage of the structure of the linear kernel

- **control**
  - (Optional) A list of control options for the estimation procedure:
    - `maxit` The maximum number of iterations for the quasi-Newton optimisation or the EM algorithm. Defaults to 100.
    - `em.maxit` For method = "mixed", the number of EM steps before switching to direct optimisation. Defaults to 5.
    - `stop.crit` The stopping criterion for the EM and L-BFGS algorithm, which is the difference in successive log-likelihood values. Defaults to 1e-8.
    - `theta0` The initial values for the hyperparameters. Defaults to random starting values.
report The interval of reporting for the optim() function.
restarts The number of random restarts to perform. Defaults to 0. It’s also possible to set it to TRUE, in which case the number of random restarts is set to the total number of available cores.
no.cores The number of cores in which to do random restarts. Defaults to the total number of available cores.
omega The overrelaxation parameter for the EM algorithm - a value between 0 and 1.
interactions Character vector to specify the interaction terms. When using formulas, this is specified automatically, so is not required. Syntax is "a:b" to indicate variable a interacts with variable b.
est.lambda Logical. Estimate the scale parameters? Defaults to TRUE.
est.hurst Logical. Estimate the Hurst coefficients for fBm kernels? Defaults to FALSE.
est.lengthscale Logical. Estimate the lengthscales for SE kernels? Defaults to FALSE.
est.offset Logical. Estimate the offsets for polynomial kernels? Defaults to FALSE.
est.psi Logical. Estimate the error precision? Defaults to TRUE.
fixed.hyp Logical. If TRUE, then no hyperparameters are estimated, i.e. all of the above est.x are set to FALSE, and vice versa. If NULL (default) then all of the est.x defaults are respected.
lambda Initial/Default scale parameters. Relevant especially if est.lambda = FALSE.
psi Initial/Default value for error precision. Relevant especially if est.psi = FALSE.
nystrom Either logical or an integer indicating the number of Nystrom samples to take. Defaults to FALSE. If TRUE, then approximately 10% of the sample size is used for the Nystrom approximation.
nys.seed The random seed for the Nystrom sampling. Defaults to NULL, which means the random seed is not fixed.
model DEPRECATED.
train.samp (Optional) A vector indicating which of the data points should be used for training, and the remaining used for testing.
test.samp (Optional) Similar to train.samp, but on test samples instead.
intercept Optional intercept term.
formula The formula to fit when using formula interface.
data Data frame containing variables when using formula interface.
one.lam Logical. When using formula input, this is a convenient way of letting the function know to treat all variables as a single variable (i.e. shared scale parameter). Defaults to FALSE.
object An ipriorKernel or ipriorMod object.
iter.update The number of iterations to perform when calling the function on an ipriorMod object. Defaults to 100.
**iprior**

### Details

The `iprior()` function is able to take formula based input and non-formula. When not using formula, the syntax is as per the default S3 method. That is, the response variable is the vector y, and any explanatory variables should follow this, and separated by commas.

As described here, the model can be loaded first into an `ipriorKernel` object, and then passed to the `iprior()` function to perform the estimation.

### Value

An `ipriorMod` object. Several accessor functions have been written to obtain pertinent things from the `ipriorMod` object. The `print()` and `summary()` methods display the relevant model information.

### Methods (by class)

- `iprior(ipriorKernel)`: Takes in object of type `ipriorKernel`, a loaded and prepared I-prior model, and proceeds to estimate it.

- `iprior(ipriorMod)`: Re-run or continue running the EM algorithm from last attained parameter values in object `ipriorMod`.

### See Also

`optim`, `update`, `check_theta`, `print`, `summary`, `plot`, `coef`, `sigma`, `fitted`, `predict`, `logLik`, `deviance`.

### Examples

```r
# Formula based input
(mod.stackf <- iprior(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
                      data = stackloss))
mod.toothf <- iprior(len ~ supp * dose, data = ToothGrowth)
summary(mod.toothf)

# Non-formula based input
mod.stacknf <- iprior(y = stackloss$stack.loss, 
                      Air.Flow = stackloss$Air.Flow, 
                      Water.Temp = stackloss$Water.Temp, 
                      Acid.Conc. = stackloss$Acid.Conc.)
mod.toothnf <- iprior(y = ToothGrowth$len, ToothGrowth$supp, 
                      ToothGrowth$dose, 
                      interactions = "1:2")

# Formula based model option one.lam = TRUE
# Sets a single scale parameter for all variables
modf <- iprior(stack.loss ~ ., data = stackloss, one.lam = TRUE)
modnf <- iprior(y = stackloss$stack.loss, X = stackloss[1:3])
all.equal(coef(modnf), coef(modnf))  # both models are equivalent

# Fit models using different kernels
dat <- gen_smooth(n = 100)
mod <- iprior(y ~ X, dat, kernel = "fbm")  # Hurst = 0.5 (default)
```
mod <- iprior(y ~ X, dat, kernel = "poly3")  # polynomial degree 3

# Fit models using various estimation methods
mod1 <- iprior(y ~ X, dat)
mod2 <- iprior(y ~ X, dat, method = "em")
mod3 <- iprior(y ~ X, dat, method = "canonical")
mod4 <- iprior(y ~ X, dat, method = "mixed")
mod5 <- iprior(y ~ X, dat, method = "fixed", lambda = coef(mod1)[1],
              psi = coef(mod1)[2])
c(logLik(mod1), logLik(mod2), logLik(mod3), logLik(mod4),
  logLik(mod5))

## Not run:
# For large data sets, it is worth trying the Nystrom method
mod <- iprior(y ~ X, gen_smooth(5000), kernel = "se", nystrom = 50,
              est.lengthscale = TRUE)  # a bit slow
plot_fitted(mod, ci = FALSE)

## End(Not run)

---

**Description**

A convenience function to perform a k-fold cross-validation experiment and obtain mean squared error of prediction. Most of the arguments are similar to `iprior()` and `kernL()`.

**Usage**

## Default S3 method:
iprior_cv(
  y,
  ...
)
  folds = 2,
  par.cv = TRUE,
  kernel = "linear",
  method = "direct",
  control = list(),
  interactions = NULL,
  est.lambda = TRUE,
  est.hurst = FALSE,
  est.lengthscale = FALSE,
  est.offset = FALSE,
  est.psi = TRUE,
  fixed.hyp = NULL,
  lambda = 1,
psi = 1,
nystrom = FALSE,
nys.seed = NULL
)

## S3 method for class 'formula'
iprior_cv(
  formula,
  data,
  folds = 2,
  one.lam = FALSE,
  par.cv = TRUE,
  kernel = "linear",
  method = "direct",
  control = list(),
  est.lambda = TRUE,
  est.hurst = FALSE,
  est.lengthscale = FALSE,
  est.offset = FALSE,
  est.psi = TRUE,
  fixed.hyp = NULL,
  lambda = 1,
  psi = 1,
  nystrom = FALSE,
  nys.seed = NULL,
  ...
)

Arguments

y Vector of response variables

... Only used when fitting using non-formula, enter the variables (vectors or matrices) separated by commas.

folds The number of cross-validation folds. Set equal to sample size or Inf to perform leave-one-out cross-validation.

par.cv Logical. Multithreading to fit the models? Defaults to TRUE.

kernel Character vector indicating the type of kernel for the variables. Available choices are:

  - "linear" - (default) for the linear kernel
  - "canonical" - alternative name for "linear"
  - "fbm", "fbm, 0.5" - for the fBm kernel with Hurst coefficient 0.5 (default)
  - "se", "se, 1" - for the SE kernel with lengthscale 1 (default)
  - "poly", "poly2", "poly2, 0" - for the polynomial kernel of degree 2 with offset 0 (default)
  - "pearson" - for the Pearson kernel
The kernel argument can also be a vector of length equal to the number of variables, therefore it is possible to specify different kernels for each variables. Note that factor type variables are assigned the Pearson kernel by default, and that non-factor types can be forced to use the Pearson kernel (not recommended).

**method**

The estimation method. One of:

- "direct" - for the direct minimisation of the marginal deviance using optim()'s L-BFGS method
- "em" - for the EM algorithm
- "mixed" - combination of the direct and EM methods
- "fixed" - for just obtaining the posterior regression function with fixed hyperparameters (default method when setting fixed.hyp = TRUE)
- "canonical" - an efficient estimation method which takes advantage of the structure of the linear kernel

**control**

(Optional) A list of control options for the estimation procedure:

- **maxit** The maximum number of iterations for the quasi-Newton optimisation or the EM algorithm. Defaults to 100.
- **em.maxit** For method = "mixed", the number of EM steps before switching to direct optimisation. Defaults to 5.
- **stop.crit** The stopping criterion for the EM and L-BFGS algorithm, which is the difference in successive log-likelihood values. Defaults to 1e-8.
- **theta0** The initial values for the hyperparameters. Defaults to random starting values.
- **report** The interval of reporting for the optim() function.
- **restarts** The number of random restarts to perform. Defaults to 0. It's also possible to set it to TRUE, in which case the number of random restarts is set to the total number of available cores.
- **no.cores** The number of cores in which to do random restarts. Defaults to the total number of available cores.
- **omega** The overrelaxation parameter for the EM algorithm - a value between 0 and 1.

**interactions**

Character vector to specify the interaction terms. When using formulas, this is specified automatically, so is not required. Syntax is "a:b" to indicate variable a interacts with variable b.

**est.lambda** Logical. Estimate the scale parameters? Defaults to TRUE.

**est.hurst** Logical. Estimate the Hurst coefficients for fBm kernels? Defaults to FALSE.

**est.lengthscale** Logical. Estimate the lengthscales for SE kernels? Defaults to FALSE.

**est.offset** Logical. Estimate the offsets for polynomial kernels? Defaults to FALSE.

**est.psi** Logical. Estimate the error precision? Defaults to TRUE.

**fixed.hyp** Logical. If TRUE, then no hyperparameters are estimated, i.e. all of the above est.x are set to FALSE, and vice versa. If NULL (default) then all of the est.x defaults are respected.

**lambda** Initial/Default scale parameters. Relevant especially if est.lambda = FALSE.
psi
Initial/Default value for error precision. Relevant especially if est.psi = FALSE.

nystrom
Either logical or an integer indicating the number of Nystrom samples to take. Defaults to FALSE. If TRUE, then approximately 10% of the sample size is used for the Nystrom approximation.

nys.seed
The random seed for the Nystrom sampling. Defaults to NULL, which means the random seed is not fixed.

formula
The formula to fit when using formula interface.

data
Data frame containing variables when using formula interface.

one.lam
Logical. When using formula input, this is a convenient way of letting the function know to treat all variables as a single variable (i.e. shared scale parameter). Defaults to FALSE.

Details
Uses a multicore loop to fit the folds by default, set par.cv = FALSE to not use multithreading.

Value
An iprior_xv object containing a data frame of the cross-validated values such as the log-likelihood, training MSE and test MSE.

Examples
## Not run:
# 5-fold CV experiment
(mod.cv <- iprior_cv(y ~ X, gen_smooth(100), kernel = "se", folds = 5))

# LOOCV experiment
(mod.cv <- iprior_cv(y ~ X, gen_smooth(100), kernel = "se", folds = Inf))

# Can also get root MSE
print(mod.cv, "RMSE")

## End(Not run)

is.iprior_x
Test iprior objects

Description
Test whether an object is an ipriorMod, ipriorKernel, or either object with Nystrom method enabled.
is.kern_x

Usage

is.ipriorMod(x)

is.ipriorKernel(x)

is.nystrom(x)

Arguments

x An ipriorMod or ipriorKernel object.

Value

Logical.

Description

Test whether an object uses a specific type of kernel.

Usage

is.kern_linear(x)

is.kern_canonical(x)

is.kern_fbm(x)

is.kern_pearson(x)

is.kern_se(x)

is.kern_poly(x)

Arguments

x An ipriorMod object, ipriorKernel object, a kernel matrix generated from one of the kern_x() functions, or even simply just a character vector.

Value

Logical.
Reproducing kernels for the I-prior package

Description

The kernel functions used in this package are:

- The (canonical) linear kernel
- The fractional Brownian motion (fBm) kernel with Hurst index $\gamma$
- The Pearson kernel
- The (scaled) $d$-degree polynomial kernel with offset $c$
- The squared exponential (SE) kernel with lengthscale $l$

Usage

kern_canonical(x, y = NULL, centre = TRUE)
kern_linear(x, y = NULL, centre = TRUE)
kern_pearson(x, y = NULL)
kern_fbm(x, y = NULL, gamma = 0.5, centre = TRUE)
kern_se(x, y = NULL, l = 1, centre = FALSE)
kern_poly(x, y = NULL, c = 0, d = 2, lam.poly = 1, centre = TRUE)

Arguments

- **x** A vector, matrix or data frame.
- **y** (Optional) vector, matrix or data frame. $x$ and $y$ must have identical column sizes.
- **centre** Logical. Whether to centre the data (default) or not.
- **gamma** The Hurst coefficient for the fBm kernel.
- **l** The lengthscale for the SE kernel.
- **c** The offset for the polynomial kernel. This is a value greater than zero.
- **d** The degree for the polynomial kernel. This is an integer value greater than or equal to two.
- **lam.poly** The scale parameter for the polynomial kernel.
Details
The Pearson kernel is used for nominal-type variables, and thus factor-type variables are treated with the Pearson kernel automatically when fitting I-prior models. The other kernels are for continuous variables, and each emits different properties of functions.

The linear kernel is used for "straight-line" functions. In addition, if squared, cubic, or higher order terms are to be modelled, then the polynomial kernel is suitable for this purpose. For smoothing models, the fBm kernel is preferred, although the SE kernel may be used as well.

Value
A matrix whose \([i, j]\) entries are given by \(h(x[i], y[j])\), with \(h\) being the appropriate kernel function. The matrix has dimensions \(m\) by \(n\) according to the lengths of \(y\) and \(x\) respectively. When a single argument \(x\) is supplied, then \(y\) is taken to be equal to \(x\), and a symmetric \(n\) by \(n\) matrix is returned.

The matrix has a "kernel" attribute indicating which type of kernel function was called.

References
https://phd.haziqj.ml/intro/

Examples
```r
kern_linear(1:3)
kern_fbm(1:5, 1:3, gamma = 0.7)
```

---

**kernL**

Load the kernel matrices for I-prior models

Description
Load the kernel matrices for I-prior models

Usage
```r
kernL(
  y,
  ...,  
  kernel = "linear",
  interactions = NULL,
  est.lambda = TRUE,
  est.hurst = FALSE,
  est.lengthscale = FALSE,
  est.offset = FALSE,
  est.psi = TRUE,
  fixed.hyp = NULL,
  lambda = 1,
)```
psi = 1,
nystrom = FALSE,
nys.seed = NULL,
model = list(),
train.samp,
test.samp,
intercept
)

## S3 method for class 'formula'
kernL(
  formula,
  data,
  kernel = "linear",
one.lam = FALSE,
est.lambda = TRUE,
est.hurst = FALSE,
est.lengthscale = FALSE,
est.offset = FALSE,
est.psi = TRUE,
fixed.hyp = NULL,
lambda = 1,
psi = 1,
nystrom = FALSE,
nys.seed = NULL,
model = list(),
train.samp,
test.samp,
intercept,
...
)

Arguments

y Vector of response variables

... Only used when fitting using non-formula, enter the variables (vectors or matrices) separated by commas.

kernel Character vector indicating the type of kernel for the variables. Available choices are:

- "linear" - (default) for the linear kernel
- "canonical" - alternative name for "linear"
- "fbm", "fbm, 0.5" - for the fBm kernel with Hurst coefficient 0.5 (default)
- "se", "se, 1" - for the SE kernel with lengthscale 1 (default)
- "poly", "poly2", "poly2, 0" - for the polynomial kernel of degree 2 with offset 0 (default)
- "pearson" - for the Pearson kernel
The kernel argument can also be a vector of length equal to the number of variables, therefore it is possible to specify different kernels for each variables. Note that factor type variables are assigned the Pearson kernel by default, and that non-factor types can be forced to use the Pearson kernel (not recommended).

**interactions**

Character vector to specify the interaction terms. When using formulas, this is specified automatically, so is not required. Syntax is "a:b" to indicate variable a interacts with variable b.

**est.lambda**

Logical. Estimate the scale parameters? Defaults to TRUE.

**est.hurst**

Logical. Estimate the Hurst coefficients for fBm kernels? Defaults to FALSE.

**est.lengthscale**

Logical. Estimate the lengthscales for SE kernels? Defaults to FALSE.

**est.offset**

Logical. Estimate the offsets for polynomial kernels? Defaults to FALSE.

**est.psi**

Logical. Estimate the error precision? Defaults to TRUE.

**fixed.hyp**

Logical. If TRUE, then no hyperparameters are estimated, i.e. all of the above est.x are set to FALSE, and vice versa. If NULL (default) then all of the est.x defaults are respected.

**lambda**

Initial/Default scale parameters. Relevant especially if est.lambda = FALSE.

**psi**

Initial/Default value for error precision. Relevant especially if est.psi = FALSE.

**nystrom**

Either logical or an integer indicating the number of Nystrom samples to take. Defaults to FALSE. If TRUE, then approximately 10% of the sample size is used for the Nystrom approximation.

**nys.seed**

The random seed for the Nystrom sampling. Defaults to NULL, which means the random seed is not fixed.

**model**

DEPRECATED.

**train.samp**

(Optional) A vector indicating which of the data points should be used for training, and the remaining used for testing.

**test.samp**

(Optional) Similar to train.samp, but on test samples instead.

**intercept**

(Optional) Intercept for response variables.

**formula**

The formula to fit when using formula interface.

**data**

Data frame containing variables when using formula interface.

**one.lam**

Logical. When using formula input, this is a convenient way of letting the function know to treat all variables as a single variable (i.e. shared scale parameter). Defaults to FALSE.

**Value**

An ipriorKernel object which contains the relevant material to be passed to the iprior function for model fitting.

**See Also**

iprior
logLik.ipriorMod

Examples

str(ToothGrowth)
(mod <- kernL(y = ToothGrowth$len,
          supp = ToothGrowth$supp,
          dose = ToothGrowth$dose,
          interactions = "1:2"))
kernL(len ~ supp * dose, data = ToothGrowth) # equivalent formula call

# Choosing different kernels
str(stackloss)
kernL(stack.loss ~ ., stackloss, kernel = "fbm") # all fBm kernels
kernL(stack.loss ~ ., stackloss, kernel = "FBm") # cApS dOn't MatTeR
kernL(stack.loss ~ ., stackloss,
       kernel = c("linear", "se", "poly3")) # different kernels

# Sometimes the print output is too long, can use str() options here
print(mod, strict.width = "cut", width = 30)

logLik.ipriorMod Obtain the log-likelihood and deviance of an I-prior model

Description

This function calculates the log-likelihood value or deviance (twice the negative log-likelihood) for I-prior models. It works for both ipriorMod and ipriorKernel class objects.

Usage

## S3 method for class 'ipriorMod'
logLik(object, theta = NULL, ...)

## S3 method for class 'ipriorMod'
deviance(object, theta = NULL, ...)

## S3 method for class 'ipriorKernel'
logLik(object, theta = NULL, ...)

## S3 method for class 'ipriorKernel'
deviance(object, theta = NULL, ...)

Arguments

object An object of class ipriorMod or ipriorKernel.
theta (Optional) Evaluates the log-likelihood at theta.
... Not used.
Details

For ipriorKernel objects, the log-likelihood or deviance is calculated at the default parameter values: scale parameters and error precision are equal to one, while hyperparameters of the kernels (e.g. Hurst index, lengthscale, etc.) are the default values (see here for details) or ones that has been specified. For ipriorMod objects, the log-likelihood or deviance is calculated at the last obtained value from the estimation method.

For both types of objects, it is possible to supply parameter values at which to calculate the log-likelihood/deviance. This makes estimating an I-prior model more flexible, by first loading the variables into an ipriorKernel object, and then using an optimiser such as optim. Parameters have been transformed so that they can be optimised unconstrained.

See Also
check_theta.

plot.ipriorMod  
Plots for I-prior models

Description

There are three types of plots that are currently written in the package:

plot_fitted  Plot the fitted regression line with credibility bands.
plot_predict  Plot residuals against fitted values.
plot_iter  Plot the progression of the log-likelihood value over time.

The S3 method plot for class ipriorMod currently returns plot_fitted.

Usage

## S3 method for class 'ipriorMod'
plot(x, ...)
plot_resid(x)

plot_fitted_multilevel(
  x,
  X.var = 1,
  grp.var = 1,
  facet = c(2, 3),
  cred.bands = TRUE,
  show.legend = TRUE,
  show.points = TRUE,
  x.lab = NULL,
  y.lab = NULL,
  grp.lab = NULL,
plot.ipriorMod = FALSE
)

plot_fitted(x, X.var = 1, cred.bands = TRUE, size = 1, linetype = "solid")

plot_iter(x, niter.plot = NULL, lab.pos = c("up", "down"))

plot_ppc(x, draws = 100)

Arguments

x An ipriorMod object.
...
X.var The index of the X variable to plot.
grp.var Index of the grouping variable for multilevel plots.
facet The index of the X variable in which to facet. This is a vector of maximum length 2.
cred.bands Logical. Plot the confidence intervals? Defaults to TRUE.
show.legend Logical. Show legend?
show.points Logical. Show data points?
x.lab (Optional) X axis label.
y.lab (Optional) Y axis label.
grp.lab (Optional) The name for the groups, which is also the legend title.
extrapolate Logical. Extend the fitted regression line to fill the plot?
size Size of the fitted line
linetype Type of the fitted line
niter.plot (Optional) Vector of length at most two, indicating the start and end points of the iterations to plot.
lab.pos Adjust the position of the log-likelihood label.
draws Number of draws for posterior predictive check.
grp The index of the groups.

Details

See ggplot2 documentation for the plotting parameters.
pollution

Air pollution and mortality

Description

Data on the relation between weather, socioeconomic, and air pollution variables and mortality rates in 60 Standard Metropolitan Statistical Areas (SMSAs) of the USA, for the years 1959-1961.

Usage

pollution

Format

A data frame of 16 observations on 16 variables.

- Mortality: Total age-adjusted mortality rate per 100,000.
- Rain: Mean annual precipitation in inches.
- Humid: Mean annual precipitation in inches.
- JanTemp: Mean annual precipitation in inches.
- JulTemp: Mean annual precipitation in inches.
- Over65: Mean annual precipitation in inches.
- Popn: Mean annual precipitation in inches.
- Educ: Mean annual precipitation in inches.
- Hous: Mean annual precipitation in inches.
- Dens: Mean annual precipitation in inches.
- NonW: Mean annual precipitation in inches.
- WhiteCol: Mean annual precipitation in inches.
- Poor: Mean annual precipitation in inches.
- HC: Mean annual precipitation in inches.
- NOx: Mean annual precipitation in inches.
- SO2: Mean annual precipitation in inches.

References


Examples

data(pollution)
str(pollution)
predict

Obtain predicted values from ipriorMod objects

Description

Obtain predicted values from ipriorMod objects

Usage

```r
## S3 method for class 'ipriorMod'
fitted(object, intervals = FALSE, alpha = 0.05, ...)

## S3 method for class 'ipriorMod'
predict(
  object,
  newdata = list(),
  y.test = NULL,
  intervals = FALSE,
  alpha = 0.05,
  ...
)
```

```r
## S3 method for class 'ipriorPredict'
print(x, rows = 10, dp = 3, ...)
```

Arguments

- `object, x` An ipriorMod object.
- `intervals` Logical. Calculate the credibility intervals for the fitted values. Defaults to FALSE.
- `alpha` The significance level for the credibility intervals. This is a number between 0 and 1.
- `...` Not used.
- `newdata` Either a data frame when using formula method, or a list of vectors/matrices if using default method. Either way, the new data must be structurally similar to the original data used to fit the model.
- `y.test` (Optional) Test data, in order to compute test error rates.
- `rows` (Optional) The number of values/rows to display.
- `dp` (Optional) Decimal places for the values.

Value

A list of class ipriorPredict containing the fitted values, residuals (observed minus fitted), the training mean squared error, and the lower and upper intervals (if called).
Examples

dat <- gen_smooth(20)
mod <- iprior(y ~ ., dat, kernel = "se")
fitted(mod)
fitted(mod, intervals = TRUE)
predict(mod, gen_smooth(5))

with(dat, mod <<- iprior(y, X, kernel = "poly"))
newdat <- gen_smooth(30)
mod.pred <- predict(mod, list(newdat$X), y.test = newdat$y, intervals = TRUE)
str(mod.pred)
print(mod.pred, row = 5)

.sigma

Obtain the standard deviation of the residuals 'sigma'

Description

Extract the standard deviation of the residuals. For I-prior models, this is \( \sigma = 1 / \sqrt{\psi} \).

Usage

```r
## S3 method for class 'ipriorMod'
sigma(object, ...)
```

Arguments

- `object`: An object of class `ipriorMod`.
- `...`: Not used.

summary.ipriorMod

Print and summary method for I-prior models

Description

Print and summary method for I-prior models

Usage

```r
## S3 method for class 'ipriorMod'
print(x, digits = 5, ...)

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

digits  Number of decimal places for the printed coefficients.
...      Not used.
object, x  An ipriorMod object.

Description

Results of I-prior cross-validation experiment on Tecator data set

Usage

tecator.cv

Format

Results from iprior_cv cross validation experiment. This is a list of seven, with each component bearing the results for the linear, quadratic, cubic, fBm-0.5, fBm-MLE and SE I-prior models. The seventh is a summarised table of the results.

Details

For the fBm and SE kernels, it seems numerical issues arise when using a direct optimisation approach. Terminating the algorithm early (say using a relaxed stopping criterion) seems to help.

Examples

# Results from the six experiments
print(tecator.cv[[1]], "RMSE")
print(tecator.cv[[2]], "RMSE")
print(tecator.cv[[3]], "RMSE")
print(tecator.cv[[4]], "RMSE")
print(tecator.cv[[5]], "RMSE")
print(tecator.cv[[6]], "RMSE")

# Summary of results
print(tecator.cv[[7]])

## Not run:

# Prepare data set
data(tecator, package = "caret")
endpoints <- as.data.frame(endpoints)
colnames(endpoints) <- c("water", "fat", "protein")
absorp <- -t(diff(t(absorp))) # this takes first differences using diff()
fat <- endpoints$fat
# Here is the code to replicate the results

```r
mod1.cv <- iprior_cv(fat, absorp, folds = Inf)
mod2.cv <- iprior_cv(fat, absorp, folds = Inf, kernel = "poly2",
                    est.offset = TRUE)
mod3.cv <- iprior_cv(fat, absorp, folds = Inf, kernel = "poly3",
                    est.offset = TRUE)
mod4.cv <- iprior_cv(fat, absorp, method = "em", folds = Inf, kernel = "fbm",
                    control = list(stop.crit = 1e-2))
mod5.cv <- iprior_cv(fat, absorp, folds = Inf, kernel = "fbm",
                    est.hurst = TRUE, control = list(stop.crit = 1e-2))
mod6.cv <- iprior_cv(fat, absorp, folds = Inf, kernel = "se",
                    est.lengthscale = TRUE, control = list(stop.crit = 1e-2))

tecator_res_cv <- function(mod) {
  res <- as.numeric(apply(mod$res[, -1], 2, mean)) # Calculate RMSE
  c("Training RMSE" = res[1], "Test RMSE" = res[2])
}

tecator_tab_cv <- function() {
  tab <- t(sapply(list(mod1.cv, mod2.cv, mod3.cv, mod4.cv, mod5.cv, mod6.cv),
                  tecator_res_cv))
  rownames(tab) <- c("Linear", "Quadratic", "Cubic", "fbm-0.5", "fbm-MLE",
                     "SE-MLE")
  tab
}

tecator.cv <- list(
  "linear" = mod1.cv,
  "quadratic" = mod2.cv,
  "cubic" = mod3.cv,
  "fbm-0.5" = mod4.cv,
  "fbm-MLE" = mod5.cv,
  "SE" = mod6.cv,
  "summary" = tecator_tab_cv()
)
```

## End(Not run)
Usage

```r
## S3 method for class 'ipriorMod'
update(object, method = NULL, control = list(), iter.update = 100, ...)
```

Arguments

- `object`: An `ipriorMod` object.
- `method`: An optional method. See [here](#) for details.
- `control`: An optional list of controls for the estimation procedure. See [here](#) for details.
- `iter.update`: The number of additional iterations to update the I-prior model.
- `...`: Not used.
Index

* datasets
  hsb, 10
  hsbsmall, 11
  pollution, 28
  tecator.cv, 31

Accessors, 2
as.time, 4

check_theta, 5, 15, 26
decimal_place, 5
deviance.ipriorKernel
  (logLik.ipriorMod), 25
deviance.ipriorMod (logLik.ipriorMod),
  25
eigenCpp, 6
factor, 10, 22
fastSquare, 6
fastVDiag, 7
fitted.ipriorMod (predict), 29
gen_multilevel, 7
gen_smooth, 8
get_convergence (Accessors), 2
get_degree (Accessors), 2
get_estl (Accessors), 2
get_hurst (Accessors), 2
get_hyp (Accessors), 2
get_intercept (Accessors), 2
get_kern_matrix (Accessors), 2
get_kernels (Accessors), 2
get_lambda (Accessors), 2
get_lengthscale (Accessors), 2
get_method (Accessors), 2
get_niter (Accessors), 2
get_offset (Accessors), 2
get_prederror (Accessors), 2
get_psi (Accessors), 2
get_se (Accessors), 2
get_size (Accessors), 2
get_theta (Accessors), 2
get_time (Accessors), 2
get_y (Accessors), 2
ipriorColPal (gg_colour_hue), 9
gg_color_hue (gg_colour_hue), 9
gg_colour_hue, 9
ggColPal (gg_colour_hue), 9

here, 15, 26, 33
hsb, 10
hsbsmall, 11

iprior, 11, 24
iprior_cv, 16
ipriorColPal (gg_colour_hue), 9
is.iprior_x, 19
is.ipriorKernel (is.iprior_x), 19
is.ipriorMod (is.iprior_x), 19
is.kernCanonical (is.kern_x), 20
is.kern_fbm (is.kern_x), 20
is.kern_linear (is.kern_x), 20
is.kern_pearson (is.kern_x), 20
is.kern_poly (is.kern_x), 20
is.kern_se (is.kern_x), 20
is.kern_x, 20
is.nystrom (is.iprior_x), 19

kern_canonical (kernel), 21
kern_fbm (kernel), 21
kern_linear (kernel), 21
kern_pearson (kernel), 21
kern_poly (kernel), 21
kern_se (kernel), 21
kernel, 21
kernels (kernel), 21
kernL, 22
logLik.ipriorKernel (logLik.ipriorMod), 25
logLik.ipriorMod, 25

optim, 15, 26
plot.ipriorMod, 26
plot_fitted (plot.ipriorMod), 26
plot_fitted_multilevel (plot.ipriorMod), 26
plot_iter (plot.ipriorMod), 26
plot_ppc (plot.ipriorMod), 26
plot_resid (plot.ipriorMod), 26
pollution, 28
predict, 29
print.ipriorMod (summary.ipriorMod), 30
print.ipriorPredict (predict), 29

sigma, 30
summary.ipriorMod, 30

tecator.cv, 31

update, 15
update.ipriorMod, 32