Package ‘pendensity’

April 8, 2019

Type Package
Title Density Estimation with a Penalized Mixture Approach
Version 0.2.13
Date 2019-04-07
Depends R (>= 2.15.1), lattice, fda
Author Christian Schellhase
Maintainer Christian Schellhase <christian.schellhase@gmx.net>
Description Estimation of univariate (conditional) densities using penalized B-splines with automatic selection of optimal smoothing parameter.
License GPL (>= 2)
LazyLoad yes
NeedsCompilation no
Repository CRAN
Date/Publication 2019-04-07 22:10:10 UTC

R topics documented:

pendensity-package .......................................................... 2
Allianz ................................................................. 3
bias.par .............................................................. 4
ck ................................................................. 5
D.m ................................................................. 6
Derv1 .............................................................. 7
Derv2 .............................................................. 8
DeutscheBank .......................................................... 9
distr.func ........................................................... 10
dpendensity .......................................................... 11
f.hat ............................................................... 12
L.mat ............................................................... 13
marg.likelihood ....................................................... 13
my.AIC ............................................................ 14
my.bspline ........................................................ 15
The package 'pendensity' offers routines for estimating penalized unconditional and conditional (on factor groups) densities.

Description

The package 'pendensity' offers routines for estimating penalized unconditional and conditional (on factor groups) densities. For details see the description of the function pendensity.

Details

Package: pendensity
Type: Package
Version: 0.2.12
Date: 2019-04-07
License: GPL (>= 2) LazyLoad: yes

The package contributes the function 'pendensity()' for estimating densities using penalized splines techniques.

Author(s)

Christian Schellhase <christian.schellhase@gmx.net>

References


Examples

```r
y <- rnorm(100)
test <- pendensity(y~1)
```
Allianz

plot(test)

# second simple example
# with covariate

x <- rep(c(0.1),200)
y <- rnorm(400,x*0.2,1)
test <- pendensity(y~as.factor(x),lambda0=2e+07)
plot(test)

# calculate the value at some (maybe not observed) value yi=c(0,1) of the estimated density
plot(test,val=c(0,1))

# density-example of the stock exchange Allianz in 2006

data(Allianz)
time.Allianz <- strptime(Allianz[,1],form="%d.%m.%y")

# looking for all dates in 2006
data.Allianz <- Allianz[which(time.Allianz$year==2006),2]

# building differences of first order
d.Allianz <- diff(data.Allianz)

# estimating the density, choosing a special start value for lambda
density.Allianz <- pendensity(d.Allianz~1,lambda0=90000)
plot(density.Allianz)

Allianz

Daily final prices (DAX) of the German stock Allianz in the years 2006 and 2007

Description

Containing the daily final prices of the German stock Allianz in the years 2006 and 2007.

Usage

data(Allianz)
**Format**

A data frame with 507 observations of the following 2 variables.

- **Date** Date
- **ClosingPrice** ClosingPrice

**Examples**

```r
data(Allianz)
form <- '%d. %m. %y'

time.Allianz <- strftime(Allianz[,1], form)

# looking for all dates in 2006
data.Allianz <- Allianz[which(time.Allianz$year == 2006),]

# building differences of first order
d.Allianz <- diff(data.Allianz)

# estimating the density
density.Allianz <- pendensity(d.Allianz ~ 1)
plot(density.Allianz)
```

---

**Bias.par**  
*Calculating the bias of the parameter beta*

**Description**

Calculating the bias of the parameter beta.

**Usage**

```r
bias.par(penden.env)
```

**Arguments**

- **penden.env**  
  Containing all information, environment of pendensity()

**Details**

The bias of the parameter beta is calculated as the product of the penalty parameter lambda, the penalized second order derivative of the log likelihood function w.r.t. beta 'Derv.pen', the penalty matrix 'Dm' and the parameter set 'beta'.

\[
Bias(\beta) = -\lambda Derv2.pen(\beta)^{-1} Dm \beta
\]

The needed values are saved in the environment.
Calculating the actual weights $ck$

Description

Calculating the actual weights $ck$ for each factor combination of the covariates combinations.

Usage

$ck(\text{penden.env}, \text{beta.val})$

Arguments

- `penden.env`: Containing all information, environment of `pendensity()`
- `beta.val`: actual parameter beta

Details

The weights in depending of the covariate 'x' are calculated as follows.

$$ck(x, \beta) = \frac{\exp(Z(x)\beta_k)}{\sum_{j=-K}^{K} \exp(Z(x)\beta_j)}$$

For estimations without covariates, $Z$ doesn’t appear in calculations.

$$ck(\beta) = \frac{\exp(\beta_k)}{\sum_{k=-K}^{K} \exp(\beta_k)}$$

Starting density calculation, the groupings of the covariates are indexed in the main program. The groupings are saved in 'x.factor', the index which response belongs to which group is saved in 'Z.index'. Therefore, one can link to the rows in 'x.factor' to calculate the weights 'ck'.

The needed values are saved in the environment.

Value

Returning the actual weights $ck$, depending on the actual parameter beta in a matrix with rows.
Author(s)
Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References

---

### D.m

**Calculating the penalty matrix**

**Description**

Calculating the penalty matrix depending on the number of covariates \( p \), the order of differences to be penalized \( m \), the corresponding difference matrix \( L \) of order \( m \), the covariate matrix \( Z \), the number of observations \( n \) and the number of knots \( K \).

**Usage**

\[ D.m(penden.env) \]

**Arguments**

- **penden.env**: Containing all information, environment of `pendensity()`

**Details**

The penalty matrix is calculated as
\[
D_m = (L^T \otimes I_p)(I_{K-m} \otimes \frac{Z^T Z}{n})(L \otimes I_p)
\]

The needed values are saved in the environment.

**Value**

Returning the penalty matrix.

**Author(s)**
Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

**References**
Calculating the first derivative of the pendensity likelihood function w.r.t. parameter beta

**Description**
Calculating the first derivative of the pendensity likelihood function w.r.t. parameter beta.

**Usage**
```
Derv1(penden.env)
```

**Arguments**
- `penden.env` Containing all information, environment of pendensity()

**Details**
We calculate the first derivative of the pendensity likelihood function w.r.t. the parameter beta. The calculation of the first derivative of the pendensity likelihood function w.r.t. parameter beta is done in four steps. The first derivative equals in the case of covariates

\[
s(\beta) = \frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{n} s_i(\beta)
\]

where

\[
s_i(\beta) = Z^T(x_i)C^T(x_i, \beta) \frac{\hat{\phi}_i}{f(y_i|x_i)}
\]

Without covariates, the matrix 'Z' doesn’t appear. Starting density calculation, the groupings of the covariates are indexed in the main program. The groupings are saved in 'x.factor'. Creating an index that reports which response belongs to which covariate group, saving in 'Z.index'. Therefore, one can link to the rows in the object 'ck' to calculate the matrix 'C.bold', which depends only on the grouping of the covariate. Without any covariate, 'C.bold' is equal for every observation.

The calculation of the first derivative of the pendensity likelihood function w.r.t. parameter beta is done in four steps. Firstly, we calculate the matrix 'C.bold', depending on the groups of 'x.factor'. Secondly, for calculating we need the fitted values of each observation, 'f.hat'. These values are calculated for the actual parameter set beta in the program 'f.hat'. Of course, we need the value of the base for each observation, \( \phi[i] \).

Moreover, for the case of conditional density estimation, we need a Z-Matrix, due to the rules for derivations of the function 'exp()'. This Z-matrix doesn’t appear directly in the calculations. We construct the multiplication with this Z-matrix with using an outer product between the corresponding grouping in 'x.factor' and the product of the corresponding values 'C.bold' and 'base.den', divided by the fitted value 'f.hat'. Finally, we add some penalty on the derivative, which is calculated in the fourth step. The penalty equals \( -\lambda D_m \beta \).

For later use, we save the unpenalized first derivative as a matrix, in which the i-th column contains the first derivative of the pendensity likelihood function, evaluated for the i-th value of the response. The needed values are saved in the environment.
Value

- **Derv1.cal** matrix, in which the i-th column contains the first derivative, evaluated for the i-th value of the response variable without penalty. Needed for calculating the second order derivative, called \( s(\beta) \)
- **Derv1.pen** first order derivation of the penalized likelihood function w.r.t. parameter beta, called \( s_p(\beta) \)
- **f.hat.val** fitted values of the response for actual parameter beta, called \( \hat{f} \)

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References


Description

Calculating the second order derivative of the likelihood function of the pendensity approach w.r.t. the parameter beta. Thereby, for later use, the program returns the second order derivative with and without the penalty.

Usage

```r
Derv2(penden.env, lambda0)
```

Arguments

- **penden.env** Containing all information, environment of pendensity()
- **lambda0** smoothing parameter lambda

Details

We approximate the second order derivative in this approach with the negative fisher information.

\[
J(\beta) = - \frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} \approx \sum_{i=1}^{n} s_i(\beta) s_i^T(\beta).
\]

Therefore we construct the second order derivative of the i-th observation w.r.t. beta with the outer product of the matrix Derv1.cal and the i-th row of the matrix Derv1.cal.

The penalty is computed as

\[ \lambda D_m \]
DeutscheBank

Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derv2.pen</td>
<td>second order derivative w.r.t. beta with penalty</td>
</tr>
<tr>
<td>Derv2.cal</td>
<td>second order derivative w.r.t. beta without penalty. Needed for calculating of e.g. AIC.</td>
</tr>
</tbody>
</table>

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References


DeutscheBank  Daily final prices (DAX) of the German stock Deutsche Bank in the years 2006 and 2007

Description

Containing the daily final prices of the German stock Deutsche Bank in the years 2006 and 2007.

Usage

data(DeutscheBank)

Format

A data frame with 507 observations of the following 2 variables.

Date  Date
ClosingPrice  ClosingPrice

Examples

data(DeutscheBank)
form<-,'%d.%m.%y'
time.DeutscheBank <- strptime(DeutscheBank[,1],form)

#looking for all dates in 2006
data.DeutscheBank <- DeutscheBank[which(time.DeutscheBank$year==2006),2]

#building differences of first order
d.DeutscheBank <- diff(data.DeutscheBank)

deDensity.DeutscheBank <- pendensity(d.DeutscheBank~1)
plot(density.DeutscheBank)
**distr.func**

These functions are used for calculating the empirical and theoretical distribution functions.

### Description

These functions cooperate with each other for calculating the distribution functions. 'distr.func' is the main program, calling 'distr.func.help', generating an environment with needed values for calculating the distribution of each interval between two neighbouring knots. 'distr.func' returns analytical functions of the distribution of each interval between two neighbouring knots. Therefore the function 'poly.part' is needed to construct these functions. 'cal.int' evaluates these integrals, considering if the whole interval should be evaluated or if any discrete value 'yi' is of interest.

### Usage

```r
distr.func(yi = NULL, obj, help.env=distr.func.help(obj))
distr.func.help(obj)
cal.int(len.b, q, help.env, knots.val)
poly.part(i,j,knots.val,help.env,q, yi=NULL, poly=FALSE)
```

### Arguments

- **yi**
  - if the distribution at any discrete point is of interest, you can call for it. Default=NULL doesn’t consider any discrete point
- **obj**
  - a object of class pendensity
- **help.env**
  - object is generated with calling distr.func.help(obj)
- **len.b**
  - length of B-Spline
- **q**
  - order of the B-Spline
- **knots.val**
  - values of the used knots
- **poly**
  - TRUE/FALSE
- **i**
  - internal values for calculating the polynomials of each B-Spline
- **j**
  - internal values for calculating the polynomials of each B-Spline

### Value

- **distr.func**
  - returns analytical functions of the distributions between each two neighbouring intervals
- **distr.func.help**
  - creating environment 'help.env', creating help points between each two neighbouring knots and calculates the polynomial-coefficients of each base part
- **cal.int**
  - evaluating the result of distr.func. Thereby it's possible to call for an explicit distribution values F(yi)
- **poly.part**
  - using in 'distr.func' for creating the polynomial functions of each interval of each two neighbouring knots
**Author(s)**

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

**References**


---

**dpendensity**

*Calculating the fitted density or distribution*

**Description**

Calculating the fitted density or distribution.

**Usage**

```r
x
val
```

**Arguments**

- `x` Object of class `pendensity`
- `val` Vector of values where to calculate the density

**Author(s)**

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

**References**

Calculating the actual fitted values 'f.hat' of the estimated density function \( f \) for the response \( y \)

Description

Calculating the actual fitted values of the response, depending on the actual parameter set \( \beta \)

Usage

\[
f.hat(penden.env,ck.temp=NULL)
\]

Arguments

- \( penden.env \) Containing all information, environment of \texttt{pendensity()}
- \( ck.temp \) actual weights, depending on the actual parameter set \( \beta \). If NULL, the \( \beta \) parameter is caught in the environment

Details

Calculating the actual fitted values of the response, depending on the actual parameter set \( \beta \). Multiplying the actual set of parameters \( c_k \) with the base 'base.den' delivers the fitted values, depending on the group of covariates, listed in 'x.factor'.

Value

The returned value is a vector of the fitted value for each observation of \( y \).

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References

L.mat

Calculates the difference matrix of order m

Description
Calculating the differences matrix 'L' of order 'm', depending on the number of knots 'k'.

Usage
L.mat(penden.env)

Arguments
penden.env Environment of pendensity()

Value
Returns the difference matrix of order 'm' for given number of knots 'K'.
The needed values are saved in the environment.

Note
Right now, the difference matrix is implemented for m=1,2,3,4.

Author(s)
Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References

marg.likelihood
Calculating the marginal likelihood

Description
Calculating the marginal likelihood.

Usage
marg.likelihood(penden.env,pen.likelihood)
Arguments
penden.env  Containing all information, environment of pendensity()
pen.likelihood  penalized log likelihood

Details
Calculating is done using a Laplace approximation for the integral of the marginal likelihood.
The needed values are saved in the environment.

Value
Returning the marginal likelihood.

Author(s)
Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References

Description
Calculating the AIC value of the density estimation. Therefore, we add the unpenalized log likelihood of the estimation and the degree of freedom, which are

Usage
my.AIC(penden.env, lambda0, opt.Likelihood = NULL)

Arguments
penden.env  Containing all information, environment of pendensity()
lambda0  penalty parameter lambda
opt.Likelihood  optimal unpenalized likelihood of the density estimation

Details
AIC is calculated as $AIC(\lambda) = -l(\hat{\theta}) + df(\lambda)$
**Value**

- myAIC: sum of the negative unpenalized log likelihood and mytrace
- mytrace: calculated mytrace as the sum of the diagonal matrix df, which results as the product of the inverse of the penalized second order derivative of the log likelihood with the unpenalized second order derivative of the log likelihood

**Author(s)**

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

**References**


---

**my.bspline**

**Description**

Integrates the normal B-Spline Base to a value of one. The dimension of the base depends on the input of number of knots 'k' and of the order of the B-Spline base 'q'.

**Usage**

`my.bspline(h, q, knots.val, y, K, plot.bsp)`

**Arguments**

- **h**: if equidistant knots are just (default in pendensity()), h is the distance between two neighbouring knots
- **q**: selected order of the B-Spline base
- **knots.val**: selected values for the knots
- **y**: values of the response variable y
- **K**: the number of knots K for the construction of the base
- **plot.bsp**: Indicator variable TRUE/FALSE if the integrated B-Spline base should be plotted
Details

Firstly, the function constructs the B-Spline base to the given number of knots 'K' and the given locations of the knots 'knots.val$val'. Due to the recursive construction of the B-Spline, for all orders greater than 2, the dimension of the B-Spline base of given K grows up with help.degree=q-2. Avoiding open B-Splines at the boundary, we simulate 6 extra knots at both ends of the support, saved in knots.val$all. Therefore, we get normal B-Splines at the given knots 'knots.val$val'. For these knots, we construct the B-Spline base of order 'q' and for order 'q+1' (using for calculation the distribution). Additionally, we save q-1 knots at both ends of the support of 'knots.val$val'. After construction, we get a base of dimension K=K+help.degree. So, we define our value K and cut our B-Spline base at both ends to get the adequate base due to the order 'q' and the number of knots 'K'. For the base of order 'q+1', we need to get an additional base, due to the construction of the B-Splines. Due to the fact, that we use equidistant knots, we can integrate our B-Splines very simple to the value of 1. The integration is done by the well-known factor q/(knots.val$help[i+q]-knots.val$help[i]). This results the standardization coefficients 'stand.num' for each B-spline (which are identically for equidistant knots). Moreover, one can draw the integrated base and, if one calls this function with the argument 'plot.bsp=TRUE'.

Value

- base.den: The integrated B-Spline base of order q
- base.den2: The integrated B-Spline base of order q+1
- stand.num: The coefficients for standardization of the ordinary B-Spline base
- knots.val: This return is a list. It consider of the used knots 'knots.val$val', the help knots 'knots.val$help' and the additional knots 'knots.val$all', used for the construction of the base and the calculation of the distribution function of each B-Spline.
- K: The transformed value of K, due to used order 'q' and the input of 'K'
- help.degree: Due to the recursive construction of the B-Spline, for all orders greater than 2, the dimension of the B-Spline base of given K grows up with 'help.degree=q-2'. This value is returned for later use.

Note

This functions uses the fda-package for the construction of the B-Spline Base.

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>
Usage

```r
my.positive.definite.solve(A, eps = 1e-15)
```  
Arguments

- **A**: quadratic positive definite matrix
- **eps**: level of the lowest eigenvalue to consider

Details

The program makes an eigenvalue decomposition of the positive definite matrix A and searches all eigenvalues greater than eps. The value of return is the inverse matrix of A, constructed with the matrix product of the corresponding eigenvalues and eigenvectors.

Value

The return is the inverse matrix of A.

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

---

new.beta.val

Calculating the new parameter beta

Description

Calculating the direction of the Newton-Raphson step for the known beta and iterate a step size bisection to control the maximizing of the penalized likelihood.

Usage

```r
new.beta.val(llold, penden.env)
```  
Arguments

- **llold**: log likelihood of the algorithm one step before
- **penden.env**: Containing all information, environment of pendensity()
Details

We terminate the search for the new beta, when the new log likelihood is smaller than the old likelihood and the step size is smaller or equal 1e-3. We calculate the direction of the Newton Raphson step for the known beta_t and iterate a step size bisection to control the maximizing of the penalized likelihood

\[ l_p(\beta_t, \lambda_0) \]

. This means we set

\[ \beta_{t+1} = \beta_t - 2^{-v} \{ s_p(\beta_t, \lambda_0) \cdot (-J_p(\beta_t, \lambda_0))^{-1} \} \]

with \( s_p \) as penalized first order derivative and \( J_p \) as penalized second order derivative. We begin with \( v = 0 \). Not yielding a new maximum for a current \( v \), we increase \( v \) step by step respectively bisect the step size. We terminate the iteration, if the step size is smaller than some reference value epsilon (eps=1e-3) without yielding a new maximum. We iterate for new parameter beta until the new log likelihood depending on the new estimated parameter beta differ less than 0.1 log-likelihood points from the log likelihood estimated before.

Value

<table>
<thead>
<tr>
<th>Likeli</th>
<th>corresponding log likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>step</td>
<td>used step size</td>
</tr>
</tbody>
</table>

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References


new.lambda

Calculating new penalty parameter lambda

Description

Calculating new penalty parameter lambda.

Usage

new.lambda(penden.env,lambda0)

Arguments

| penden.env | Containing all information, environment of pendensity() |
| lambda0   | actual penalty parameter lambda                  |
Iterations for the lambda is stopped, when the changes between the old and the new lambda is smaller than 0.01*lambda0. If this criterion isn’t reached, the iteration is terminated after 11 iterations.

The iteration formulae is

\[ \lambda^{-1} = \frac{\hat{\beta}^T D_m \hat{\beta}}{df(\lambda) - (m - 1)}. \]

Value

Returning the new lambda.

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References

Details

The calculation depends on the fitted values of the response as well as on the penalty parameter lambda and the penalty matrix Dm.

\[ l(\beta) = \sum_{i=1}^{n} \left[ \log \left\{ \sum_{k=-K}^{K} c_k(x_i, \beta) \phi_k(y_i) \right\} \right] - \frac{1}{2} \lambda \beta^T D_m \beta \]

The needed values are saved in the environment.

Value

Returns the log likelihood depending on the penalty parameter lambda.

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References


---

**pendenForm**  
*Formula interpretation and data transfer*

Description

Function ‘pendenForm’ interprets the input ‘form’ of the function pendensity(), transfers the response and covariates data back to the main program and categorize the values to groupings.

Usage

```r
pendenform(pendenNenvI, stringNhelp(stringL star = " ")
```

Arguments

penden.env environment used in pendensity()
string string of the formula
star separating letter

Value

Returning the values and the structure of response and covariates.

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>
Calculating penalized density

Description

Main program for estimation penalized densities. The estimation can be done for response with or without any covariates. The covariates have to be factors. The response is called 'y', the covariates 'x'. We estimate densities using penalized splines. This done by using a number of knots and a penalty parameter, which are sufficient large. We penalize the m-order differences of the beta-coefficients to estimate the weights 'ck' of the used base functions.

Usage

pendensity(form, base, no.base, max.iter, lambda0, q, sort, with.border, m, data, eps)

Arguments

form formula describing the density, the formula is

\[ y \sim x_1 + x_2 + \ldots x_n \]

where 'x' have to be factors.

base supported bases are "bspline" or "gaussian"

no.base how many knots 'K', following the approach to use 2*no.base+1 knots, if 'no.base' is NULL, default is K=41.

max.iter maximum number of iteration, the default is max.iter=20.

lambda0 start penalty parameter, the default is lambda0=500

q order of B-Spline base, the default is 'q=3'

sort TRUE or FALSE, if TRUE the response and the covariates should be sorted. Default is TRUE.

with.border determining the number of additional knots on the left and the right of the support of the response. The number of knots 'no.base' is not influenced by this parameter. The amount of knots 'no.base' are placed on the support of the response. The amount of knots determined in 'with.border' is placed outside the support and reduce the amount of knots on the support about its value. Default is NULL.

m m-th order difference for penalization. Default is m=q.

data reference to the data. Default reference is the data=parent.frame().

eps Level of percentage to determine calculation of optimal lambda, default=0.01
Details

`pendensity()` begins with setting the parameters for the estimation. Checking the formula and transferring the data into the program, setting the knots and creating the base, depending on the chosen parameter 'base'. Moreover the penalty matrix is constructed. At the beginning of the first iteration the beta parameter are set equal to zero. With this setup, the first log likelihood is calculated and is used for the first iteration for a new beta parameter.

The iteration for a new beta parameter is done with a Newton-Raphson-Iteration and implemented in the function 'new.beta.val'. We calculate the direction of the Newton Raphson step for the known \( \beta_t \) and iterate a step size bisection to control the maximizing of the penalized likelihood

\[
l_p(\beta_t, \lambda_0)
\]

. This means we set

\[
\beta_{t+1} = \beta_t - 2^{-v}\{s_p(\beta_t, \lambda_0) \cdot (-J_p(\beta_t, \lambda_0))^{-1}\}
\]

with \( s_p \) as penalized first order derivative and \( J_p \) as penalized second order derivative. We begin with \( v = 0 \). Not yielding a new maximum for a current \( v \), we increase \( v \) step by step respectively bisect the step size. We terminate the iteration, if the step size is smaller than some reference value epsilon (eps=1e-3) without yielding a new maximum. We iterate for new parameter beta until the new log likelihood depending on the new estimated parameter beta differ less than 0.1 log-likelihood points from the log likelihood estimated before.

After reaching the new parameter beta, we iterate for a new penalty parameter lambda. This iteration is done by the function 'new.lambda’. The iteration formula is

\[
\lambda^{-1} = \frac{\hat{\beta}^T D_m \hat{\beta}}{\text{df}(\lambda) - p(m-1)}
\]

The iteration for the new lambda is terminated, if the approximate degree of freedom minus p*(m-1) is smaller than some epsilon2 (eps2=0.01). Moreover, we terminate the iteration if the new lambda is approximatively converted, i.e. the new lambda differs only 0.001*old lambda (*) from the old lambda. If these both criteria doesn’t fit, the lambda iteration is terminated after eleven iterations.

We begin a new iteration with the new lambda, restarting with parameter beta setting equal to zero again. This procedure is repeated until convergence of lambda, i.e. that the new lambda fulfills the criteria (*). If this criteria is not fulfilled after 20 iterations, the total iteration terminates.

After terminating all iterations, the final AIC, ck and beta are saved in the output.

For speediness, all values, matrices, vectors etc. are saved in an environment called 'penden.env'. Most of the used programs get only this environment as input.

Value

Returning an object of class `pendensity`. The class `pendensity` consists of the “call” and three main groups "values", "splines" and "results".

###

\$values contains: y: the values of the response variable x: the values of the covariate(s) sort: TRUE/FALSE if TRUE the response (and covariates) have been sorted in increasing order of the response
$\text{values}$ covariate contains Z: matrix Z levels: existing levels of each covariate how.levels: number of existing levels of all covariates how.combi: number of combination of levels x.factor: list of all combination of levels

###

$splines$ contains K: number of knots N: number of coefficients estimated for each base, depending on the number of covariates. MeanW: values of the knots used for splines and means of the Gaussian densities Stand.abw: values of the standard deviance of the Gaussian densities h: distance between the equidistant knots m: used order of the B-Spline base stand.num: calculated values for standardization getting B-Spline densities base.den: values of the base of order q created with knots=knots.val$val base.den2: values of the base of order q+1 created with knots=knots.val$val. Used for calculating the distribution function(s). L: used difference matrix Dm: used penalty matrix, depending on lambda0, L, (Z) and n=number of observations help.degree: additional degree(s) depending on the number of knots K and the used order q

$splines$knots.val contains: val: list of the used knots in the support of the response all: list of the used knots extended with additional knots used for constructing

###

results contains: ck: final calculated weights ck beta.val: final calculated parameter beta lambda0: final calculated lambda0 fitted: fitted values of the density f(y) variance.par: final variances of the parameter beta bias.par: final bias of the parameter beta

results$AIC contains: my.AIC: final AIC value my.trace: trace component of the final AIC

results$Derv contains: Derv2.pen: final penalized second order derivation Derv2.cal: final non-penalized second order derivation Derv1.cal: final non-penalized first order derivation

results$iterations contains: list.opt.results: list of the final results of each iteration of new beta + new lambda all.lists: list of lists. Each list contains the result of one iteration


**Author(s)**

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

**References**


---

**plot.pendensity**

Plotting estimated penalized densities

**Description**

Plotting estimated penalized densities, need object of class `pendensity`. 

---
Usage

```r
# S3 method for class 'pendensity'
plot(x, plot.val = 1, val=NULL, latt = FALSE, kernel = FALSE, confi = TRUE,
     main = NULL, sub = NULL, xlab = NULL, ylab = NULL, plot.base = FALSE,
     lwd=NULL,legend.txt=NULL,plot.dens=TRUE,...)
```

Arguments

- `x`: object of class pendensity
- `plot.val`: if plot.val=1 the density is plotted, if plot.val=2 the distribution function of the observation values is plotted, if plot.val=3 the distribution function is plotted as function
- `val`: vector of y, at which the estimated density should be calculated. If plot.val=2, the calculated values of distribution are returned and the values are pointed in the distribution function of the observed values.
- `latt`: TRUE/FALSE, if TRUE the lattice interface should be used for plotting, default=FALSE
- `kernel`: TRUE/FALSE, if TRUE a kernel density estimation should be added to the density plots, default=FALSE
- `confi`: TRUE/FALSE, if TRUE confidence intervals should be added to the density plots, default=TRUE
- `main`: Main of the density plot, if NULL main contains settings 'K', 'AIC' and 'lambda0' of the estimation
- `sub`: sub of the density plot, if NULL sub contains settings used base 'base' and used order of B-Spline 'q'
- `xlab`: xlab of the density plot, if NULL xlab contains 'y'
- `ylab`: ylab of the density plot, if NULL ylab contains 'density'
- `plot.base`: TRUE/FALSE, if TRUE the weighted base should be added to the density plot, default=FALSE
- `lwd`: lwd of the lines of density plot, if NULL lwd=3, the confidence bands are plotted with lwd=2
- `legend.txt`: if FALSE no legend is plotted, legend.txt can get a vector of characters with length of the groupings. legend.txt works only for plot.val=1
- `plot.dens`: TRUE/FALSE, if the estimated density should be plotted. Default=TRUE. Interesting for evaluating densities in values 'val', while this special plot is not needed.
- `...`: further arguments

Details

Each grouping of factors is plotted. Therefore, equidistant help values are constructed in the support of the response for each grouping of factors. Weighting these help values with knots weights ck results in the density estimation for each grouping of factors. If asked for, pointwise confidence intervals are computed and plotted.
Value

If the density function is plotted, function returns two values

help.env  Contains the constructed help values for the response, the corresponding values
           for the densities and if asked for the calculated confidence intervals
combi    list of all combinations of the covariates

If additionally the function is called with a valid argument for 'val', a list returns with

y         values at which the estimated density has been calculated
fy        calculated density values in y
sd.up.y.val the values of the upper confidence interval of y
sd.down.y.val the values of the lower confidence interval of y

If the empirical distribution function is plotted, the function returns

y         containing the observed values y
sum       containing the empirical distribution of each observation y

If the theoretical distribution function is plotted, the function returns an environment. For plotting
the theoretical distributions, each interval between two knots is evaluated at 100 equidistant simulated
points between the two knots considered. These points are saved in the environment with
the name "paste("x",i,sep="")" for each interval i, the calculated distribution is save with the name
"paste("F(x)",i,sep="")" for each interval i. For these points, the distribution is calculated.

Note

For plotting the density and e.g. the empirical distributions, use e.g. 'X11()' before calling the
second plot to open a new graphic device.

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References


Examples

```r
y <- rnorm(100)
test <- pendensity(y~1)
plot(test)
#distribution
plot(test, plot.val=2)
```
print.pendensity  

Printing the main results of the (conditional) penalized density estimation

Description
Printing the call of the estimation, the resulting weights, the final \( \lambda_0 \) and the corresponding value of AIC. Need an object of class pendensity.

Usage

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

Arguments

- `x`  
x has to be object of class pendensity
- `...`  
further arguments

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

test.equal  

Testing pairwise equality of densities

Description

Every group of factor combination is tested pairwise for equality to all other groups.

Usage

test.equal(obj)

Arguments

- `obj`  
object of class pendensity

Details

We consider the distribution of the integrated B-Spline density base. This is saved in the program in the object named `mat1`. Moreover, we use the variance `var.par` of the estimation, the weights and some matrices `C` of the two compared densities to construct the matrix `W`. We simulate the distribution of the test statistic using a spectral decomposition of `W`. 
Calculating the variance of the parameters

Description
Calculating the variance of the parameters of the estimation, depending on the second order derivative and the penalized second order derivative of the density estimation.

Usage
```
variance.par(penden.env)
```

Arguments

- `penden.env` Containing all information, environment of `pendensity()`

Details
The variance of the parameters of the estimation results as the product of the inverse of the penalized second order derivative times the second order derivative without penalization time the inverse of the penalized second order derivative.

\[
V(\beta, \lambda_0) = I_p^{-1}(\beta, \lambda)I_p(\beta, \lambda = 0)I_p^{-1}(\beta, \lambda) \quad \text{with} \quad I_p(\beta^{-1}, \lambda) = E_{f(y)}\{J_p(\beta, \lambda)\}
\]

The needed values are saved in the environment.

Value
The return is a variance matrix of the dimension (K-1)x(K-1).

Author(s)
Christian Schellhase <cschellhase@wiwi.uni-bielefeld.de>

References
Calculating variance and standard deviation of each observation.

Description

Calculating the variance and standard deviation of each observation. Therefore we use the variance of the parameter set beta, called 'var.par'.

Usage

variance.val(base.den, var.par, weight, K, x, list.len, Z, x.factor, y.val=NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base.den</td>
<td>base values</td>
</tr>
<tr>
<td>var.par</td>
<td>variance of the parameter set beta</td>
</tr>
<tr>
<td>weight</td>
<td>weights ck</td>
</tr>
<tr>
<td>K</td>
<td>number of knots</td>
</tr>
<tr>
<td>x</td>
<td>covariates</td>
</tr>
<tr>
<td>list.len</td>
<td>number of covariate combinations</td>
</tr>
<tr>
<td>Z</td>
<td>covariate matrix</td>
</tr>
<tr>
<td>x.factor</td>
<td>list of covariate combinations</td>
</tr>
<tr>
<td>y.val</td>
<td>optimal values for calculating the variance in any point yi in the case of a factorial density</td>
</tr>
</tbody>
</table>

Value

Returning a vector with the standard deviation of each observation.

Author(s)

Christian Schellhase <cschellhase@wiwi.uni-bielefeld>

References

Index

*Topic **IO**
  pendenForm, 20

*Topic **algebra**
  my.positive.definite.solve, 16

*Topic **aplot**
  plot.pendensity, 23

*Topic **datasets**
  Allianz, 3
  DeutscheBank, 9

*Topic **math**
  Derv1, 7
  Derv2, 8
  distr.func, 10
  pendensity, 11
  marg.likelihood, 13
  my.AIC, 14
  my.bspline, 15

*Topic **nonparametric**
  bias.par, 4
  ck, 5
  D.m, 6
  f.hat, 12
  L.mat, 13
  new.beta.val, 17
  new.lambda, 18
  pendensity, 21
  pendensity-package, 2
  test.equal, 26
  variance.par, 27
  variance.val, 28

*Topic **print**
  print.pendensity, 26

Allianz, 3

bias.par, 4

cal.int (distr.func), 10
ck, 5

D.m, 6
Derv1, 7
Derv2, 8
DeutscheBank, 9
distr.func, 10
dpendensity, 11
f.hat, 12
L.mat, 13
marg.likelihood, 13
my.AIC, 14
my.bspline, 15
my.positive.definite.solve, 16
new.beta.val, 17
new.lambda, 18
pen.log.like, 19
pendensity, 21
pendensity-package, 2
plot (plot.pendensity), 23
plot.pendensity, 23
poly.part (distr.func), 10
ppendensity (pendensity), 11
print (print.pendensity), 26
print.pendensity, 26
string.help (pendenForm), 20
test.equal, 26
variance.par, 27
variance.val, 28