Package 'KRLS'

May 7, 2026

Type Package

Title Kernel-Based Regularized Least Squares

Version 1.1-0

Date 2026-04-29

Description Implements Kernel-based Regularized Least Squares (KRLS), a machine learning method to fit multidimensional functions y = f(x) for regression and classification problems without relying on linearity or additivity assumptions. KRLS finds the best fitting function by minimizing the squared loss of a Tikhonov regularization problem, using Gaussian kernels as radial basis functions. For further details see Hainmueller and Hazlett (2014, <doi:10.1093/pan/mpt019>).

License GPL (>= 2)

Imports grDevices, graphics, stats

Suggests lattice, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://web.stanford.edu/~jhain/, https://github.com/j-hai/KRLS

BugReports https://github.com/j-hai/KRLS/issues

Encoding UTF-8

NeedsCompilation no

Author Jens Hainmueller [aut, cre],
Chad Hazlett [aut]

Maintainer Jens Hainmueller <jhain@stanford.edu>

Repository CRAN

Date/Publication 2026-04-30 05:10:43 UTC

Contents

`fdskrls` *Compute first differences with KRLS*

Description

Internal function that is called by `krls` to computes first differences for binary predictors in the X matrix. It would normally not be called by the user directly.

Usage

```
fdskrls(object,...)
```

Arguments

`object`	Object from call to `krls`.
`...`	additional arguments to be passed to lower level functions

Value

A object of class `krls` where the derivatives, average derivatives, and the varinaces of the average derivatives are replaced with the first differences for binary predictors. The binaryindicator is also updated and set to TRUE for binary predictors.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

See Also

`krls`

gausskernel *Gaussian Kernel Distance Computation*

Description

Given a *N* by *D* numeric data matrix, this function computes the *N* by *N* distance matrix with the pairwise distances between the rows of the data matrix as measured by a Gaussian Kernel.

Usage

```
gausskernel(X = NULL, sigma = NULL)
```

Arguments

X	*N* by *N* numeric data matrix.
sigma	Positive scalar that specifies the bandwidth of the Gaussian kernel (see details).

Details

Given two *D* dimensional vectors x_i and x_j. The Gaussian kernel is defined as

$$k(x_i, x_j) = exp(\frac{-||x_i - x_j||^2}{\sigma^2})$$

where $||x_i - x_j||$ is the Euclidean distance given by

$$||x_i - x_j|| = ((x_i 1 - x_j 1)^2 + (x_i 2 - x_j 2)^2 + ... + (x_i D - x_j D)^2)^{\cdot 5}$$

and σ^2 is the bandwidth of the kernel.

Note that the Gaussian kernel is a measure of similarity between x_i and x_j. It evalues to 1 if the x_i and x_j are identical, and approaches 0 as x_i and x_j move further apart.

The function relies on the `dist` function in the stats package for an initial estimate of the euclidean distance.

Value

An *N* by *N* numeric distance matrix that contains the pairwise distances between the rows in *X*.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

See Also

`dist` function in the stats package.

Examples

```
X <- matrix(rnorm(6),ncol=2)
gausskernel(X=X,sigma=1)
```

---

krls *Kernel-based Regularized Least Squares (KRLS)*

Description

Function implements Kernel-Based Regularized Least Squares (KRLS), a machine learning method described in Hainmueller and Hazlett (2014) that allows users to solve regression and classification problems without manual specification search and strong functional form assumptions. KRLS finds the best fitting function by minimizing a Tikhonov regularization problem with a squared loss, using Gaussian Kernels as radial basis functions. KRLS reduces misspecification bias since it learns the functional form from the data. Yet, it nevertheless allows for interpretability and inference in ways similar to ordinary regression models. In particular, KRLS provides closed-form estimates for the predicted values, variances, and the pointwise partial derivatives that characterize the marginal effects of each independent variable at each data point in the covariate space. The distribution of pointwise marginal effects can be used to examine effect heterogeneity and or interactions.

Usage

```
krls(X = NULL, y = NULL, whichkernel = "gaussian", lambda = NULL,
sigma = NULL, derivative = TRUE, binary= TRUE, vcov=TRUE,
print.level = 1,L=NULL,U=NULL,tol=NULL,eigtrunc=NULL)
```

Arguments

X	*N* by *D* data numeric matrix that contains the values of *D* predictor variables for $i = 1, \ldots, N$ observations. The matrix may not contain missing values or constants. Note that no intercept is required since the function operates on demeaned data and subtracting the mean of *y* is equivalent to including an (unpenalized) intercept into the model.
y	*N* by *1* data numeric matrix or vector that contains the values of the response variable for all observations. This vector may not contain missing values.
whichkernel	String vector that specifies which kernel should be used. Must be one of `gaussian`, `linear`, `poly1`, `poly2`, `poly3`, or `poly4` (see details). Default is `gaussian`.
lambda	A positive scalar that specifies the λ parameter for the regularizer (see details). It governs the tradeoff between model fit and complexity. By default, this parameter is chosen by minimizing the sum of the squared leave-one-out errors.
sigma	A positive scalar that specifies the bandwidth of the Gaussian kernel (see `gausskernel` for details). By default, the bandwidth is set equal to *D* (the number of dimensions) which typically yields a reasonable scaling of the distances between observations in the standardized data that is used for the fitting.
derivative	Logical that specifies whether pointwise partial derivatives should be computed. Currently, derivatives are only implemented for the Gaussian Kernel.

`binary`	Logical that specifies whether first-differences instead of pointwise partial derivatives should be computed for binary predictors. Ignored unless `derivative=TRUE`.
`vcov`	Logical that specifies whether variance-covariance matrix for the choice coefficients *c* and fitted values should be computed. Note that `derivative=TRUE` requires that `vcov=TRUE`.
`print.level`	Positive integer that determines the level of printing. Set to 0 for no printing and 2 for more printing.
`L`	Non-negative scalar that determines the lower bound of the search window for the leave-one-out optimization to find λ. Default is `NULL` which means that the lower bound is found by using an algorithm outlined in `lambdasearch`.
`U`	Positive scalar that determines the upper bound of the search window for the leave-one-out optimization to find λ. Default is `NULL` which means that the upper bound is found by using an algorithm outlined in `lambdasearch`.
`tol`	Positive scalar that determines the tolerance used in the optimization routine used to find λ. Default is `NULL` which means that convergence is achieved when the difference in the sum of squared leave-one-out errors between the *i* and the *i+1* iteration is less than *N * 10^-3*.
`eigtrunc`	Positive scalar that determines how much eignvalues should be trunacted for finding the upper bound of the search window in the algorithm outlined in `lambdasearch`. If `eigtrunc` is set to *10^-6* this means that we keep only eigenvalues that are *10^-6* as large as the first. Default is `eigtrunc=NULL` which means no truncation is used.

Details

`krls` implements the Kernel-based Regularized Least Squares (KRLS) estimator as described in Hainmueller and Hazlett (2014). Please consult this reference for any details.

Kernel-based Regularized Least Squares (KRLS) arises as a Tikhonov minimization problem with a squared loss. Assume we have data of the from y_i, x_i where *i* indexes observations, $y_i \in R$ is the outcome and $x_i \in R^D$ is a *D*-dimensional vector of predictor values. Then KRLS searches over a space of functions H and chooses the best fitting function f according to the rule:

$$argmin_{f \in H} \sum_{i}^{N} (y_i - f(x_i))^2 + \lambda ||f||_{H^2}$$

where $(y_i - f(x_i))^2$ is a loss function that computes how 'wrong' the function is at each observation *i* and $||f||_{H^2}$ is the regularizer that measures the complexity of the function according to the L_2 norm $||f||^2 = \int f(x)^2 dx$. λ is the scalar regularization parameter that governs the tradeoff between model fit and complexity. By default, λ is chosen by minimizing the sum of the squared leave-one-out errors, but it can also be specified by the user in the `lambda` argument to implement other approaches.

Under fairly general conditions, the function that minimizes the regularized loss within the hypothesis space established by the choice of a (positive semidefinite) kernel function $k(x_i, x_j)$ is of the form

$$f(x_j) = \sum_{i}^{N} c_i k(x_i, x_j)$$

where the kernel function $k(x_i, x_j)$ measures the distance between two observations x_i and x_j and c_i is the choice coefficient for each observation i. Let K be the N by N kernel matrix with all pairwise distances $K_i j = k(x_i, x_j)$ and c be the N by 1 vector of choice coefficients for all observations then in matrix notation the space is $y = Kc$.

Accordingly, the `krls` function solves the following minimization problem

$$argmin_{f \in H} \sum_{i}^{n} (y - Kc)'(y - Kc) + \lambda c' K c$$

which is convex in c and solved by $c = (K + \lambda I)^{-}1y$ where I is the identity matrix. Note that this linear solution provides a flexible fitted response surface that typically reduces misspecification bias because it can learn a wide range of nonlinear and or nonadditive functions of the predictors.

If `vcov=TRUE` is specified, `krls` also computes the variance-covariance matrix for the choice coefficients c and fitted values $y = Kc$ based on a variance estimator developed in Hainmueller and Hazlett (2014). Note that both matrices are *N* by *N* and therefore this results in increased memory and computing time.

By default, `krls` uses the Gaussian Kernel (`whichkernel = "gaussian"`) given by

$$k(x_i, x_j) = exp(\frac{-||x_i - x_j||^2}{\sigma^2})$$

where $||x_i - x_j||$ is the Euclidean distance. The kernel bandwidth σ^2 is set to D, the number of dimensions, by default, but the user can also specify other values using the `sigma` argument to implement other approaches.

If `derivative=TRUE` is specified, `krls` also computes the pointwise partial derivatives of the fitted function wrt to each predictor using the estimators developed in Hainmueller and Hazlett (2014). These can be used to examine the marginal effects of each predictor and how the marginal effects vary across the covariate space. Average derivatives are also computed with variances. Note that the `derivative=TRUE` option results in increased computing time and is only supported for the Gaussian kernel, i.e. when `whichkernel = "gaussian"`. Also `derivative=TRUE` requires that `vcov=TRUE`.

If `binary=TRUE` is also specified, the function will identify binary predictors and return first differences for these predictors instead of partial derivatives. First differences are computed going from the minimum to the maximum value of each binary predictor. Note that first differences are more appropriate to summarize the effects for binary predictors (see Hainmueller and Hazlett (2014) for details).

A few other kernels are also implemented, but derivatives are currently not supported for these: "linear": $k(x_i, x_j) = x_i' x_j$, "poly1", "poly2", "poly3", "poly4" are polynomial kernels based on $k(x_i, x_j) = (x_i' x_j + 1)^p$ where p is the order.

Value

A list object of class `krls` with the following elements:

`K`	N by N matrix of pairwise kernel distances between observations.
`coeffs`	N by 1 vector of choice coefficients c.
`Le`	scalar with sum of squared leave-one-out errors.
`fitted`	N by 1 vector of fitted values.
`X`	original N by D predictor data matrix.
`y`	original N by 1 matrix of values of the outcome variable.
`sigma`	scalar with value of bandwidth, σ^2, used for the Gaussian kernel.
`lambda`	scalar with value of regularization parameter, λ, used (user specified or based on leave-one-out cross-validation).
`R2`	scalar with value of R-squared
`vcov.c`	N by N variance covariance matrix for choice coefficients (`NULL` unless `vcov=TRUE` is specified).
`vcov.fitted`	N by N variance covariance matrix for fitted values (`NULL` unless `vcov=TRUE` is specified).
`derivatives`	N by D matrix of pointwise partial derivatives based on the Gaussian kernel (`NULL` unless `derivative=TRUE` is specified. If `binary=TRUE` is specified, first differences are returned for binary predictors.
`avgderivatives`	1 by D matrix of average derivative based on the Gaussian kernel (`NULL` unless `derivative=TRUE` is specified. If `binary=TRUE` is specified, average first differences are returned for binary predictors.
`var.avgderivatives`	1 by D matrix of variances for average derivative based on gaussian kernel (`NULL` unless `derivative=TRUE` is specified. If `binary=TRUE` is specified, variances for average first differences are returned for binary predictors.
`binaryindicator`	1 by D matrix that indicates for each predictor if it is treated as binary or not (evaluates to FALSE unless `binary=TRUE` is specified and a predictor is recognized binary.

Note

The function requires the storage of a N by N kernel matrix and can therefore exceed the memory limits for very large datasets.

Setting `derivative=FALSE` and `vcov=FALSE` is useful to reduce computing time if pointwise partial derivatives and or variance covariance matrices are not needed.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

References

Jeremy Ferwerda, Jens Hainmueller, Chad J. Hazlett (2017). Kernel-Based Regularized Least Squares in R (KRLS) and Stata (krls). Journal of Statistical Software, 79(3), 1-26. doi:10.18637/jss.v079.i03

Hainmueller, J. and Hazlett, C. (2014). Kernel Regularized Least Squares: Reducing Misspecification Bias with a Flexible and Interpretable Machine Learning Approach. Political Analysis, 22(2)

Rifkin, R. 2002. Everything Old is New Again: A fresh look at historical approaches in machine learning. Thesis, MIT. September, 2002.

Evgeniou, T., Pontil, M., and Poggio, T. (2000). Regularization networks and support vector machines. Advances In Computational Mathematics, 13(1):1-50.

Schoelkopf, B., Herbrich, R. and Smola, A.J. (2001) A generalized representer theorem. In 14th Annual Conference on Computational Learning Theory, pages 416-426.

Kimeldorf, G.S. Wahba, G. 1971. Some results on Tchebycheffian spline functions. Journal of Mathematical Analysis and Applications, 33:82-95.

See Also

`predict.krls` for fitted values and predictions. `summary.krls` for summary of the fit. `plot.krls` for plots of the fit.

Examples

```
# Linear example
# set up data
N <- 200
x1 <- rnorm(N)
x2 <- rbinom(N,size=1,prob=.2)
y <- x1 + .5*x2 + rnorm(N,0,.15)
X <- cbind(x1,x2)
# fit model
krlsout <- krls(X=X,y=y)
# summarize marginal effects and contribution of each variable
summary(krlsout)
# plot marginal effects and conditional expectation plots
plot(krlsout)


# non-linear example
# set up data
N <- 200
x1 <- rnorm(N)
x2 <- rbinom(N,size=1,prob=.2)
y <- x1^3 + .5*x2 + rnorm(N,0,.15)
X <- cbind(x1,x2)

# fit model
krlsout <- krls(X=X,y=y)
# summarize marginal effects and contribution of each variable
summary(krlsout)
```

```
# plot marginal effects and conditional expectation plots
plot(krlsout)

## 2D example:
# predictor data
X <- matrix(seq(-3,3,.1))
# true function
Ytrue <- sin(X)
# add noise
Y     <- sin(X) + rnorm(length(X),sd=.3)
# approximate function using KRLS
out <- krls(y=Y,X=X)
# get fitted values and ses
fit <- predict(out,newdata=X,se.fit=TRUE)
# results
par(mfrow=c(2,1))
plot(y=Ytrue,x=X,type="l",col="red",ylim=c(-1.2,1.2),lwd=2,main="f(x)")
points(y=fit$fit,X,col="blue",pch=19)
arrows(y1=fit$fit+1.96*fit$se.fit,
       y0=fit$fit-1.96*fit$se.fit,
       x1=X,x0=X,col="blue",length=0)
legend("bottomright",legend=c("true f(x)=sin(x)","KRLS fitted f(x)"),
       lty=c(1,NA),pch=c(NA,19),lwd=c(2,NA),col=c("red","blue"),cex=.8)

plot(y=cos(X),x=X,type="l",col="red",ylim=c(-1.2,1.2),lwd=2,main="df(x)/dx")
points(y=out$derivatives,X,col="blue",pch=19)

legend("bottomright",legend=c("true df(x)/dx=cos(x)","KRLS fitted df(x)/dx"),
       lty=c(1,NA),pch=c(NA,19),lwd=c(2,NA),col=c("red","blue"),,cex=.8)

## 3D example
# plot true function
par(mfrow=c(1,2))
f<-function(x1,x2){ sin(x1)*cos(x2)}
x1 <- x2 <-seq(0,2*pi,.2)
z   <-outer(x1,x2,f)
persp(x1, x2, z,theta=30,main="true f(x1,x2)=sin(x1)cos(x2)")
# approximate function with KRLS
# data and outcomes
X <- cbind(sample(x1,200,replace=TRUE),sample(x2,200,replace=TRUE))
y   <- f(X[,1],X[,2])+ runif(nrow(X))
# fit surface
krlsout <- krls(X=X,y=y)
# plot fitted surface
ff  <- function(x1i,x2i,krlsout){predict(object=krlsout,newdata=cbind(x1i,x2i))$fit}
z   <- outer(x1,x2,ff,krlsout=krlsout)
persp(x1, x2, z,theta=30,main="KRLS fitted f(x1,x2)")
```

---

lambdasearch *Leave-one-out optimization to find λ*

Description

Function conducts leave-one-out optimization to find λ using a golden search search with caching. This function is called internally by krls. It would normally not be called by the user directly.

Usage

```
lambdasearch(L=NULL,
             U=NULL,
             y=NULL,
             Eigenobject=NULL,
             tol=NULL,
             noisy=FALSE,
             eigtrunc=NULL)
```

Arguments

L	Non-negative scalar that determines the lower bound of the search window. Default is NULL which means that the lower bound is found using an algorithm (see details).
U	Positive scalar that determines the upper bound of the search window. Default is NULL which means that the upper bound is found using an algorithm (see details).
y	*N* by *1* matrix of outcomes.
Eigenobject	List that contains the eigenvalues and eigenvectors of the kernel matrix K.
tol	Positive scalar that determines the tolerance used in the optimization routine used to find λ. Default is NULL which means that convergence is achieved when the difference in the sum of squared leave-one-out errors between the i and the i+1 iteration is less than N * 10^-3.
noisy	If TRUE, the function will print details of the golden section search.
eigtrunc	Positive scalar value that determines truncation of eigenvalues for lamnda search window. See krls for details. Default is NULL which means no truncation.

Details

By default, upper bound is found as follows: Set j to n, decrease by one until the following is longer true: sum(EigenValues / (EigenValues + j)) < 1.

By default, upper bound is found as follows: Get the position, q, of the eigenvalue that is closest to max(Eigenvalue)/1000. Set j to 0, increase in steps of 0.05 until the below is longer true: sum(EigenValues / (EigenValues + j)) > q.

Value

A scalar that contains the λ that minimizes the sum of squared leave-one-out errors.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

See Also

krls

---

looloss *Loss Function for Leave One Out Error*

Description

Internal function that computes Leave-On-Out (LOO) Error for KRLS given a fixed value for lambda (the parameter that governs the tradeoff between model fit and complexity in KRLS). This function is called internally by `krls` to find value of lambda that minimizes the LOO error. It would normally not be called by the user directly.

Usage

```
looloss(y = NULL, Eigenobject = NULL,
lambda = NULL,eigtrunc=NULL)
```

Arguments

`y`	n by 1 vector of outcomes.
`Eigenobject`	Object from call to `eigen` that contains spectral decomposition of the n by n Kernel matrix.
`lambda`	Positive scalar value for lamnbda parameter.
`eigtrunc`	Positive scalar value that determines truncation of eigenvalues for lamnda search window. See `krls` for details. Default is `NULL` which means no truncation.

Value

Scalar value for LOO error.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

See Also

krls

plot.krls *Plot method for Kernel-based Regularized Least Squares (KRLS) Model Fits*

Description

Produces two types of plots. The first type of plot shows histograms for the pointwise partial derivatives to examine the heterogeneity in the marginal effects of each predictor (`which==1`). The second type of plot shows estimates of the conditional expectation functions of $E[Y|X]$ for each predictor (`which==2`). For each plot, the predictor of interest varies from its 1st to its 3rd quartile values, while the other predictors are kept at the means (or other values specified in `setx`). For binary varibales the $E[Y|X]$ are predicted at the max and the min value of the predictor (instead of the range from the 1st to the 3rd quantile).

Usage

```
 ## S3 method for class 'krls'
plot(x,which=c(1:2),
 main="distributions of pointwise marginal effects",
 setx="mean",ask = prod(par("mfcol")) < nplots,nvalues=50,probs=c(.25,.75),...)
```

Arguments

`x`	An object of class "`krls`" that results from call to `krls`.
`which`	if a subset of the plots is required, specify a subset of the numbers `1:2`.
`main`	main title for histograms of pointwise partial derivatives.
`setx`	either one of `mean` or `median` to hold other predictors at their mean or median values for the conditional expectation plots. Alternativley the user can specific a numeric vector with predictor values at which the other predictors should be fixed for the conditional expectation plots. If specifed in this way there must be one value per predictor and the order of the values much match the order of the predictor used in the predictor matrix of the krls fit passed in `x`.
`ask`	logical; if `TRUE`, the user is asked before each plot, see `par` (`ask=.`).
`nvalues`	scalar that specifies the number of values at which conditional expectations should be plotted.
`probs`	vector with numbers between 0 and 1 that specify the quantiles that determine the range for of the predictor values for which the conditional expectation should be plotted. By default we vary each predictor from the 1st quartile to the 3rd quartile value.
`...`	additional arguments to be passed to lower level functions

Details

Notice that the historgrams for the partial derivatives can only be plotted if the KRLS object was computed with `krls(,derivatives=TRUE)`.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

See Also

krls

Examples

```
# non-linear example
# set up data
N <- 200
x1 <- rnorm(N)
x2 <- rbinom(N,size=1,prob=.2)
y <- x1^3 + .5*x2 + rnorm(N,0,.15)
X <- cbind(x1,x2)

# fit model
krlsout <- krls(X=X,y=y)
# summarize marginal effects and contribution of each variable
summary(krlsout)
# plot marginal effects and conditional expectation plots
plot(krlsout)
```

---

predict.krls *Predict method for Kernel-based Regularized Least Squares (KRLS) Model Fits*

Description

Predicted values and standard errors based on krls model object.

Usage

```
 ## S3 method for class 'krls'
predict(object, newdata, se.fit = FALSE , ...)
```

Arguments

object	Fitted KRLS model, i.e. an object of class krls
newdata	A data frame or matrix with variables values at which to predict the outcome. Number and order of columns in newdata have to match the corresponding predictors used in the fitted krls model given in object.
se.fit	logical flag if standard errors should be computed for pointwise predictions.
...	additional arguments affecting the predictions produced.

Details

Function produces predicted values, obtained by evaluating the fitted krls function with the newdata (ie. the test points). The prediction at a new test point x_i is based on

$$f(x_i) = \sum_j = 1^n c_j K_{x_j}(x_i)$$

where K is the kernel matrix and thus K_{x_j} is a vector whose j-th entry is $K(x_j, x_i)$ (e.g. the distance between the test point x_i and the training point x_j). The training points are passed to the function with the krls fit in `object`.

When data are missing in `newdata` during prediction, the value of each $k(x_i, x_j)$ is computed by using an adjusted Euclidean distance in the kernel definition. Assume x is D-dimensional but a given pair of observations x_i and x_j have only $D' < D$ non-missing dimensions in common. The adjusted Euclidean distance computes the sum of squared differences over the D' non-missing dimensions, rescales this sum by D/D', and takes the square root. The result corresponds to an assumption that conditional on the observed data, the missing values would not have contributed new information predictive of the outcome.

Value

`fit`	*M* by 1 vector of fitted values for *M* test points.
`se.fit`	*M* by 1 vector of standard errors for the fitted values for *M* test points (`NULL` unless `se.fit=TRUE` is specified).
`vcov.fit`	*M* by *M* variance-covariance matrix for the fitted values for *M* test points (`NULL` unless `se.fit=TRUE` is specified).
`newdata`	*M* by *D* data matrix of of *M* test points with *D* predictors.
`newdataK`	*M* by *N* data matrix for pairwise Gauss Kernel distances between *M* test points and *N* training points from krls model fit in `object`.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

See Also

`krls`

Examples

```
# make up data
X <- seq(-3,3,.1)
Y <- sin(X) + rnorm(length(X),.1)

# fit krls
krlsout <- krls(y=Y,X=X)

# get in-sample prediction
predin <- predict(krlsout,newdata=X,se.fit=TRUE)
```

```
# get out-of-sample prediction
X2 <- runif(5)
predout <- predict(krlsout,newdata=X2,se.fit=TRUE)

# plot true function and predictions
plot(y=sin(X),x=X,type="l",col="red",ylim=c(-1.8,1.8),lwd=2,ylab="f(X)")
points(y=predin$fit,x=X,col="blue",pch=19)
arrows(y1=predin$fit+2*predin$se.fit,
       y0=predin$fit-2*predin$se.fit,
       x1=X,x0=X,col="blue",length=0)

points(y=predout$fit,x=X2,col="green",pch=17)
arrows(y1=predout$fit+2*predout $se.fit,
       y0=predout$fit-2*predout $se.fit,
       x1=X2,x0=X2,col="green",length=0)

legend("bottomright",
       legend=c("true f(x)=sin(X)",
                "KRLS fitted in-sample",
                "KRLS fitted out-of-sample"),
       lty=c(1,NA,NA),pch=c(NA,19,17),
       lwd=c(2,NA,NA),
       col=c("red","blue","green"),
       cex=.8)
```

`solveforc` *Solve for Choice Coefficients in KRLS*

Description

Internal function that computes choice coefficients for KRLS given a fixed value for lambda (the parameter that governs the tradeoff between model fit and complexity in KRLS). This function is called internally by `krls`. It would normally not be called by the user directly.

Usage

```
solveforc(y = NULL, Eigenobject = NULL,
lambda = NULL,eigtrunc=NULL)
```

Arguments

`y`	n by 1 matrix of outcomes.
`Eigenobject`	Object from call to `eigen` that contains spectral decomposition of the n by n Kernel matrix.
`lambda`	Positive scalar value for lamnbda parameter.
`eigtrunc`	Positive scalar value that determines truncation of eigenvalues for lamnda search window. See `krls` for details. Default is `NULL` which means no truncation.

Details

Function relies on fast eigenvalue decomposition method described in method Rifkin and Lippert (2007).

Value

`coeffs`	n by 1 one matrix of choice coefficients for KRLS model.
`Le`	n by 1 matrix of errors from leave-one-out validation.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

References

Rifkin, Ryan M. and Lippert, Ross A. (2007). Notes on Regularized Least Squares. MIT-CSAIL-TR-2007-025. CBCL-268

See Also

`krls`

---

summary.krls — *Summary method for Kernel-based Regularized Least Squares (KRLS) Model Fits*

Description

Summarizes average partial derivatives (i.e. marginal effects) and the distribution of the partial derivatives for each predictor. For binary predictors, the marginal effects are the first differences if `krls(,derivatives=TRUE,binary=TRUE)` was specified.

Usage

```
 ## S3 method for class 'krls'
summary(object, probs=c(.25,.5,.75),...)
```

Arguments

`object`	Fitted krls model, i.e. an object of class krls
`probs`	numeric vector with numbers between 0 and 1 that specify the quantiles of the pointwise marginal effects for the summary (see the `quantile` function for details).
`...`	additional arguments to be passed to lower level functions

Details

Notice that the partial derivatives can only be summarized if the krls object was computed with `krls(,derivatives=TRUE)`.

Value

`coefficients`	matrix with average partial derivates and or first differences (point estimates, standart errors, t-values, p-values).
`qcoefficients`	matrix with 1st, 2nd, and 3rd quatriles of distribution of pointwise marinal effects.

Author(s)

Jens Hainmueller (Stanford) and Chad Hazlett (MIT)

See Also

`krls`

Examples

```
# non-linear example
# set up data
N <- 200
x1 <- rnorm(N)
x2 <- rbinom(N,size=1,prob=.2)
y <- x1^3 + .5*x2 + rnorm(N,0,.15)
X <- cbind(x1,x2)

# fit model
krlsout <- krls(X=X,y=y)
# summarize marginal effects and contribution of each variable
summary(krlsout)
# plot marginal effects and conditional expectation plots
plot(krlsout)
```

Index