Package ‘lazy’

July 20, 2018

Version 1.2-16
Date 2018-07-20
Title Lazy Learning for Local Regression
Author Mauro Birattari <mbiro@ulb.ac.be> and Gianluca Bontempi <gbonte@ulb.ac.be>
Maintainer Gianluca Bontempi <gbonte@ulb.ac.be>
Description By combining constant, linear, and quadratic local models, lazy estimates the value of an unknown multivariate function on the basis of a set of possibly noisy samples of the function itself. This implementation of lazy learning automatically adjusts the bandwidth on a query-by-query basis through a leave-one-out cross-validation.
License GPL (>= 2)
Repository CRAN
NeedsCompilation yes
Date/Publication 2018-07-20 16:40:02 UTC
RoxygenNote 6.0.1

R topics documented:

  lazy ................................................................. 2
  lazy.control ...................................................... 3
  predict.lazy ...................................................... 5

Index 9
Description

By combining constant, linear, and quadratic local models, lazy estimates the value of an unknown multivariate function on the basis of a set of possibly noisy samples of the function itself. This implementation of lazy learning automatically adjusts the bandwidth on a query-by-query basis through a leave-one-out cross-validation.

Usage

```r
lazy(formula, data=NULL, weights, subset, na.action,
     control=lazy.control(...), ...)
```

Arguments

- `formula`: A formula specifying the response and some numeric predictors.
- `data`: An optional data frame within which to look first for the response, predictors, and weights (the latter will be ignored).
- `weights`: Optional weights for each case (ignored).
- `subset`: An optional specification of a subset of the data to be used.
- `na.action`: The action to be taken with missing values in the response or predictors. The default is to stop.
- `control`: Control parameters: see `lazy.control`.
- `...`: Control parameters can also be supplied directly.

Details

For one or more query points, lazy estimates the value of an unknown multivariate function on the basis of a set of possibly noisy samples of the function itself. Each sample is an input/output pair where the input is a vector and the output is a number. For each query point, the estimation of the function is obtained by combining different local models. Local models considered for combination by lazy are polynomials of zeroth, first, and second degree that fit a set of samples in the neighborhood of the query point. The neighbors are selected according to either the Manhattan or the Euclidean distance. It is possible to assign weights to the different directions of the input domain for modifying their importance in the computation of the distance. The number of neighbors used for identifying local models is automatically adjusted on a query-by-query basis through a leave-one-out validations of models, each fitting a different numbers of neighbors. The local models are identified using the recursive least-squares algorithm, and the leave-one-out cross-validation is obtained through the PRESS statistic.

As the name lazy suggests, this function does not do anything... apart from checking the options and properly packing the data. All the actual computation is done when a prediction is request for a specific query point, or for a set of query points: see `predict.lazy`.
Value

An object of class lazy.

Author(s)

Mauro Birattari and Gianluca Bontempi

References


See Also

lazy.control, predict.lazy

Examples

library("lazy")
data(cars)
cars.lazy <- lazy(dist ~ speed, cars)
predict(cars.lazy, data.frame(speed = seq(5, 30, 1)))

lazy.control  Set parameters for lazy learning

Description

Set control parameters for a lazy learning object.

Usage

lazy.control(conIdPar=NULL, linIdPar=1, quaIdPar=NULL, distance=c("manhattan","euclidean"), metric=NULL, cmbPar=1, lambda=1e+06)
Arguments

**conIdPar**

Parameter controlling the number of neighbors to be used for identifying and validating constant models. **conIdPar** can assume different forms:

- **conIdPar=c(idm0, idm0, valm0):** In this case, `idm0:idm0` is the range in which the best number of neighbors is searched when identifying the local polynomial models of degree 0 and where `valm0` is the maximum number of neighbors used for their validation. This means that the constant models identified with `k` neighbors, are validated on the first `v` neighbors, where \( v = \min(k, valm0) \). If `valm0`=0, `valm0` is set to `idm0`: see next case for details.

- **conIdPar=c(idm0, idm0):** Here `idm0` and `idm0` have the same role as in previous case, and `valm0` is by default set to `idm0`: each model is validated on all the neighbors used in identification.

- **conIdPar=p:** Here `idm0` and `idm0` are obtained according to the following formulas: `idm0`=3 and `idm0`=5*p. Recommended choice: `p`=1. As far as the quantity `valm0` is concerned, it gets the default value as in previous case.

- **conIdPar=NULL:** No constant model is considered.

**linIdPar**

Parameter controlling the number of neighbors to be used for identifying and validating linear models. **linIdPar** can assume different forms:

- **linIdPar=c(idm1, idm1, valm1):** In this case, `idm1:idm1` is the range in which the best number of neighbors is searched when identifying the local polynomial models of degree 1 and where `valm1` is the maximum number of neighbors used for their validation. This means that the linear models identified with `k` neighbors, are validated on the first `v` neighbors, where \( v = \min(k, valm1) \). If `valm1`=0, `valm1` is set to `idm1`: see next case for details.

- **linIdPar=c(idm1, idm1):** Here `idm1` and `idm1` have the same role as in previous case, and `valm1` is by default set to `idm1`: each model is validated on all the neighbors used in identification.

- **linIdPar=p:** Here `idm0` and `idm0` are obtained according to the following formulas: `idm1`=3*noPar and `idm1`=5*p*noPar, where `noPar`=nx+1 is the number of parameter of the polynomial model of degree 1, and `nx` is the dimensionality of the input space. Recommended choice: `p`=1. As far as the quantity `valm1` is concerned, it gets the default value as in previous case.

- **linIdPar=NULL:** No linear model is considered.

**quaIdPar**

Parameter controlling the number of neighbors to be used for identifying and validating quadratic models. **quaIdPar** can assume different forms:

- **quaIdPar=c(idm2, idm2, valm2):** In this case, `idm2:idm2` is the range in which the best number of neighbors is searched when identifying the local polynomial models of degree 2 and where `valm2` is the maximum number of neighbors used for their validation. This means that the quadratic models identified with `k` neighbors, are validated on the first `v` neighbors, where \( v = \min(k, valm2) \). If `valm2`=0, `valm2` is set to `idm2`: see next case for details.
quaidPar=c(idm2, idM2): Here idm2 and idM2 have the same role as in previous case, and valM2 is by default set to idM2: each model is validated on all the neighbors used in identification.

quaidPar=p: Here idM0 and idM0 are obtained according to the following formulas: idm2=3*noPar and idM2=5*p*noPar, where in this case the number of parameters is noPar=(nx+1)*(nx+2)/2, and nx is the dimensionality of the input space. Recommended choice: p=1. As far as the quantity valM2 is concerned, it gets the default value as in previous case.

quaidPar=NULL: No quadratic model is considered.

distance The distance metric: can be manhattan or euclidean.

metric Vector of n elements. Weights used to evaluate the distance between query point and neighbors.

cmbPar Parameter controlling the local combination of models. cmbPar can assume different forms:

cmbPar=c(cmb0, cmb1, cmb2): In this case, cmbX is the number of polynomial models of degree X that will be included in the local combination. Each local model will be therefore a combination of the best cmb0 models of degree 0, the best cmb1 models of degree 1, and the best cmb2 models of degree 2 identified as specified by idPar.

cmbPar=cmb: Here cmb is the number of models that will be combined, disregarding any constraint on the degree of the models that will be considered. Each local model will be therefore a combination of the best cmb models, identified as specified by id_par.

lambda Initialization of the diagonal elements of the local variance/covariance matrix for Ridge Regression.

Value

The output of lazy.control is a list containing the following components: conIdPar, linIdPar, quaIdPar, distance, metric, cmbPar, lambda.

Author(s)

Mauro Birattari and Gianluca Bontempi

See Also

lazy, predict.lazy

predict.lazy

Predict method for lazy learning

Description

Obtains predictions from a lazy learning object
Usage

```r
## S3 method for class 'lazy'
predict(object, newdata=NULL,
         t.out=FALSE, k.out=FALSE,
         S.out=FALSE, T.out=FALSE, I.out=FALSE, ...)
```

Arguments

- **object**: Object of class inheriting from `lazy`.
- **newdata**: Data frame (or matrix, vector, etc...) defining of the query points for which a prediction is to be produced.
- **t.out**: Logical switch indicating if the function should return the parameters of the local models used to perform each estimation.
- **k.out**: Logical switch indicating if the function should return the number of neighbors used to perform each estimation.
- **S.out**: Logical switch indicating if the function should return the estimated variance of the prediction suggested by all the models identified for each query point.
- **T.out**: Logical switch indicating if the function should return the parameters of all the models identified for each query point.
- **I.out**: Logical switch indicating if the function should return the index i of all the samples (X[i,], Y[i]) used to perform each estimation.
- **...**: Arguments passed to or from other methods.

Value

The output of the method is a list containing the following components:

- **h**: Vector of q elements, where q is the number of rows in `newdata`, i.e. the number of query points. The element in position i is the estimate of the value of the unknown function in the query point `newdata[i,]`. The component h is always returned.
- **t**: Matrix of z*q elements, where z=2 if at least one model of degree 2 was identified (see `quaIdPar` in `lazy.control`), otherwise z=1 i.e., number of parameters of a linear model if at least one model of degree 1 was identified (see `linIdPar` in `lazy.control`), or z=1 if only models of degree 0 were considered. In the general case, the elements of the vector t[j]=c(a0, a1, ..., an, a11, a12, ..., a22, a23, ..., a33, a34, ..., ann) are the parameters of the local model used for estimating the function in the jth query point: the cross-terms terms a11, a12, ..., ann will be missing if no quadratic model is identified and the terms a1, ..., an, will be missing if no linear model is identified. If, according to `cmbPar` (see `lazy.control`), estimations are to be performed by a combination of models, the elements of t[j] are a weighted average of the parameters of the selected models where the weight of each model is the inverse of the leave-one-out estimate of the variances of the model itself. **REMARK**: a translation of the axes is considered which centers all the local models in the respective query point.
Vector of \( q \) elements. Selected number of neighbors for each query point. If, according to \texttt{cmbPar} (see \texttt{lazy.control}), a local combination of models is considered, \( k[j] \) is the largest value among the number of neighbors used by the selected models for estimating the value in the \( j \)th query point.

**S**

List of up to 3 components: Each component is a matrix containing an estimate, obtained through a leave-one-out cross-validation, of the variance of local models.

- **con** Matrix of \( \text{idm}0*q \) elements, where \( \text{idm}0 \) is the maximum number of neighbors used to fit local polynomial models of degree 0 (see \texttt{lazy.control}): Estimated variance of all the constant models identified for each query point. If no constant model is identified (see \texttt{conIdPar} and \texttt{cmbPar} in \texttt{lazy.control}) \$S$con is not returned.

- **lin** Matrix of \( \text{idm}1*q \) elements, where \( \text{idm}1 \) is the maximum number of neighbors used to fit local polynomial models of degree 1 (see \texttt{lazy.control}): Estimated variance of all the linear models identified for each query point. If no linear model is identified (see \texttt{linIdPar} and \texttt{cmbPar} in \texttt{lazy.control}) \$S$lin is not returned.

- **qua** Matrix of \( \text{idm}2*q \) elements, where \( \text{idm}2 \) is the maximum number of neighbors used to fit local polynomial models of degree 1 (see \texttt{lazy.control}): Estimated variance of all the quadratic models identified for each query point. If no quadratic model is identified (see \texttt{quaIdPar} and \texttt{cmbPar} in \texttt{lazy.control}) \$S$qua is not returned.

The component \( S \) is returned only if \( S\text{.out}=\text{TRUE} \) in the function call.

**T**

List of up to 3 components:

- **con** Array of \( z0*\text{idm}0*q \) elements, where \( z0=1 \) is the number of parameters of a model of degree 0. The element \( \text{T$\text{con}[1,i,j]=a0} \) is the single parameter of the local model identified on \( i \) neighbors of the \( q \)th query point.

- **lin** Array of \( z1*\text{idm}1*q \) elements where, if \( n \) is the dimensionality of the input space, \( z1=n+1 \) is the number of parameter of a model of degree 1. The vector \( \text{T$\text{lin}[i,j]=c(a0,a1,\ldots,an)} \) is the vector of parameters of the local model identified on \( i \) neighbors of the \( q \)th query point. In particular, \( a0 \) is the constant term, \( a1 \) is the parameter associated with the first input variable and so on.

- **qua** Array of \( z2*\text{idm}2*q \) elements where, if \( n \) is the dimensionality of the input space, \( z2=(n+1)*(n+2)/2 \) is the number of parameter of a model of degree 2. The vector \( \text{T$\text{qua}[i,j]=c(a0,a1,\ldots,an,a11,a12,\ldots,a22,\ldots,a33,a34,\ldots)} \) is the vector of parameters of the local quadratic model identified on \( i \) neighbors of the \( q \)th query point. In particular, \( a0,\ldots,a1 \) are the constant and linear parameters as in \( \text{T$\text{lin}} \), while \( a11,a12,\ldots,ann \) are the quadratic ones: \( a11 \) is associated with the quadratic term \( x1^2 \), \( a12 \) with the cross-term \( x1*x2 \), and so on.

**REMARK:** a translation of the axes is considered which centers all the local models in the respective query point. The component \( T \) is returned only if \( T\text{.out}=\text{TRUE} \) in the function call.

**I**

Matrix of \( \text{idm}*q \) elements, where \( \text{idm} \) is the largest of \( \text{idm}0, \text{idm}1, \) and \( \text{idm}2 \). Contains the index of the neighbors of each query point in \texttt{newdata}. In particular, \( I[i,j] \) is the \( i \)th nearest neighbor of the \( q \)th query point.
Author(s)
Mauro Birattari and Gianluca Bontempi

See Also
lazy, lazy.control

Examples
library("lazy")
data(cars)
cars.lazy <- lazy(dist ~ speed, cars)
predict(cars.lazy, data.frame(speed = seq(5, 30, 1)))
Index

*Topic regression
  lazy, 2
  lazy.control, 3
  predict.lazy, 5
  lazy, 2, 5, 8
  lazy.control, 2, 3, 3, 6–8
  predict.lazy, 2, 3, 5, 5
  print.lazy(lazy), 2
  print.summary.lazy(lazy), 2
  summary.lazy(lazy), 2