

Package ‘lazy’

April 6, 2022

Version 1.2-17

Date 2022-04-05

Title Lazy Learning for Local Regression

Author Mauro Birattari <mauro.birattari@ulb.be> and Gianluca Bontempi
<gianluca.bontempi@ulb.be>

Maintainer Theo Verhelst <theo.verhelst@ulb.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 2022-04-06 10:12:30 UTC

RoxygenNote 6.0.1

R topics documented:

lazy	2
lazy.control	3
predict.lazy	5
Index	9

 lazy

Lazy learning for local regression

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

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

Arguments

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

- D.W. Aha (1997) Editorial. *Artificial Intelligence Review*, **11**(1–5), pp. 1–6. Special Issue on Lazy Learning.
- C.G. Atkeson, A.W. Moore, and S. Schaal (1997) Locally Weighted Learning. *Artificial Intelligence Review*, **11**(1–5), pp. 11–73. Special Issue on Lazy Learning.
- W.S. Cleveland, S.J. Devlin, and S.J. Grosse (1988) Regression by Local Fitting: Methods, Prospectives and Computational Algorithms. *Journal of Econometrics*, **37**, pp. 87–114.
- M. Birattari, G. Bontempi, and H. Bersini (1999) Lazy learning meets the recursive least squares algorithm. *Advances in Neural Information Processing Systems 11*, pp. 375–381. MIT Press.
- G. Bontempi, M. Birattari, and H. Bersini (1999) Lazy learning for modeling and control design. *International Journal of Control*, **72**(7/8), pp. 643–658.
- G. Bontempi, M. Birattari, and H. Bersini (1999) Local learning for iterated time-series prediction. *International Conference on Machine Learning*, pp. 32–38. Morgan Kaufmann.

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 $v = \min(k, \text{valM0})$. 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 $v = \min(k, \text{valM1})$. 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 $v = \min(k, \text{valM2})$. If `valM2=0`, `valM2` is set to `idM2`: see next case for details.

	<p>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.</p> <p>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.</p> <p>quaIdPar=NULL: No quadratic model is considered.</p>
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	<p>Parameter controlling the local combination of models. cmbPar can assume different forms:</p> <p>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 <i>the best cmb0 models of degree 0, the best cmb1 models of degree 1, and the best cmb2 models of degree 2</i> identified as specified by idPar.</p> <p>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 <i>the best cmb models</i>, identified as specified by id_par.</p>
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

```
## 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>i</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 <i>q</i> elements, where <i>q</i> is the number of rows in <i>newdata</i> , i.e. the number of query points. The element in position <i>i</i> is the estimate of the value of the unknown function in the query point <i>newdata</i> [<i>i</i> ,]. The component <i>h</i> is always returned.
t	Matrix of <i>z</i> * <i>q</i> elements, where <i>z</i> = <i>z</i> ₂ i.e., number of parameters of a quadratic model if at least one model of degree 2 was identified (see <code>quaIdPar</code> in lazy.control), otherwise <i>z</i> = <i>z</i> ₁ i.e., number of parameters of a linear model if at least one model of degree 1 was identified (see <code>linIdPar</code> in lazy.control), or <i>z</i> =1 if only models of degree 0 were considered. In the general case, the elements of the vector $t[, j]=c(a_0, a_1, \dots, a_n, a_{11}, a_{12}, \dots, a_{22}, a_{23}, \dots, a_{33}, a_{34}, \dots, a_{nn})$ are the parameters of the local model used for estimating the function in the <i>j</i> th query point: the cross-terms terms $a_{11}, a_{12}, \dots, a_{nn}$ will be missing if no quadratic model is identified and the terms a_1, \dots, a_n , will be missing if no linear model is identified. If, according to <code>cmbPar</code> (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 a 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.

- k** Vector of q elements. Selected number of neighbors for each query point. If, according to `cmbPar` (see [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 $idM0 \times q$ elements, where $idM0$ is the maximum number of neighbors used to fit local polynomial models of degree 0 (see [lazy.control](#)): Estimated variance of **all** the constant models identified for each query point. If no constant model is identified (see `conIdPar` and `cmbPar` in [lazy.control](#)) `S$con` is not returned.
 - `lin` Matrix of $idM1 \times q$ elements, where $idM1$ is the maximum number of neighbors used to fit local polynomial models of degree 1 (see [lazy.control](#)): Estimated variance of **all** the linear models identified for each query point. If no linear model is identified (see `linIdPar` and `cmbPar` in [lazy.control](#)) `S$lin` is not returned.
 - `qua` Matrix of $idM2 \times q$ elements, where $idM1$ is the maximum number of neighbors used to fit local polynomial models of degree 1 (see [lazy.control](#)): Estimated variance of **all** the quadratic models identified for each query point. If no quadratic model is identified (see `quaIdPar` and `cmbPar` in [lazy.control](#)) `S$qua` is not returned.
- The component `S` is returned only if `S.out=TRUE` in the function call.
- T** List of up to 3 components:
- `con` Array of $z0 \times idM0 \times q$ elements, where $z0=1$ is the number of parameters of a model of degree 0. The element `T$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 $z1 \times idM1 \times q$ 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 `T$lin[, i, j]=c(a0, a1, ..., 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 $z2 \times idM2 \times q$ 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 `T$qua[, i, j]=c(a0, a1, ..., an, a11, a12, ..., a22, a23, ..., a33, a34, ..., ann)` is the vector of parameters of the local quadratic model identified on i neighbors of the q th query point. In particular, `a0, ..., a1` are the constant and liner parameters as in `T$lin`, while `a11, a12, ..., ann` are the quadratic ones: `a11` is associated with the quadratic term $x1^2$, `a12` with the cross-term $x1 \times x2$, 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.out=TRUE` in the function call.
- I** Matrix of $idM \times q$ elements, where idM is the largest of $idM0$, $idM1$, and $idM2$. Contains the index of the neighbors of each query point in `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

* 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](#)