Package 'loclm'

November 17, 2026

Title Local Linear Regression

Version 1.1.0

Description A bare-bones implementation of local linear regression, using high-level functions. Can handle both numerical and non-numerical (factor) variables. See Loader (1999) <doi:10.1007/b98858>, ch2, or Hastie et al (2009) <doi:10.1007/978-0-387-84858-7>, ch6. The package also contains a scale_df() function which only scales the numeric variables in a dataframe.

License GPL (>= 3)

Encoding UTF-8

Suggests future, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports future.apply, stats

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.0

VignetteBuilder knitr

NeedsCompilation no

Author Jonathan Rougier [aut, cre] (ORCID: <https://orcid.org/0000-0003-3072-7043>)

Maintainer Jonathan Rougier <j.c.rougier@bristol.ac.uk>

Repository CRAN

Date/Publication 2026-11-17 18:20:08 UTC

Contents

hatvalues.loclm 2
loclm 2
LOO_loclm 5
scale_df 6
x2m 7

Index **9**

hatvalues.loclm *Hat Values for Local Linear Regression*

Description

Compute the diagonal of the hat matrix. The trace of the hat matrix (the sum of the diagonal) measures the effective number of parameters.

Usage

```
## S3 method for class 'loclm'
hatvalues(model, ...)
```

Arguments

model	Object with S3 class "loclm", see loclm().
...	For compatibility with S3 generic hatvalues().

Value

A vector of hat values.

Examples

```
## see ?loclm
```

loclm *Local Linear Regression*

Description

A bare-bones implementation of local linear regression, using high-level functions. Can handle both numerical and non-numerical (factor) variables. Non-numerical regressors are converted to factors.

Usage

```
loclm(x, y, span = 0.75)

## S3 method for class 'loclm'
predict(object, newdata, ...)

## S3 method for class 'loclm'
fitted(object, ...)

## S3 method for class 'loclm'
residuals(object, ...)
```

Arguments

x	Dataframe or numeric matrix of regressors. If a dataframe, x must contain at least one numeric variable.
y	Numeric vector of responses.
span	Proportion of points in the neighbourhood. Can be larger than one: span = Inf is equivalent to linear regression.
object	Object with S3 class "loclm".
newdata	Dataframe or matrix at which to predict, defaults to x.
...	For compatibility with S3 generic predict().

Details

Local linear regression fits a plane to the points in the neighbourhood of x, in order to predict $y(x)$. These points are weighted in the fit using the tricube weighting function. Because the neighbourhood and the weights depend on a measure of distance, numeric variables in x and newdata are prescaled using scale_df(). Factors are converted to columns of dummy variables.

The distance between different levels of the same factor (in dummy variables) is 1 or sqrt(2). The mean distance between two independent standard Normal random variables is a little over one (about 1.13). Therefore standardizing the numeric variables and dummying the factors puts all variables on roughly the same footing, as far as distance is concerned.

The behaviour with missing values can be complicated, and therefore only complete cases are used, see complete.cases().

Value

A list with S3 class "loclm". This has a predict() method which returns a numeric vector of point predictions for the rows of newdata.

There are also fitted() and residuals() methods, although these are just wrappers for predict().

Quadratic regression

Modifying the code to use quadratic regression is fairly straightforward: the main difference is that the model matrix is not longer the same as the data matrix. However, there can be a lot of extra terms, especially when including factors with more than two levels. This creates a prediction with low bias but high variance, especially for extrapolation. For four or fewer numeric variables, use stats::loess() with degree = 2 (the default). Otherwise, change the code at your own risk.

Parallel implementation

predict.loclm() uses future.apply::future_vapply() under the hood. The default is sequential (one core). If newdata has many rows it will be faster to use multiple cores using

```
plan(multisession)
```

before calling predict.loclm(). See the vignette for an example, and future::plan() for more details about setting up a plan.

References

More detail about local regression:

T. Hastie, R. Tibshirani, J. Friedman (2009), The Elements of Statistical Learning, Springer, 2nd ed, ch6.

C. Loader (1999), Local Regression and Likelihood, Springer, ch2.

See Also

`stats::loess()` is faster and more flexible. However, this function can only handle up to four variables, all numeric.

`LOO_loclm()` for leave-one-out cross-validation to set span.

`hatvalues.loclm()` calculates hat values.

`x2m()` creates the model matrix (not including the constant). This is a faster version than working through `model.matrix()`.

Examples

```
## loess example

fit <- loclm(cars[, "speed", drop=FALSE], cars$dist)
newd <- data.frame(speed = seq(5, 30, 1))
yhat <- predict(fit, newd)
cex <- 0.5
plot(newd$speed, yhat, pch = 19, cex = cex, xlab = "Speed",
  ylab = "Predicted distance", main = "Cars dataset")

## stackloss data

x <- stack.x
y <- stack.loss
fit <- loclm(x, y, span = 0.5)
yhat <- fitted(fit)
cex <- 0.8
plot(y, yhat, pch = 19, cex = cex, xlab = "Actual",
  ylab = "Predicted", main = "Stackloss dataset", asp = 1)
abline(a = 0, b = 1, lty = 2)

## add bogus factor

x <- as.data.frame(x)
x$ferret <- factor(rep(LETTERS[1:3], length.out = nrow(x)))
y <- y + as.numeric(x$ferret)

fit <- loclm(x, y, span = 0.5)
show(head(fit$m)) # now with ferret!
yhat <- fitted(fit)
plot(y, yhat, pch = 19, cex = 0.8, xlab = "Actual",
  ylab = "Predicted", main = "Stackloss dataset, with ferret",
  asp = 1)
```

```
abline(a = 0, b = 1, lty = 2)

## hat values

infl <- hatvalues(fit)
show(sum(infl)) # 17.6, effective number of parameters
fit0 <- loclm(x, y, span = Inf) # linear
infl0 <- hatvalues(fit0)
show(sum(infl0)) # 6 parameters

## compare with linear model

fit1 <- lm(y ~ ., data.frame(x, y = y))
infl1 <- hatvalues(fit1)
all.equal(infl0, infl1, check.attributes = FALSE) # TRUE

## some timings

big.x <- data.frame(A = x, B = x)
big.x <- do.call("rbind", rep(list(big.x), 100)) # 2100 x 8
big.y <- rep(y, length.out = nrow(big.x))
system.time({
  fit <- loclm(big.x, big.y)
  yhat <- predict(fit, big.x)
}) # about 2s

# see the vignette for parallel predict
```

`LOO_loclm` *Leave-One-Out Tuning of the Span*

Description

This is a quick version of leave-one-out, in which, for each row of the training dataset, the model is trained on all other rows and used to predict that row.

Usage

```
LOO_loclm(object, span = NULL)
```

Arguments

`object`	Object with S3 class `"loclm"`.
`span`	Proportion of points in the neighbourhood. Can be larger than one: `span = Inf` is equivalent to linear regression.

Value

A vector of residuals. For tuning, several candidate values for the span can be evaluated in parallel, see the vignette.

Examples

```
## stackloss data

x <- stack.x
y <- stack.loss
fit <- loclm(x, y, span = 0.5)

## LOO-RMSE for different spans

span_vals <- sort(c(seq(0.7, 1.6, 0.1), 0.75))
rmse <- sapply(span_vals, \(span) {
  loo_resid <- LOO_loclm(fit, span = span)
  sqrt(mean(loo_resid^2))
})
show(cbind(span = span_vals, RMSE = rmse))
```

---

scale_df *Scale a Dataframe*

Description

Only rescale the numeric variables in a dataframe.

Warning: does not retain the default functionality of `base::scale()`, because the returned object is always a dataframe. If `base::scale()` is called with a dataframe it will convert the returned object into a matrix.

INCLUDED FOR BACK COMPATIBILITY. NO LONGER USED IN `loclm()`.

Usage

```
scale_df(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	Dataframe.
`center`	Either a logical value or a numeric vector of length equal to the number of numeric columns of `x`.
`scale`	Either a logical value or a numeric vector of length equal to the number of numeric columns of `x`.

Details

If `center` is numeric, `names(center)` identifies the numeric columns in `x`. Otherwise, if `scale` is numeric, `names(scale)` identifies the numeric columns in `x`. `center` and `scale` are passed to `base::scale()`.

Value

A dataframe like `x`, with numeric columns centered and/or scaled. The numeric centering and scalings used (if any) are returned as attributes `"scaled:center"` and `"scaled:scale"`.

See Also

`base::scale()`, which this function uses but only on a subset of the columns.

Examples

```
n <- 11
X <- data.frame(
  x = rnorm(n),
  foo = factor(sample(LETTERS[1:3], n, replace = TRUE)),
  bar = sample(LETTERS[4:5], n, replace = TRUE),
  y = rexp(n, rate = 0.2))
print(Y <- scale_df(X))
str(Y) # see attributes

## scale only

sc <- attr(Y, "scaled:scale") # has names
print(Z <- scale_df(X, center = FALSE, scale = sc))

## no numeric variables

Y <- X[, c("foo", "bar")]
Z <- scale_df(Y)
identical(Y, Z) # TRUE

## error messages

try(Y <- scale_df(X, center = c(foo = 1, bob = 2)))
try(Y <- scale_df(X, scale = c(foo = 1, bob = 2)))
```

---

x2m *Dataframe to Model Matrix*

Description

Rescales numeric variables, expands out factors, and drops variables with no variation.

Usage

```
x2m(x, scale = NULL)
```

Arguments

x	Matrix or dataframe.
scale	Parameter of the transformation, see Details.

Details

The scaled argument is a named list including the variables in x. If scale[[j]] is 0 the variable is ignored. If it is postive, the numeric variable is rescaled. If is it a character vector of levels, the factor is expanded out into dummy variables, dropping the first level.

If scale = NULL, scale is deduced from x.

Value

A numeric model matrix, *not* including a constant, with a "scale" attribute.

Examples

```
## stackloss data

x <- data.frame(stack.x)
lev <- c("sable", "chocolate", "champagne", "silver")
x$Ferret <- factor(sample(lev, nrow(x), replace = TRUE), lev)
x$boring <- rep(17.2, nrow(x))
x$also_boring <- factor(rep(lev[2], nrow(x)), lev)
M <- x2m(x)
show(head(M))

sc <- attr(M, "scale")
y <- x[, sample.int(ncol(x))] # permute
My <- x2m(y, sc)
identical(M, My) # TRUE
```

Index

base::scale(), *7*

complete.cases(), *3*

fitted.loclm *(*loclm*)*, 2
future.apply::future_vapply(), *3*
future::plan(), *3*

hatvalues(), *2*
hatvalues.loclm, 2
hatvalues.loclm(), *4*

loclm, 2
loclm(), *2*, *6*
LOO_loclm, 5
LOO_loclm(), *4*

model.matrix(), *4*

predict(), *3*
predict.loclm *(*loclm*)*, 2

residuals.loclm *(*loclm*)*, 2

scale_df, 6
scale_df(), *3*
stats::loess(), *3*, *4*

x2m, 7
x2m(), *4*