Package ‘glmnetUtils’

August 1, 2017

Type Package

Version 1.1

Title Utilities for 'Glmnet'

Description Provides a formula interface for the 'glmnet' package for elasticnet regression, a method for cross-validating the alpha parameter, and other quality-of-life tools.

Imports stats, graphics, grDevices, glmnet, parallel, Matrix

Suggests knitr, rmarkdown, MASS, doParallel, testthat

Author Microsoft [cph], Hong Ooi [aut, cre]

Maintainer Hong Ooi <hongooi@microsoft.com>

NeedsCompilation no

VignetteBuilder knitr

Copyright Microsoft

License GPL-2

URL https://github.com/Hong-Revo/glmnetUtils

Collate 'glmnetUtils.r' 'cvGlmnetFormula.r' 'cvaGlmnetFormula.r' 'glmnetFormula.r' 'makeModelComponents.r'

RoxygenNote 6.0.1

Repository CRAN

Date/Publication 2017-08-01 12:37:24 UTC

R topics documented:

`cv.glmnet` *Formula interface for elastic net cross-validation with cv.glmnet*

Description

Formula interface for elastic net cross-validation with cv.glmnet

Usage

```
cv.glmnet(x, ...)

## Default S3 method:
cv.glmnet(x, y, ...)

## S3 method for class 'formula'
cv.glmnet(formula, data, alpha = 1, nfolds = 10, ...,
  weights = NULL, offset = NULL, subset = NULL,
  na.action = getOption(”na.action”), drop.unused.levels = FALSE,
  xlev = NULL, sparse = FALSE, use.model.frame = FALSE)

## S3 method for class 'cv.glmnet.formula'
predict(object, newdata, na.action = na.pass, ...)

## S3 method for class 'cv.glmnet.formula'
coef(object, ...)

## S3 method for class 'cv.glmnet.formula'
print(x, ...)
```

Arguments

`x` For the default method, a matrix of predictor variables.

`...` For `cv.glmnet.formula` and `cv.glmnet.default`, other arguments to be passed to glmnet::cv.glmnet; for the `predict` and `coef` methods, arguments to be passed to their counterparts in package glmnet.

`y` For the default method, a response vector or matrix (for a multinomial response).

`formula` A model formula; interaction terms are allowed and will be expanded per the usual rules for linear models.

`data` A data frame or matrix containing the variables in the formula.

`alpha` The elastic net mixing parameter. See glmnet::glmnet for more details.

`nfolds` The number of crossvalidation folds to use. See glmnet::cv.glmnet for more details.

`weights` An optional vector of case weights to be used in the fitting process. If missing, defaults to an unweighted fit.

`offset` An optional vector of offsets, an *a priori* known component to be included in the linear predictor.

`subset` An optional vector specifying the subset of observations to be used to fit the model.

`na.action` A function which indicates what should happen when the data contains missing values. For the `predict` method, `na.action = na.pass` will predict missing values with `NA`; `na.omit` or `na.exclude` will drop them.

`drop.unused.levels`
Should factors have unused levels dropped? Defaults to `FALSE`.

`xlev` A named list of character vectors giving the full set of levels to be assumed for each factor.

`sparse` Should the model matrix be in sparse format? This can save memory when dealing with many factor variables, each with many levels.

`use.model.frame`
Should the base model.frame function be used when constructing the model matrix? This is the standard method that most R modelling functions use, but has some disadvantages. The default is to avoid `model.frame` and construct the model matrix term-by-term; see discussion.

`object` For the `predict` and `coef` methods, an object of class `cv.glmnet.formula`.

`newdata` For the `predict` method, a data frame containing the observations for which to calculate predictions.

Details

The `cv.glmnet` function in this package is an S3 generic with a formula and a default method. The former calls the latter, and the latter is simply a direct call to the `cv.glmnet` function in package `glmnet`. All the arguments to `glmnet::cv.glmnet` are (or should be) supported.

There are two ways in which the matrix of predictors can be generated. The default, with `use.model.frame = FALSE`, is to process the additive terms in the formula independently. With wide datasets, this is much faster and more memory-efficient than the standard R approach which uses the `model.frame` and `model.matrix` functions. However, the resulting model object is not exactly the same as if the standard approach had been used; in particular, it lacks a bona fide terms object. If you require interoperability with other packages that assume the standard model object structure, set `use.model.frame = TRUE`. See discussion for more information on this topic.

The `predict` and `coef` methods are wrappers for the corresponding methods in the glmnet package. The former constructs a predictor model matrix from its `newdata` argument and passes that as the `newx` argument to `glmnet:::predict.cv.glmnet`.

Value

For `cv.glmnet.formula`, an object of class `cv.glmnet.formula`. This is basically the same object created by `glmnet::cv.glmnet`, but with extra components to allow formula usage.

See Also

glmnet::cv.glmnet, glmnet::predict.cv.glmnet, glmnet::coef.cv.glmnet, model.frame, model.matrix

Examples

```
cv.glmnet(mpg ~ ., data=mtcars)

cv.glmnet(Species ~ ., data=iris, family="multinomial")

## Not run:

# Leukemia example dataset from Trevor Hastie's website
download.file("http://web.stanford.edu/~hastie/glmnet/glmnetData/Leukemia.RData",
              "Leukemia.RData")
load("Leukemia.Rdata")
leuk <- do.call(data.frame, Leukemia)
cv.glmnet(y ~ ., leuk, family="binomial")

## End(Not run)
```

---

cva.glmnet *Do elastic net cross-validation for alpha and lambda simultaneously*

Description

Do elastic net cross-validation for alpha and lambda simultaneously

Usage

```
cva.glmnet(x, ...)

## Default S3 method:
cva.glmnet(x, y, alpha = seq(0, 1, len = 11)^3,
  nfolds = 10, foldid = sample(nfolds, nrow(x), replace = TRUE), ...,
  outerParallel = NULL, checkInnerParallel = TRUE)

## S3 method for class 'formula'
cva.glmnet(formula, data, ..., weights = NULL,
  offset = NULL, subset = NULL, na.action = getOption("na.action"),
  drop.unused.levels = FALSE, xlev = NULL, sparse = FALSE,
  use.model.frame = FALSE)

## S3 method for class 'cva.glmnet'
predict(object, newx, alpha, which = match(TRUE,
  abs(object$alpha - alpha) < 1e-08), ...)

## S3 method for class 'cva.glmnet.formula'
predict(object, newdata, alpha,
  which = match(TRUE, abs(object$alpha - alpha) < 1e-08),
  na.action = na.pass, ...)

## S3 method for class 'cva.glmnet'
```

```
coef(object, alpha, which = match(TRUE, abs(object$alpha
  - alpha) < 1e-08), ...)

## S3 method for class 'cva.glmnet.formula'
print(x, ...)

## S3 method for class 'cva.glmnet'
plot(x, ...)

minlossplot(x, ...)

## S3 method for class 'cva.glmnet'
minlossplot(x, ..., cv.type = c("1se", "min"))
```

Arguments

`x` A matrix of predictor variables; or for the plotting methods, an object returned by `cva.glmnet`.

`...` Further arguments to be passed to lower-level functions. In the case of `cva.glmnet`, these arguments are passed to `cv.glmnet`; for `predict` and `coef`, they are passed to `predict.cv.glmnet`; and for `plot` and `minlossplot`, to `plot`.

`y` A response vector or matrix (for a multinomial response).

`alpha` A vector of alpha values for which to do cross-validation. The default is a sequence of 11 values more closely spaced around alpha = 0. For the `predict` and `coef` methods, the specific value of alpha for which to return predictions/regression coefficients.

`nfolds` The number of cross-validation folds to use. Defaults to 10.

`foldid` Vector of fold IDs for cross-validation. See glmnet::cv.glmnet.

`outerParallel` Method of parallelising the outer loop over alpha. See 'Details' below. If `NULL`, the loop is run sequentially.

`checkInnerParallel`
If the outer loop is run in parallel, check that the inner loop over lambda will not be in contention for cores.

`formula` A model formula; interaction terms are allowed and will be expanded per the usual rules for linear models.

`data` A data frame or matrix containing the variables in the formula.

`weights` An optional vector of case weights to be used in the fitting process. If missing, defaults to an unweighted fit.

`offset` An optional vector of offsets, an *a priori* known component to be included in the linear predictor.

`subset` An optional vector specifying the subset of observations to be used to fit the model.

`na.action` A function which indicates what should happen when the data contains missing values. For the `predict` method, `na.action = na.pass` will predict missing values with `NA`; `na.omit` or `na.exclude` will drop them.

`drop.unused.levels`
: Should factors have unused levels dropped? Defaults to `FALSE`.

`xlev`
: A named list of character vectors giving the full set of levels to be assumed for each factor.

`sparse`
: Should the model matrix be in sparse format? This can save memory when dealing with many factor variables, each with many levels.

`use.model.frame`
: Should the base model.frame function be used when constructing the model matrix? This is the standard method that most R modelling functions use, but has some disadvantages. The default is to avoid `model.frame` and construct the model matrix term-by-term; see discussion.

`object`
: For the `predict` and `coef` methods, an object returned by `cva.glmnet`.

`newx`
: For the `predict` method, a matrix of predictor variables.

`which`
: An alternative way of specifying alpha; the index number of the desired value within the alpha vector. If both `which` and `alpha` are supplied, the former takes precedence.

`newdata`
: For the `predict` and `coef` methods, a data frame containing the observations for which to calculate predictions.

`cv.type`
: For `minlossplot`, which cross-validated loss value to plot for each value of alpha. This can be either `"min"` which is the minimum loss, or `"1se"` which is the highest loss within 1 standard error of the minimum. The default is `"1se"`.

Details

The `cva.glmnet` function does simultaneous cross-validation for both the alpha and lambda parameters in an elastic net model. The procedure is as outlined in the documentation for glmnet::cv.glmnet: it creates a vector `foldid` allocating the observations into folds, and then calls `cv.glmnet` in a loop over different values of alpha, but the same values of `foldid` each time.

Optionally this loop over alpha can be parallelised; currently, `cva.glmnet` knows about two methods of doing so:

- Via parLapply in the parallel package. To use this, set `outerParallel` to a valid cluster object created by makeCluster.
- Via `rxExec` as supplied by Microsoft R Server's RevoScaleR package. To use this, set `outerParallel` to a valid compute context created by `RxComputeContext`, or a character string specifying such a context.

If the outer loop is run in parallel, `cva.glmnet` can check if the inner loop (over lambda) is also set to run in parallel, and disable this if it would lead to contention for cores. This is done if it is likely that the parallelisation is local on a multicore machine, ie if `outerParallel` is a `SOCKcluster` object running on `"localhost"`, or if the RevoScaleR compute context is local parallel.

There are two ways in which the matrix of predictors can be generated. The default, with `use.model.frame = FALSE`, is to process the additive terms in the formula independently. With wide datasets, this is much faster and more memory-efficient than the standard R approach which uses the `model.frame` and `model.matrix` functions. However, the resulting model object is not exactly the same as if the standard approach had been used; in particular, it lacks a bona fide terms object. If you require interoperability with other packages that assume the standard model object structure, set `use.model.frame = TRUE`. See discussion for more information on this topic.

The `predict` method computes predictions for a specific alpha value given a `cva.glmnet` object. It looks up the supplied alpha (possibly supplied indirectly via the `which` argument) in the object's stored `alpha` vector, and calls `glmnet:::predict.cv.glmnet` on the corresponding `cv.glmnet` fit. All the arguments to that function are (or should be) supported.

The `coef` method is similar, returning the coefficients for the selected alpha value via `glmnet:::coef.cv.glmnet`.

The plot method for `cva.glmnet` objects plots the average cross-validated loss by lambda, for each value of alpha. Each line represents one `cv.glmnet` fit, corresponding to one value of alpha. Note that the specific lambda values can vary substantially by alpha.

The `minlossplot` function gives the best (lowest) cross-validated loss for each value of alpha.

Value

For `cva.glmnet.default`, an object of class `cva.glmnet`. This is a list containing the following:

- `alpha` The vector of alpha values
- `nfolds` The number of folds
- `modlist` A list of `cv.glmnet` objects, containing the cross-validation results for each value of alpha

The function `cva.glmnet.formula` adds a few more components to the above, to facilitate working with formulas.

For the `predict` method, a vector or matrix of predicted values.

For the `coef` method, a vector of regularised regression coefficients.

See Also

glmnet::cv.glmnet

glmnet::predict.cv.glmnet, glmnet::coef.cv.glmnet

cva.glmnet, glmnet::cv.glmnet, plot

Examples

```
cva <- cva.glmnet(mpg ~ ., data=mtcars)
predict(cva, mtcars, alpha=1)

## Not run:

# Leukemia example dataset from Trevor Hastie's website
download.file("http://web.stanford.edu/~hastie/glmnet/glmnetData/Leukemia.RData",
              "Leukemia.RData")
load("Leukemia.Rdata")
leuk <- do.call(data.frame, Leukemia)
leuk.cva <- cva.glmnet(y ~ ., leuk, family="binomial")
leuk.pred <- predict(leuk.cva, leuk, which=6)

## End(Not run)
```

glmnet *Formula interface for elastic net modelling with glmnet*

Description

Formula interface for elastic net modelling with glmnet

Usage

```
glmnet(x, ...)

## Default S3 method:
glmnet(x, y, ...)

## S3 method for class 'formula'
glmnet(formula, data, alpha = 1, ..., weights = NULL,
  offset = NULL, subset = NULL, na.action = getOption("na.action"),
  drop.unused.levels = FALSE, xlev = NULL, sparse = FALSE,
  use.model.frame = FALSE)

## S3 method for class 'glmnet.formula'
predict(object, newdata, offset = NULL,
  na.action = na.pass, ...)

## S3 method for class 'glmnet.formula'
coef(object, ...)

## S3 method for class 'glmnet.formula'
print(x, digits = max(3, getOption("digits") - 3),
  print.deviance.ratios = FALSE, ...)
```

Arguments

`x` For the default method, a matrix of predictor variables.

`...` For `glmnet.formula` and `glmnet.default`, other arguments to be passed to glmnet::glmnet; for the `predict` and `coef` methods, arguments to be passed to their counterparts in package glmnet.

`y` For the default method, a response vector or matrix (for a multinomial response).

`formula` A model formula; interaction terms are allowed and will be expanded per the usual rules for linear models.

`data` A data frame or matrix containing the variables in the formula.

`alpha` The elastic net mixing parameter. See glmnet::glmnet for more details.

`weights` An optional vector of case weights to be used in the fitting process. If missing, defaults to an unweighted fit.

`offset` An optional vector of offsets, an *a priori* known component to be included in the linear predictor.

`subset` An optional vector specifying the subset of observations to be used to fit the model.

`na.action` A function which indicates what should happen when the data contains missing values. For the `predict` method, `na.action = na.pass` will predict missing values with `NA`; `na.omit` or `na.exclude` will drop them.

`drop.unused.levels`
Should factors have unused levels dropped? Defaults to `FALSE`.

`xlev` A named list of character vectors giving the full set of levels to be assumed for each factor.

`sparse` Should the model matrix be in sparse format? This can save memory when dealing with many factor variables, each with many levels.

`use.model.frame`
Should the base model.frame function be used when constructing the model matrix? This is the standard method that most R modelling functions use, but has some disadvantages. The default is to avoid `model.frame` and construct the model matrix term-by-term; see discussion.

`object` For the `predict` and `coef` methods, an object of class `glmnet.formula`.

`newdata` For the `predict` method, a data frame containing the observations for which to calculate predictions.

`digits` Significant digits in printed output.

`print.deviance.ratios`
Whether to print the table of deviance ratios, as per glmnet::print.glmnet.

Details

The `glmnet` function in this package is an S3 generic with a formula and a default method. The former calls the latter, and the latter is simply a direct call to the `glmnet` function in package glmnet. All the arguments to `glmnet::glmnet` are (or should be) supported.

There are two ways in which the matrix of predictors can be generated. The default, with `use.model.frame = FALSE`, is to process the additive terms in the formula independently. With wide datasets, this is much faster and more memory-efficient than the standard R approach which uses the `model.frame` and `model.matrix` functions. However, the resulting model object is not exactly the same as if the standard approach had been used; in particular, it lacks a bona fide terms object. If you require interoperability with other packages that assume the standard model object structure, set `use.model.frame = TRUE`. See discussion for more information on this topic.

The `predict` and `coef` methods are wrappers for the corresponding methods in the glmnet package. The former constructs a predictor model matrix from its `newdata` argument and passes that as the `newx` argument to `glmnet:::predict.glmnet`.

Value

For `glmnet.formula`, an object of class `glmnet.formula`. This is basically the same object created by `glmnet::glmnet`, but with extra components to allow formula usage.

See Also

glmnet::glmnet, glmnet::predict.glmnet, glmnet::coef.glmnet, model.frame, model.matrix

Examples

```
glmnet(mpg ~ ., data=mtcars)

glmnet(Species ~ ., data=iris, family="multinomial")

## Not run:

# Leukemia example dataset from Trevor Hastie's website
download.file("http://web.stanford.edu/~hastie/glmnet/glmnetData/Leukemia.RData",
              "Leukemia.RData")
load("Leukemia.Rdata")
leuk <- do.call(data.frame, Leukemia)
glmnet(y ~ ., leuk, family="binomial")

## End(Not run)
```

glmnet.model.matrix *Model matrix options for glmnet*

Description

This page describes the options available for generating the model matrix.

Details

There are two ways in which glmnetUtils can generate a model matrix out of a formula and data frame. The first is to use the standard R machinery comprising model.frameand model.matrix; and the second is to build the matrix one variable at a time. These options are discussed and contrasted below.

Using `model.frame`

This is the simpler option, and the one that is most compatible with other R modelling functions. The `model.frame` function takes a formula and data frame and returns a *model frame*: a data frame with special information attached that lets R make sense of the terms in the formula. For example, if a formula includes an interaction term, the model frame will specify which columns in the data relate to the interaction, and how they should be treated. Similarly, if the formula includes expressions like `exp(x)` or `I(x^2)` on the RHS, `model.frame` will evaluate these expressions and include them in the output.

The major disadvantage of using `model.frame` is that it generates a terms object, which encodes how variables and interactions are organised. One of the attributes of this object is a matrix with one row per variable, and one column per main effect and interaction. At minimum, this is (approximately) a $p \times p$ square matrix where p is the number of main effects in the model. For wide datasets with $p > 10000$, this matrix can approach or exceed a gigabyte in size. Even if there is enough memory to store such an object, generating the model matrix can take a significant amount of time.

Another issue with the standard R approach is the treatment of factors. Normally, `model.matrix` will turn an N-level factor into an indicator matrix with $N - 1$ columns, with one column being dropped. This is necessary for unregularised models as fit with `lm` and `glm`, since the full set of N columns is linearly dependent. With the usual treatment contrasts, the interpretation is that the dropped column represents a baseline level, while the coefficients for the other columns represent the difference in the response relative to the baseline.

This may not be appropriate for a regularised model as fit with glmnet. The regularisation procedure shrinks the coefficients towards zero, which forces the estimated differences from the baseline to be smaller. But this only makes sense if the baseline level was chosen beforehand, or is otherwise meaningful as a default; otherwise it is effectively making the levels more similar to an arbitrarily chosen level.

Manually building the model matrix

To deal with the problems above, glmnetUtils by default will avoid using `model.frame`, instead building up the model matrix term-by-term. This avoids the memory cost of creating a `terms` object, and can be noticeably faster than the standard approach. It will also include one column in the model matrix for *all* levels in a factor; that is, no baseline level is assumed. In this situation, the coefficients represent differences from the overall mean response, and shrinking them to zero *is* meaningful (usually).

This works in an additive fashion, ie the formula `~ a + b:c + d*e` is treated as consisting of three terms, `a`, `b:c` and `d*e` each of which is processed independently of the others. A dot in the formula includes all main effect terms, ie `~ . + a:b + f(x)` expands to `~ a + b + x + a:b + f(x)` (assuming a, b and x are the only columns in the data). Note that a formula like `~ (a + b) + (c + d)` will be treated as two terms, `a + b` and `c + d`.

The code can handle fairly complex formulas, but it is not as sophisticated as base `model.frame` and `model.matrix`. In particular, terms that are to be *omitted* from the model must be at the end of the formula: `~ . - c` works, but not `~ -c + .`.

`glmnetUtils` *Utilities for glmnet*

Description

Some quality-of-life functions to streamline the process of fitting elastic net models with the `glmnet` package, specifically:

- `glmnet.formula` provides a formula/data frame interface to `glmnet`.
- `cv.glmnet.formula` does a similar thing for `cv.glmnet`.
- Methods for `predict` and `coef` for both the above.
- A function `cva.glmnet` to choose both the alpha and lambda parameters via cross-validation, following the approach described in the help page for `cv.glmnet`. Optionally does the cross-validation in parallel.
- Methods for `plot`, `predict` and `coef` for the above.

Index