Package 'multiview'

September 2, 2022

Type Package

Title Cooperative Learning for Multi-View Analysis

Version 0.4

Date 2022-08-27

VignetteBuilder knitr

Depends R (>= 3.5.0)

Description Cooperative learning combines the usual squared error loss of predictions with an agreement penalty to encourage the predictions from different data views to agree. By varying the weight of the agreement penalty, we get a continuum of solutions that include the well-known early and late fusion approaches. Cooperative learning chooses the degree of agreement (or fusion) in an adaptive manner, using a validation set or cross-validation to estimate test set prediction error. In the setting of cooperative regularized linear regression, the method combines the lasso penalty with the agreement penalty (Ding, D., Li, S., Narasimhan, B., Tibshirani, R. (2021) <arXiv:2112.12337>).

License GPL-2

Encoding UTF-8

RoxygenNote 7.2.1

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

Imports glmnet, Matrix, methods, RColorBrewer, Rcpp, stats, survival, utils

Config/testthat/edition 3

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

Author Daisy Yi Ding [aut], Robert J. Tibshirani [aut], Balasubramanian Narasimhan [aut, cre], Trevor Hastie [aut], Kenneth Tay [aut], James Yang [aut]

Maintainer Balasubramanian Narasimhan <naras@stanford.edu>

Repository CRAN

Date/Publication 2022-09-02 08:00:02 UTC

R topics documented:

`multiview-package` *Cooperative learning for multiple views using generalized linear models*

Description

This package performs a version of early and late fusion of multiple views using penalized generalized regression.

`coef.cv.multiview` *Extract coefficients from a cv.multiview object*

Description

Extract coefficients from a cv.multiview object

Usage

```
## S3 method for class 'cv.multiview'
coef(object, s = c("lambda.1se", "lambda.min"), ...)
```

Arguments

`object`	Fitted `"cv.multiview"` object.
`s`	Value(s) of the penalty parameter `lambda` at which predictions are required. Default is the value `s="lambda.1se"` stored on the CV `object`. Alternatively `s="lambda.min"` can be used. If `s` is numeric, it is taken as the value(s) of `lambda` to be used. (For historical reasons we use the symbol 's' rather than 'lambda' to reference this parameter.)
`...`	This is the mechanism for passing arguments like `x=` when `exact=TRUE`; see `exact` argument.

Value

the matrix of coefficients for specified lambda.

Examples

```
set.seed(1)
x = matrix(rnorm(100*20), 100, 20)
z = matrix(rnorm(100*20), 100, 20)
U = matrix(rnorm(100*5), 100, 5)
for (m in seq(5)){
    u = rnorm(100)
    x[, m] = x[, m] + u
    z[, m] = z[, m] + u
    U[, m] = U[, m] + u}
x = scale(x, center = TRUE, scale = FALSE)
z = scale(z, center = TRUE, scale = FALSE)
beta_U = c(rep(0.1, 5))
y = U %*% beta_U + 0.1 * rnorm(100)
fit1 = cv.multiview(list(x=x,z=z), y, rho = 0.3)
coef(fit1, s="lambda.min")

# Binomial

by = 1 * (y > median(y))
```

```
fit2 = cv.multiview(list(x=x,z=z), by, family = binomial(), rho = 0.9)
coef(fit2, s="lambda.min")

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = cv.multiview(list(x=x,z=z), py, family = poisson(), rho = 0.6)
coef(fit3, s="lambda.min")
```

`coef.multiview` *Extract coefficients from a multiview object*

Description

Extract coefficients from a multiview object

Usage

```
## S3 method for class 'multiview'
coef(object, s = NULL, ...)
```

Arguments

`object`	Fitted "multiview" object.
`s`	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model.
`...`	This is the mechanism for passing arguments like `x=` when `exact=TRUE`; see `exact` argument.

Value

a matrix of coefficients for specified lambda.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
z = matrix(rnorm(100 * 10), 100, 10)
y = rnorm(100)
fit1 = multiview(list(x=x,z=z), y, rho = 0)
coef(fit1, s=0.1)

# Binomial
by = sample(c(0,1), 100, replace = TRUE)
fit2 = multiview(list(x=x,z=z), by, family = binomial(), rho=0.5)
coef(fit2, s=0.1)

# Poisson
py = matrix(rpois(100, exp(y)))
```

```
fit3 = multiview(list(x=x,z=z), py, family = poisson(), rho=0.5)
coef(fit3, s=0.1)
```

`coef_ordered` *Extract an ordered list of standardized coefficients from a* `multiview` *or* `cv.multiview` *object*

Description

This function extracts a ranked list of coefficients after the coefficients are standardized by the standard deviation of the corresponding features. The ranking is based on the magnitude of the standardized coefficients. It also outputs the data view to which each coefficient belongs.

Usage

```
coef_ordered(object, ...)
```

Arguments

`object`	Fitted "`multiview`" or "`cv.multiview`" object. coefficients are required.
`...`	This is the mechanism for passing arguments like `x=` when `exact=TRUE`; see `exact` argument.

Details

The output table shows from left to right the data view each coefficient comes from, the column index of the feature in the corresponding data view, the coefficient after being standardized by the standard deviation of the corresponding feature, and the original fitted coefficient.

Value

data frame of consisting of view name, view column, coefficient and standardized coefficient ordered by rank of standardized coefficient.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
z = matrix(rnorm(100 * 10), 100, 10)
y = rnorm(100)
fit1 = multiview(list(x=x,z=z), y, rho = 0)
coef_ordered(fit1, s=0.1)

# Binomial
by = sample(c(0,1), 100, replace = TRUE)
fit2 = multiview(list(x=x,z=z), by, family = binomial(), rho=0.5)
coef_ordered(fit2, s=0.1)
```

```
# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = multiview(list(x=x,z=z), py, family = poisson(), rho=0.5)
coef_ordered(fit3, s=0.1)
```

`coef_ordered.cv.multiview` *Extract an ordered list of standardized coefficients from a cv.multiview object*

Description

This function extracts a ranked list of coefficients after the coefficients are standardized by the standard deviation of the corresponding features. The ranking is based on the magnitude of the standardized coefficients. It also outputs the data view to which each coefficient belongs.

Usage

```
## S3 method for class 'cv.multiview'
coef_ordered(object, s = c("lambda.1se", "lambda.min"), ...)
```

Arguments

`object`	Fitted `"cv.multiview"` object.
`s`	Value(s) of the penalty parameter `lambda` at which predictions are required. Default is the value `s="lambda.1se"` stored on the CV `object`. Alternatively `s="lambda.min"` can be used. If `s` is numeric, it is taken as the value(s) of `lambda` to be used. (For historical reasons we use the symbol 's' rather than 'lambda' to reference this parameter.)
`...`	This is the mechanism for passing arguments like `x=` when `exact=TRUE`; see `exact` argument.

Details

The output table shows from left to right the data view each coefficient comes from, the column index of the feature in the corresponding data view, the coefficient after being standardized by the standard deviation of the corresponding feature, and the original fitted coefficient.

Value

data frame of consisting of view name, view column, coefficient and standardized coefficient ordered by rank of standardized coefficient.

Examples

```
set.seed(1)
x = matrix(rnorm(100*20), 100, 20)
z = matrix(rnorm(100*20), 100, 20)
U = matrix(rnorm(100*5), 100, 5)
for (m in seq(5)){
    u = rnorm(100)
    x[, m] = x[, m] + u
    z[, m] = z[, m] + u
    U[, m] = U[, m] + u}
x = scale(x, center = TRUE, scale = FALSE)
z = scale(z, center = TRUE, scale = FALSE)
beta_U = c(rep(0.1, 5))
y = U %*% beta_U + 0.1 * rnorm(100)
fit1 = cv.multiview(list(x=x,z=z), y, rho = 0.3)
coef_ordered(fit1, s="lambda.min")

# Binomial

by = 1 * (y > median(y))
fit2 = cv.multiview(list(x=x,z=z), by, family = binomial(), rho = 0.9)
coef_ordered(fit2, s="lambda.min")

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = cv.multiview(list(x=x,z=z), py, family = poisson(), rho = 0.6)
coef_ordered(fit3, s="lambda.min")
```

`coef_ordered.multiview` *Extract an ordered list of standardized coefficients from a multiview object*

Description

This function extracts a ranked list of coefficients after the coefficients are standardized by the standard deviation of the corresponding features. The ranking is based on the magnitude of the standardized coefficients. It also outputs the data view to which each coefficient belongs.

Usage

```
## S3 method for class 'multiview'
coef_ordered(object, s = NULL, ...)
```

Arguments

`object`	Fitted "`multiview`" object.
`s`	Value(s) of the penalty parameter `lambda` at which coefficients are required.

...	This is the mechanism for passing arguments like x= when exact=TRUE; see exact argument.

Details

The output table shows from left to right the data view each coefficient comes from, the column index of the feature in the corresponding data view, the coefficient after being standardized by the standard deviation of the corresponding feature, and the original fitted coefficient.

Value

data frame of consisting of view name, view column, coefficient and standardized coefficient ordered by rank of standardized coefficient.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
z = matrix(rnorm(100 * 10), 100, 10)
y = rnorm(100)
fit1 = multiview(list(x=x,z=z), y, rho = 0)
coef_ordered(fit1, s=0.1)

# Binomial
by = sample(c(0,1), 100, replace = TRUE)
fit2 = multiview(list(x=x,z=z), by, family = binomial(), rho=0.5)
coef_ordered(fit2, s=0.1)

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = multiview(list(x=x,z=z), py, family = poisson(), rho=0.5)
coef_ordered(fit3, s=0.1)
```

---

collapse_named_lists *Collapse a list of named lists into one list with the same name*

Description

Collapse a list of named lists into one list with the same name

Usage

```
collapse_named_lists(in_list)
```

Arguments

in_list	a list of named lists all with same names (not checked for efficiency)

Value

a single list with named components all concatenated

`cv.multiview` *Perform k-fold cross-validation for cooperative learning*

Description

Does k-fold cross-validation (CV) for multiview and produces a CV curve.

Usage

```
cv.multiview(
  x_list,
  y,
  family = gaussian(),
  rho = 0,
  weights = NULL,
  offset = NULL,
  mvlambda = NULL,
  type.measure = c("default", "mse", "deviance", "class", "auc", "mae", "C"),
  nfolds = 10,
  foldid = NULL,
  alignment = c("lambda", "fraction"),
  grouped = TRUE,
  keep = FALSE,
  trace.it = 0,
  ...
)
```

Arguments

`x_list`	a list of `x` matrices with same number of rows `nobs`
`y`	the quantitative response with length equal to `nobs`, the (same) number of rows in each `x` matrix
`family`	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. Default is stats::gaussian. (See stats::family for details on family functions.)
`rho`	the weight on the agreement penalty, default 0. `rho=0` is a form of early fusion, and `rho=1` is a form of late fusion. We recommend trying a few values of `rho` including 0, 0.1, 0.25, 0.5, and 1 first; sometimes `rho` larger than 1 can also be helpful.
`weights`	Observation weights; defaults to 1 per observation
`offset`	Offset vector (matrix) as in `multiview`

`mvlambda` A user supplied `lambda` sequence, default `NULL`. Typical usage is to have the program compute its own `mvlambda` sequence. This sequence, in general, is different from that used in the `glmnet::glmnet()` call (named `lambda`). Note that this is done for the full model (master sequence), and separately for each fold. The fits are then aligned using the glmnet lambda sequence associated with the master sequence (see the `alignment` argument for additional details). Adapting `mvlambda` for each fold leads to better convergence. When `mvlambda` is supplied, the same sequence is used everywhere, but in some GLMs can lead to convergence issues.

`type.measure` loss to use for cross-validation. Currently five options, not all available for all models. The default is `type.measure="deviance"`, which uses squared-error for gaussian models (a.k.a `type.measure="mse"` there), deviance for logistic and poisson regression, and partial-likelihood for the Cox model. `type.measure="class"` applies to binomial and multinomial logistic regression only, and gives misclassification error. `type.measure="auc"` is for two-class logistic regression only, and gives area under the ROC curve. `type.measure="mse"` or `type.measure="mae"` (mean absolute error) can be used by all models except the `"cox"`; they measure the deviation from the fitted mean to the response. `type.measure="C"` is Harrel's concordance measure, only available for `cox` models.

`nfolds` number of folds - default is 10. Although `nfolds` can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is `nfolds=3`

`foldid` an optional vector of values between 1 and `nfold` identifying what fold each observation is in. If supplied, `nfold` can be missing.

`alignment` This is an experimental argument, designed to fix the problems users were having with CV, with possible values `"lambda"` (the default) else `"fraction"`. With `"lambda"` the `lambda` values from the master fit (on all the data) are used to line up the predictions from each of the folds. In some cases this can give strange values, since the effective `lambda` values in each fold could be quite different. With `"fraction"` we line up the predictions in each fold according to the fraction of progress along the regularization. If in the call a `lambda` argument is also provided, `alignment="fraction"` is ignored (with a warning).

`grouped` This is an experimental argument, with default `TRUE`, and can be ignored by most users. For all models except the `"cox"`, this refers to computing `nfolds` separate statistics, and then using their mean and estimated standard error to describe the CV curve. If `grouped=FALSE`, an error matrix is built up at the observation level from the predictions from the `nfold` fits, and then summarized (does not apply to `type.measure="auc"`). For the `"cox"` family, `grouped=TRUE` obtains the CV partial likelihood for the Kth fold by *subtraction*; by subtracting the log partial likelihood evaluated on the full dataset from that evaluated on the on the (K-1)/K dataset. This makes more efficient use of risk sets. With `grouped=FALSE` the log partial likelihood is computed only on the Kth fold

`keep` If `keep=TRUE`, a *prevalidated* array is returned containing fitted values for each observation and each value of `lambda`. This means these fits are computed with this observation and the rest of its fold omitted. The `foldid` vector is also returned. Default is keep=FALSE.

trace.it	If trace.it=1, then progress bars are displayed; useful for big models that take a long time to fit.
...	Other arguments that can be passed to multiview

Details

The current code can be slow for "large" data sets, e.g. when the number of features is larger than 1000. It can be helpful to see the progress of multiview as it runs; to do this, set trace.it = 1 in the call to multiview or cv.multiview. With this, multiview prints out its progress along the way. One can also pre-filter the features to a smaller set, using the exclude option, with a filter function.

If there are missing values in the feature matrices: we recommend that you center the columns of each feature matrix, and then fill in the missing values with 0.

For example,

```
x <- scale(x,TRUE,FALSE)
x[is.na(x)] <- 0
z <- scale(z,TRUE,FALSE)
z[is.na(z)] <- 0
```

Then run multiview in the usual way. It will exploit the assumed shared latent factors to make efficient use of the available data.

The function runs multiview nfolds+1 times; the first to get the lambda sequence, and then the remainder to compute the fit with each of the folds omitted. The error is accumulated, and the average error and standard deviation over the folds is computed. Note that cv.multiview does NOT search for values for rho. A specific value should be supplied, else rho=0 is assumed by default. If users would like to cross-validate rho as well, they should call cv.multiview with a pre-computed vector foldid, and then use this same fold vector in separate calls to cv.multiview with different values of rho.

Value

an object of class "cv.multiview" is returned, which is a list with the ingredients of the cross-validation fit.

lambda	the values of lambda used in the fits.
cvm	The mean cross-validated error - a vector of length length(lambda).
cvsd	estimate of standard error of cvm.
cvup	upper curve = cvm+cvsd.
cvlo	lower curve = cvm-cvsd.
nzero	number of non-zero coefficients at each lambda.
name	a text string indicating type of measure (for plotting purposes).
multiview.fit	a fitted multiview object for the full data.
lambda.min	value of lambda that gives minimum cvm.
lambda.1se	largest value of lambda such that error is within 1 standard error of the minimum.

fit.preval	if keep=TRUE, this is the array of prevalidated fits. Some entries can be NA, if that and subsequent values of lambda are not reached for that fold
foldid	if keep=TRUE, the fold assignments used
index	a one column matrix with the indices of lambda.min and lambda.1se in the sequence of coefficients, fits etc.

Examples

```
# Gaussian
# Generate data based on a factor model
set.seed(1)
x = matrix(rnorm(100*20), 100, 20)
z = matrix(rnorm(100*20), 100, 20)
U = matrix(rnorm(100*5), 100, 5)
for (m in seq(5)){
    u = rnorm(100)
    x[, m] = x[, m] + u
    z[, m] = z[, m] + u
    U[, m] = U[, m] + u}
x = scale(x, center = TRUE, scale = FALSE)
z = scale(z, center = TRUE, scale = FALSE)
beta_U = c(rep(0.1, 5))
y = U %*% beta_U + 0.1 * rnorm(100)
fit1 = cv.multiview(list(x=x,z=z), y, rho = 0.3)

# plot the cross-validation curve
plot(fit1)

# extract coefficients
coef(fit1, s="lambda.min")

# extract ordered coefficients
coef_ordered(fit1, s="lambda.min")

# make predictions
predict(fit1, newx = list(x[1:5, ],z[1:5,]), s = "lambda.min")

# Binomial

by = 1 * (y > median(y))
fit2 = cv.multiview(list(x=x,z=z), by, family = binomial(), rho = 0.9)
predict(fit2, newx = list(x[1:5, ],z[1:5,]), s = "lambda.min", type = "response")
plot(fit2)
coef(fit2, s="lambda.min")
coef_ordered(fit2, s="lambda.min")

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = cv.multiview(list(x=x,z=z), py, family = poisson(), rho = 0.6)
predict(fit3, newx = list(x[1:5, ],z[1:5,]), s = "lambda.min", type = "response")
plot(fit3)
coef(fit3, s="lambda.min")
```

```
coef_ordered(fit3, s="lambda.min")
```

`dev_function` *Elastic net deviance value*

Description

Returns the elastic net deviance value.

Usage

```
dev_function(y, mu, weights, family)
```

Arguments

`y`	Quantitative response variable.
`mu`	Model's predictions for `y`.
`weights`	Observation weights.
`family`	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function.

`elnet.fit` *Solve weighted least squares (WLS) problem for a single lambda value*

Description

Solves the weighted least squares (WLS) problem for a single lambda value. Internal function that users should not call directly.

Usage

```
elnet.fit(
  x,
  y,
  weights,
  lambda,
  alpha = 1,
  intercept = TRUE,
  thresh = 1e-07,
  maxit = 1e+05,
  penalty.factor = rep(1, nvars),
  exclude = c(),
  lower.limits = -Inf,
  upper.limits = Inf,
```

```
  warm = NULL,
  from.glmnet.fit = FALSE,
  save.fit = FALSE
)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector. If it is a sparse matrix, it is assumed to be unstandardized. It should have attributes xm and xs, where xm(j) and xs(j) are the centering and scaling factors for variable j respsectively. If it is not a sparse matrix, it is assumed that any standardization needed has already been done.
y	Quantitative response variable.
weights	Observation weights. elnet.fit does NOT standardize these weights.
lambda	A single value for the lambda hyperparameter.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2\|\|\beta\|\|_2^2 + \alpha\|\|\beta\|\|_1.$$ alpha=1 is the lasso penalty, and alpha=0 the ridge penalty.
intercept	Should intercept be fitted (default=TRUE) or set to zero (FALSE)?
thresh	Convergence threshold for coordinate descent. Each inner coordinate-descent loop continues until the maximum change in the objective after any coefficient update is less than thresh times the null deviance. Default value is 1e-7.
maxit	Maximum number of passes over the data; default is 10^5. (If a warm start object is provided, the number of passes the warm start object performed is included.)
penalty.factor	Separate penalty factors can be applied to each coefficient. This is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in exclude). Note: the penalty factors are internally rescaled to sum to nvars.
exclude	Indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor.
lower.limits	Vector of lower limits for each coefficient; default -Inf. Each of these must be non-positive. Can be presented as a single value (which will then be replicated), else a vector of length nvars.
upper.limits	Vector of upper limits for each coefficient; default Inf. See lower.limits.
warm	Either a glmnetfit object or a list (with names beta and a0 containing coefficients and intercept respectively) which can be used as a warm start. Default is NULL, indicating no warm start. For internal use only.
from.glmnet.fit	Was elnet.fit() called from glmnet.fit()? Default is FALSE.This has implications for computation of the penalty factors.
save.fit	Return the warm start object? Default is FALSE.

Details

WARNING: Users should not call `elnet.fit` directly. Higher-level functions in this package call `elnet.fit` as a subroutine. If a warm start object is provided, some of the other arguments in the function may be overriden.

`elnet.fit` is essentially a wrapper around a C++ subroutine which minimizes

$$1/2 \sum w_i(y_i - X_i^T \beta)^2 + \sum \lambda \gamma_j [(1-\alpha)/2\beta^2 + \alpha|\beta|],$$

over β, where γ_j is the relative penalty factor on the jth variable. If `intercept = TRUE`, then the term in the first sum is $w_i(y_i - \beta_0 - X_i^T \beta)^2$, and we are minimizing over both β_0 and β.

None of the inputs are standardized except for `penalty.factor`, which is standardized so that they sum up to `nvars`.

Value

An object with class "glmnetfit" and "glmnet". The list returned has the same keys as that of a `glmnet` object, except that it might have an additional `warm_fit` key.

`a0`	Intercept value.
`beta`	A `nvars x 1` matrix of coefficients, stored in sparse matrix format.
`df`	The number of nonzero coefficients.
`dim`	Dimension of coefficient matrix.
`lambda`	Lambda value used.
`dev.ratio`	The fraction of (null) deviance explained. The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.
`nulldev`	Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)). The null model refers to the intercept model.
`npasses`	Total passes over the data.
`jerr`	Error flag, for warnings and errors (largely for internal debugging).
`offset`	Always FALSE, since offsets do not appear in the WLS problem. Included for compability with glmnet output.
`call`	The call that produced this object.
`nobs`	Number of observations.
`warm_fit`	If `save.fit=TRUE`, output of C++ routine, used for warm starts. For internal use only.

get_eta *Helper function to get etas (linear predictions)*

Description

Given x, coefficients and intercept, return linear predictions. Wrapper that works with both regular and sparse x. Only works for single set of coefficients and intercept.

Usage

```
get_eta(x, beta, a0)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector. If it is a sparse matrix, it is assumed to be unstandardized. It should have attributes xm and xs, where xm(j) and xs(j) are the centering and scaling factors for variable j respsectively. If it is not a sparse matrix, it is assumed to be standardized.
beta	Feature coefficients.
a0	Intercept.

get_start *Get null deviance, starting mu and lambda max*

Description

Return the null deviance, starting mu and lambda max values for initialization. For internal use only.

Usage

```
get_start(
  x,
  y,
  weights,
  family,
  intercept,
  is.offset,
  offset,
  exclude,
  vp,
  alpha
)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector. If it is a sparse matrix, it is assumed to be unstandardized. It should have attributes xm and xs, where xm(j) and xs(j) are the centering and scaling factors for variable j respsectively. If it is not a sparse matrix, it is assumed to be standardized.
y	Quantitative response variable.
weights	Observation weights.
family	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. (See family for details on family functions.)
intercept	Does the model we are fitting have an intercept term or not?
is.offset	Is the model being fit with an offset or not?
offset	Offset for the model. If is.offset=FALSE, this should be a zero vector of the same length as y.
exclude	Indices of variables to be excluded from the model.
vp	Separate penalty factors can be applied to each coefficient.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$.

Details

This function is called by glmnet.path for null deviance, starting mu and lambda max values. It is also called by glmnet.fit when used without warmstart, but they only use the null deviance and starting mu values.

When x is not sparse, it is expected to already by centered and scaled. When x is sparse, the function will get its attributes xm and xs for its centering and scaling factors.

Note that whether x is centered & scaled or not, the values of mu and nulldev don't change. However, the value of lambda_max does change, and we need xm and xs to get the correct value.

---

make_row *Build a block row matrix for multiview*

Description

Build a block row matrix for multiview

Usage

```
make_row(x_list, p_x, pair, rho)
```

Arguments

x_list	list of x matrices
p_x	a list of ncol of elements in x_list
pair	an integer vector of two indices
rho	the rho value

Value

a block row of matrix for multiview

`multiview` *Perform cooperative learning using the direct algorithm for two or more views.*

Description

`multiview` uses `glmnet::glmnet()` to do most of its work and therefore takes many of the same parameters, but an intercept is always included, standardization is always done and several other parameters do not apply. Therefore they are always overridden and warnings issued.

Usage

```
multiview(
  x_list,
  y,
  rho = 0,
  family = gaussian(),
  exclude = NULL,
  mvlambda = NULL,
  ...
)
```

Arguments

`x_list`	a list of `x` matrices with same number of rows `nobs`
`y`	the quantitative response with length equal to `nobs`, the (same) number of rows in each `x` matrix
`rho`	the weight on the agreement penalty, default 0. `rho=0` is a form of early fusion, and `rho=1` is a form of late fusion. We recommend trying a few values of `rho` including 0, 0.1, 0.25, 0.5, and 1 first; sometimes `rho` larger than 1 can also be helpful.
`family`	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. Default is stats::gaussian. (See stats::family for details on family functions.)
`exclude`	Indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor for the variables excluded (next item). Users can supply instead an `exclude` function that generates the list of indices. This function is most generally defined as `function(x_list, y, ...)`, and is called inside `multiview` to generate the indices for excluded variables. The `...` argument is required, the others are optional. This is useful for filtering wide data, and works correctly with `cv.glmnet`.

mvlambda	A user supplied lambda sequence, default NULL. Typical usage is to have the program compute its own mvlambda sequence. This sequence, in general, is different from that used in the glmnet::glmnet() call (named lambda) Supplying a value of mvlambda overrides this. WARNING: use with care. Avoid supplying a single value for mvlambda (for predictions after CV use stats::predict() instead. Supply instead a decreasing sequence of mvlambda values as multiview relies on its warms starts for speed, and its often faster to fit a whole path than compute a single fit.
...	further arguments to glmnet, some of which may be overridden as noted above

Details

The current code can be slow for "large" data sets, e.g. when the number of features is larger than 1000. It can be helpful to see the progress of multiview as it runs; to do this, set trace.it = 1 in the call to multiview or cv.multiview. With this, multiview prints out its progress along the way. One can also pre-filter the features to a smaller set, using the exclude option, with a filter function.

If there are missing values in the feature matrices: we recommend that you center the columns of each feature matrix, and then fill in the missing values with 0.

For example,

```
x <- scale(x,TRUE,FALSE)
x[is.na(x)] <- 0
z <- scale(z,TRUE,FALSE)
z[is.na(z)] <- 0
```

Then run multiview in the usual way. It will exploit the assumed shared latent factors to make efficient use of the available data.

Value

An object with S3 class "multiview","*" , where "*" is "elnet", "lognet", "multnet", "fishnet" (poisson), "coxnet" or "mrelnet" for the various types of models.

call	the call that produced this object
a0	Intercept sequence of length length(lambda)
beta	For "elnet", "lognet", "fishnet" and "coxnet" models, a nvars x length(lambda) matrix of coefficients, stored in sparse column format ("CsparseMatrix"). For "multnet" and "mgaussian", a list of nc such matrices, one for each class.
lambda	The actual sequence of glmnet::glmnet() lambda values used. When alpha=0, the largest lambda reported does not quite give the zero coefficients reported (lambda=inf would in principle). Instead, the largest lambda for alpha=0.001 is used, and the sequence of lambda values is derived from this.
mvlambda	The corresponding sequence of multiview lambda values
dev.ratio	The fraction of (null) deviance explained (for "elnet", this is the R-square). The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.

nulldev	Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)); The NULL model refers to the intercept model, except for the Cox, where it is the 0 model.
df	The number of nonzero coefficients for each value of lambda. For "multnet", this is the number of variables with a nonzero coefficient for *any* class.
dfmat	For "multnet" and "mrelnet" only. A matrix consisting of the number of nonzero coefficients per class
dim	dimension of coefficient matrix (ices)
nobs	number of observations
npasses	total passes over the data summed over all lambda values
offset	a logical variable indicating whether an offset was included in the model
jerr	error flag, for warnings and errors (largely for internal debugging).

See Also

print, coef, coef_ordered, predict, and plot methods for "multiview", and the "cv.multiview" function.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
z = matrix(rnorm(100 * 10), 100, 10)
y = rnorm(100)
fit1 = multiview(list(x=x,z=z), y, rho = 0)
print(fit1)

# extract coefficients at a single value of lambda
coef(fit1, s = 0.01)

# extract ordered (standardized) coefficients at a single value of lambda
coef_ordered(fit1, s = 0.01)

# make predictions
predict(fit1, newx = list(x[1:10, ],z[1:10, ]), s = c(0.01, 0.005))

# make a path plot of features for the fit
plot(fit1, label=TRUE)

# Binomial
by = sample(c(0,1), 100, replace = TRUE)
fit2 = multiview(list(x=x,z=z), by, family = binomial(), rho=0.5)
predict(fit2, newx = list(x[1:10, ],z[1:10, ]), s = c(0.01, 0.005), type="response")
coef_ordered(fit2, s = 0.01)
plot(fit2, label=TRUE)

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = multiview(list(x=x,z=z), py, family = poisson(), rho=0.5)
```

```
predict(fit3, newx = list(x[1:10, ],z[1:10, ]), s = c(0.01, 0.005), type="response")
coef_ordered(fit3, s = 0.01)
plot(fit3, label=TRUE)
```

`multiview.control` *Internal multiview parameters*

Description

View and/or change the factory default parameters in multiview

Usage

```
multiview.control(
  fdev = 1e-05,
  devmax = 0.999,
  eps = 1e-06,
  big = 9.9e+35,
  mnlam = 5,
  pmin = 1e-09,
  exmx = 250,
  prec = 1e-10,
  mxit = 100,
  itrace = 0,
  epsnr = 1e-06,
  mxitnr = 25,
  factory = FALSE
)
```

Arguments

`fdev`	minimum fractional change in deviance for stopping path; factory default = 1.0e-5
`devmax`	maximum fraction of explained deviance for stopping path; factory default = 0.999
`eps`	minimum value of lambda.min.ratio (see multiview); factory default= 1.0e-6
`big`	large floating point number; factory default = 9.9e35. Inf in definition of upper.limit is set to big
`mnlam`	minimum number of path points (lambda values) allowed; factory default = 5
`pmin`	minimum probability for any class. factory default = 1.0e-9. Note that this implies a pmax of 1-pmin.
`exmx`	maximum allowed exponent. factory default = 250.0
`prec`	convergence threshold for multi response bounds adjustment solution. factory default = 1.0e-10

mxit	maximum iterations for multiresponse bounds adjustment solution. factory default = 100
itrace	If 1 then progress bar is displayed when running multiview and cv.multiview. factory default = 0
epsnr	convergence threshold for multiview.fit. factory default = 1.0e-6
mxitnr	maximum iterations for the IRLS loop in multiview.fit. factory default = 25
factory	If TRUE, reset all the parameters to the factory default; default is FALSE

Details

If called with no arguments, multiview.control() returns a list with the current settings of these parameters. Any arguments included in the call sets those parameters to the new values, and then silently returns. The values set are persistent for the duration of the R session.

Value

A list with named elements as in the argument list

See Also

multiview

Examples

```
multiview.control(fdev = 0)  #continue along path even though not much changes
multiview.control()  # view current settings
multiview.control(factory = TRUE)  # reset all the parameters to their default
```

---

multiview.fit *Fit a GLM with elastic net regularization for a single value of lambda*

Description

Fit a generalized linear model via penalized maximum likelihood for a single value of lambda. Can deal with any GLM family.

Usage

```
multiview.fit(
  x_list,
  x,
  y,
  rho,
  weights,
  lambda,
```

```
  alpha = 1,
  offset = rep(0, nobs),
  family = gaussian(),
  intercept = TRUE,
  thresh = 1e-07,
  maxit = 1e+05,
  penalty.factor = rep(1, nvars),
  exclude = c(),
  lower.limits = -Inf,
  upper.limits = Inf,
  warm = NULL,
  from.multiview.path = FALSE,
  save.fit = FALSE,
  trace.it = 0
)
```

Arguments

x_list	a list of x matrices with same number of rows nobs
x	the column-binded entries of x_list
y	the quantitative response with length equal to nobs, the (same) number of rows in each x matrix
rho	the weight on the agreement penalty, default 0. rho=0 is a form of early fusion, and rho=1 is a form of late fusion.
weights	observation weights. Can be total counts if responses are proportion matrices. Default is 1 for each observation
lambda	A single value for the lambda hyperparameter.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2\|\|\beta\|\|_2^2 + \alpha\|\|\beta\|\|_1.$$ alpha = 1 is the lasso penalty, and alpha = 0 the ridge penalty. lambda.min.ratio. Supplying a value of lambda overrides this.
offset	A vector of length nobs that is included in the linear predictor (a nobs by nc matrix for the "multinomial" family). Useful for the "poisson" family (e.g. log of exposure time), or for refining a model by starting at a current fit. Default is NULL. If supplied, then values must also be supplied to the predict function.
family	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. Default is stats::gaussian. (See stats::family for details on family functions.)
intercept	Should intercept be fitted (default TRUE) or set to zero (FALSE)?
thresh	Convergence threshold for coordinate descent. Each inner coordinate-descent loop continues until the maximum change in the objective after any coefficient update is less than thresh times the null deviance. Default value is 1e-7.
maxit	Maximum number of passes over the data; default is 10^5. (If a warm start object is provided, the number of passes the warm start object performed is included.)

`penalty.factor` List of separate penalty factors can be applied to each coefficient, consisting of `ncol(x)` elements for each `x` in `x_list`. This is a number that multiplies `lambda` to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in `exclude`). Note: the penalty factors are internally rescaled to sum to `nvars`.

`exclude` List of column indices of `x` matrices in `x_list` to be excluded from the model. Default is empty list. Equivalent to an infinite penalty factor for the variables excluded (next item). Users can supply instead an `exclude()` function that generates the list of `x` columns indices. This function is most generally defined as `function(x_list, y, weights, ...)`, and is called inside `multiview()` to generate the indices for excluded variables. The `...` argument is required, the others are optional. This is useful for filtering wide data, and works correctly with `cv.multiview`. See the vignette 'Introduction' for examples.

`lower.limits` Vector of lower limits for each coefficient consisting of `ncol(x)` elements for each xinx_list; defaul Inf'. Each of these must be non-positive.

`upper.limits` Vector of upper limits for each coefficient consisting of `ncol(x)` elements for each xinx_list; defaul Each of these must be non-negative.

`warm` Either a `multiview` object or a list (with names `beta` and `a0` containing coefficients and intercept respectively) which can be used as a warm start. Default is `NULL`, indicating no warm start. For internal use only.

`from.multiview.path` Was `multiview.fit()` called from `multiview.path()`? Default is `FALSE`.This has implications for computation of the penalty factors.

`save.fit` Return the warm start object? Default is `FALSE`.

`trace.it` Controls how much information is printed to screen. If `trace.it = 2`, some information about the fitting procedure is printed to the console as the model is being fitted. Default is `trace.it = 0` (no information printed). (`trace.it = 1` not used for compatibility with `multiview.path`.)

Details

WARNING: Users should not call `multiview.fit` directly. Higher-level functions in this package call `multiview.fit` as a subroutine. If a warm start object is provided, some of the other arguments in the function may be overriden.

`multiview.fit` solves the elastic net problem for a *single, user-specified* value of lambda. `multiview.fit` works for any GLM family. It solves the problem using iteratively reweighted least squares (IRLS). For each IRLS iteration, `multiview.fit` makes a quadratic (Newton) approximation of the log-likelihood, then calls `elnet.fit` to minimize the resulting approximation.

In terms of standardization: `multiview.fit` does not standardize `x` and `weights`. `penalty.factor` is standardized so that to sum to `nvars`.

Value

An object with class `"multiview"`. The list returned contains more keys than that of a `"multiview"` object.

a0	Intercept value.
beta	A nvars by 1 matrix of coefficients, stored in sparse matrix format.
df	The number of nonzero coefficients.
dim	Dimension of coefficient matrix.
lambda	Lambda value used.
lambda_scale	The multiview lambda scale factor
dev.ratio	The fraction of (null) deviance explained. The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.
nulldev	Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)). The null model refers to the intercept model.
npasses	Total passes over the data.
jerr	Error flag, for warnings and errors (largely for internal debugging).
offset	A logical variable indicating whether an offset was included in the model.
call	The call that produced this object.
nobs	Number of observations.
warm_fit	If save.fit = TRUE, output of C++ routine, used for warm starts. For internal use only.
family	Family used for the model.
converged	A logical variable: was the algorithm judged to have converged?
boundary	A logical variable: is the fitted value on the boundary of the attainable values?
obj_function	Objective function value at the solution.

`multiview.path` *Fit a GLM with elastic net regularization for a path of lambda values*

Description

Fit a generalized linear model via penalized maximum likelihood for a path of lambda values. Can deal with any GLM family.

Usage

```
multiview.path(
  x_list,
  y,
  rho = 0,
  weights = NULL,
  mvlambda = NULL,
  nlambda = 100,
  lambda.min.ratio = ifelse(nobs < nvars, 0.01, 1e-04),
```

```
  alpha = 1,
  offset = NULL,
  family = gaussian(),
  standardize = TRUE,
  intercept = TRUE,
  thresh = 1e-07,
  maxit = 1e+05,
  penalty.factor = rep(1, nvars),
  exclude = integer(0),
  lower.limits = -Inf,
  upper.limits = Inf,
  trace.it = 0,
  x
)
```

Arguments

x_list	a list of x matrices with same number of rows nobs
y	the quantitative response with length equal to nobs, the (same) number of rows in each x matrix
rho	the weight on the agreement penalty, default 0. rho=0 is a form of early fusion, and rho=1 is a form of late fusion.
weights	observation weights. Can be total counts if responses are proportion matrices. Default is 1 for each observation
mvlambda	A user supplied lambda sequence. Typical usage is to have the program compute its own lambda sequence based on nlambda and lambda.min.ratio. Supplying a value of lambda overrides this. WARNING: use with care. Avoid supplying a single value for lambda (for predictions after CV use predict() instead). Supply instead a decreasing sequence of lambda values. glmnet relies on its warms starts for speed, and its often faster to fit a whole path than compute a single fit.
nlambda	The number of lambda values, default is 100.
lambda.min.ratio	Smallest value for lambda as a fraction of lambda.max, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size nobs relative to the number of variables nvars. If nobs >= nvars, the default is 0.0001, close to zero. If nobs < nvars, the default is 0.01. A very small value of lambda.min.ratio will lead to a saturated fit in the nobs < nvars case. This is undefined for some families of models, and the function will exit gracefully when the percentage deviance explained is almost 1.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2\|\|\beta\|\|_2^2 + \alpha\|\|\beta\|\|_1.$$ alpha = 1 is the lasso penalty, and alpha = 0 the ridge penalty. lambda.min.ratio. Supplying a value of lambda overrides this.

offset	A vector of length nobs that is included in the linear predictor (a nobs by nc matrix for the "multinomial" family). Useful for the "poisson" family (e.g. log of exposure time), or for refining a model by starting at a current fit. Default is NULL. If supplied, then values must also be supplied to the predict function.
family	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. Default is stats::gaussian. (See stats::family for details on family functions.)
standardize	Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize = TRUE. If variables are in the same units already, you might not wish to standardize.
intercept	Should intercept be fitted (default TRUE) or set to zero (FALSE)?
thresh	Convergence threshold for coordinate descent. Each inner coordinate-descent loop continues until the maximum change in the objective after any coefficient update is less than thresh times the null deviance. Default value is 1e-7.
maxit	Maximum number of passes over the data for all lambda values; default is 10^5.
penalty.factor	List of separate penalty factors can be applied to each coefficient, consisting of ncol(x) elements for each x in x_list. This is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in exclude). Note: the penalty factors are internally rescaled to sum to nvars.
exclude	List of column indices of x matrices in x_list to be excluded from the model. Default is empty list. Equivalent to an infinite penalty factor for the variables excluded (next item). Users can supply instead an exclude() function that generates the list of x columns indices. This function is most generally defined as function(x_list, y, weights, ...), and is called inside multiview() to generate the indices for excluded variables. The ... argument is required, the others are optional. This is useful for filtering wide data, and works correctly with cv.multiview. See the vignette 'Introduction' for examples.
lower.limits	Vector of lower limits for each coefficient consisting of ncol(x) elements for each xinx_list; defaul Inf'. Each of these must be non-positive.
upper.limits	Vector of upper limits for each coefficient consisting of ncol(x) elements for each xinx_list; defaul Each of these must be non-negative.
trace.it	If trace.it = 1, then a progress bar is displayed; useful for big models that take a long time to fit.
x	the cbinded matrices in x_list

Details

multiview.path solves the elastic net problem for a path of lambda values. It generalizes multiview::multiview in that it works for any GLM family.

Sometimes the sequence is truncated before nlambda values of lambda have been used. This happens when multiview.path detects that the decrease in deviance is marginal (i.e. we are near a saturated fit).

Value

An object with class "multiview" "glmnetfit" and "glmnet"

a0	Intercept sequence of length length(lambda).
beta	A nvars x length(lambda) matrix of coefficients, stored in sparse matrix format.
df	The number of nonzero coefficients for each value of lambda.
dim	Dimension of coefficient matrix.
lambda	The actual sequence of lambda values used. When alpha=0, the largest lambda reported does not quite give the zero coefficients reported (lambda=inf would in principle). Instead, the largest lambda for alpha=0.001 is used, and the sequence of lambda values is derived from this.
mvlambda	The corresponding sequence of multiview lambda values
dev.ratio	The fraction of (null) deviance explained. The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.
nulldev	Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)). The null model refers to the intercept model.
npasses	Total passes over the data summed over all lambda values.
jerr	Error flag, for warnings and errors (largely for internal debugging).
offset	A logical variable indicating whether an offset was included in the model.
call	The call that produced this object.
family	Family used for the model.
nobs	Number of observations.

---

obj_function *Elastic net objective function value*

Description

Returns the elastic net objective function value.

Usage

```
obj_function(
  y,
  mu,
  weights,
  family,
  lambda,
  alpha,
  coefficients,
```

```
  vp,
  view_components,
  rho
)
```

Arguments

y	Quantitative response variable.
mu	Model's predictions for y.
weights	Observation weights.
family	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function.
lambda	A single value for the lambda hyperparameter.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$.
coefficients	The model's coefficients (excluding intercept).
vp	Penalty factors for each of the coefficients.
view_components	a list of lists containing indices of coefficients and associated covariate (view) pairs
rho	the fusion parameter

---

pen_function *Elastic net penalty value*

Description

Returns the elastic net penalty value without the lambda factor.

Usage

```
pen_function(coefficients, alpha = 1, vp = 1)
```

Arguments

coefficients	The model's coefficients (excluding intercept).
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$.
vp	Penalty factors for each of the coefficients.

Details

The penalty is defined as

$$(1-\alpha)/2 \sum vp_j \beta_j^2 + \alpha \sum vp_j |\beta|.$$

Note the omission of the multiplicative lambda factor.

plot.multiview *Plot coefficients from a "multiview" object*

Description

Produces a coefficient profile plot of the coefficient paths for a fitted "multiview" object. The paths are colored by the data views, from which the features come.

Usage

```
## S3 method for class 'multiview'
plot(x, col_palette = NULL, label = FALSE, ...)
```

Arguments

x	A fitted "multiview" model.
col_palette	A set of colors to use for indicating different views. If NULL, the function will use the color palette "Set1" from the RColorBrewer package.
label	If TRUE, label the curves with variable sequence. numbers.
...	Other graphical parameters to plot.

Value

a NULL value as this function is really meant for its side-effect of generating a plot.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
z = matrix(rnorm(100 * 10), 100, 10)
y = rnorm(100)
fit1 = multiview(list(x=x,z=z), y, rho = 0)
plot(fit1, label = TRUE)

# Binomial
by = sample(c(0,1), 100, replace = TRUE)
fit2 = multiview(list(x=x,z=z), by, family = binomial(), rho=0.5)
plot(fit2, label=FALSE)

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = multiview(list(x=x,z=z), py, family = poisson(), rho=0.5)
plot(fit3, label=TRUE)
```

predict.cv.multiview *Make predictions from a "cv.multiview" object.*

Description

This function makes predictions from a cross-validated multiview model, using the stored "multiview" object, and the optimal value chosen for lambda.

Usage

```
## S3 method for class 'cv.multiview'
predict(object, newx, s = c("lambda.1se", "lambda.min"), ...)
```

Arguments

object	Fitted "cv.multiview" or object.
newx	List of new view matrices at which predictions are to be made.
s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the value s="lambda.1se" stored on the CV object. Alternatively s="lambda.min" can be used. If s is numeric, it is taken as the value(s) of lambda to be used. (For historical reasons we use the symbol 's' rather than 'lambda' to reference this parameter)
...	Not used. Other arguments to predict.

Details

This function makes it easier to use the results of cross-validation to make a prediction.

Value

The object returned depends on the ... argument which is passed on to the predict method for multiview objects.

Examples

```
# Gaussian
# Generate data based on a factor model
set.seed(1)
x = matrix(rnorm(100*10), 100, 10)
z = matrix(rnorm(100*10), 100, 10)
U = matrix(rnorm(100*5), 100, 5)
for (m in seq(5)){
    u = rnorm(100)
    x[, m] = x[, m] + u
    z[, m] = z[, m] + u
    U[, m] = U[, m] + u}
x = scale(x, center = TRUE, scale = FALSE)
z = scale(z, center = TRUE, scale = FALSE)
```

```
beta_U = c(rep(0.1, 5))
y = U %*% beta_U + 0.1 * rnorm(100)
fit1 = cv.multiview(list(x=x,z=z), y, rho = 0.3)
predict(fit1, newx = list(x[1:5, ],z[1:5,]), s = "lambda.min")

# Binomial

by = 1 * (y > median(y))
fit2 = cv.multiview(list(x=x,z=z), by, family = binomial(), rho = 0.9)
predict(fit2, newx = list(x[1:5, ],z[1:5,]), s = "lambda.min", type = "response")

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = cv.multiview(list(x=x,z=z), py, family = poisson(), rho = 0.6)
predict(fit3, newx = list(x[1:5, ],z[1:5,]), s = "lambda.min", type = "response")
```

`predict.multiview` *Get predictions from a* `multiview` *fit object*

Description

Gives fitted values, linear predictors, coefficients and number of non-zero coefficients from a fitted `multiview` object.

Usage

```
## S3 method for class 'multiview'
predict(
  object,
  newx,
  s = NULL,
  type = c("link", "response", "coefficients", "class", "nonzero"),
  exact = FALSE,
  newoffset,
  ...
)
```

Arguments

`object`	Fitted "multiview" object.
`newx`	list of new matrices for `x` at which predictions are to be made. Must be a list of matrices. This argument is not used for `type = c("coefficients","nonzero")`.
`s`	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model.

type	Type of prediction required. Type "link" gives the linear predictors (eta scale); Type "response" gives the fitted values (mu scale). Type "coefficients" computes the coefficients at the requested values for s. Type "nonzero" returns a list of the indices of the nonzero coefficients for each value of s. Type "class" returns class labels for binomial family only.
exact	This argument is relevant only when predictions are made at values of s (lambda) *different* from those used in the fitting of the original model. If exact=FALSE (default), then the predict function uses linear interpolation to make predictions for values of s (lambda) that do not coincide with those used in the fitting algorithm. While this is often a good approximation, it can sometimes be a bit coarse. With exact=TRUE, these different values of s are merged (and sorted) with object$lambda, and the model is refit before predictions are made. In this case, it is required to supply the original data x= and y= as additional named arguments to predict() or coef(). The workhorse predict.multiview() needs to update the model, and so needs the data used to create it. The same is true of weights, offset, penalty.factor, lower.limits, upper.limits if these were used in the original call. Failure to do so will result in an error.
newoffset	If an offset is used in the fit, then one must be supplied for making predictions (except for type="coefficients" or type="nonzero").
...	This is the mechanism for passing arguments like x= when exact=TRUE; see exact argument.

Value

The object returned depends on type.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
z = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = multiview(list(x=x,z=z), y, rho = 0)
predict(fit1, newx = list(x[1:10, ],z[1:10, ]), s = c(0.01, 0.005))

# Binomial
by = sample(c(0,1), 100, replace = TRUE)
fit2 = multiview(list(x=x,z=z), by, family = binomial(), rho=0.5)
predict(fit2, newx = list(x[1:10, ],z[1:10, ]), s = c(0.01, 0.005), type = "response")

# Poisson
py = matrix(rpois(100, exp(y)))
fit3 = multiview(list(x=x,z=z), py, family = poisson(), rho=0.5)
predict(fit3, newx = list(x[1:10, ],z[1:10, ]), s = c(0.01, 0.005), type = "response")
```

reshape_x_to_xlist *Return a new list of x matrices of same shapes as those in x_list*

Description

Return a new list of x matrices of same shapes as those in x_list

Usage

```
reshape_x_to_xlist(x, x_list)
```

Arguments

x	the column-binded entries of x_list
x_list	a list of x matrices with same number of rows nobs

select_matrix_list_columns *Select x_list columns specified by (conformable) list of indices*

Description

Select x_list columns specified by (conformable) list of indices

Usage

```
select_matrix_list_columns(x_list, indices)
```

Arguments

x_list	a list of x matrices with same number of rows nobs
indices	a vector of indices in 1:nvars

Value

a list of x matrices

to_nvar_index — *Translate from column indices in list of x matrices to indices in* 1:nvars. *No sanity checks for efficiency*

Description

Translate from column indices in list of x matrices to indices in 1:nvars. No sanity checks for efficiency

Usage

```
to_nvar_index(x_list, index_list)
```

Arguments

x_list	a list of x matrices with same number of rows nobs
index_list	a list of column indices for each matrix, including possibly column indices of length 0

Value

a vector of indices between 1 and nvars = sum of ncol(x) for x in x_list

to_xlist_index — *Translate indices in* 1:nvars *to column indices in list of x matrices. No sanity checks*

Description

Translate indices in 1:nvars to column indices in list of x matrices. No sanity checks

Usage

```
to_xlist_index(x_list, index)
```

Arguments

x_list	a list of x matrices with same number of rows nobs
index	vector of indices between 1 and nvars = sum of ncol(x) for x in x_list

Value

a conformed list of column indices for each matrix, including possibly column indices of length 0

`view.contribution` *Evaluate the contribution of data views in making prediction*

Description

Evaluate the contribution of each data view in making prediction. The function has two options. If `force` is set to `NULL`, the data view contribution is benchmarked by the null model. If `force` is set to a list of data views, the contribution is benchmarked by the model fit on this list of data views, and the function evaluates the marginal contribution of each additional data view on top of this benchmarking list of views. The function returns a table showing the percentage improvement in reducing error as compared to the bechmarking model made by each data view.

Usage

```
view.contribution(
  x_list,
  y,
  family = gaussian(),
  rho,
  s = c("lambda.min", "lambda.1se"),
  eval_data = c("train", "test"),
  weights = NULL,
  offset = NULL,
  mvlambda = NULL,
  type.measure = c("default", "mse", "deviance", "class", "auc", "mae", "C"),
  x_list_test = NULL,
  test_y = NULL,
  nfolds = 10,
  foldid = NULL,
  force = NULL,
  ...
)
```

Arguments

`x_list`	a list of `x` matrices with same number of rows `nobs`
`y`	the quantitative response with length equal to `nobs`, the (same) number of rows in each `x` matrix
`family`	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. Default is stats::gaussian. (See stats::family for details on family functions.)
`rho`	the weight on the agreement penalty, default 0. `rho=0` is a form of early fusion, and `rho=1` is a form of late fusion. We recommend trying a few values of `rho` including 0, 0.1, 0.25, 0.5, and 1 first; sometimes `rho` larger than 1 can also be helpful.

s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the value s="lambda.1se" stored on the CV object. Alternatively s="lambda.min" can be used. If s is numeric, it is taken as the value(s) of lambda to be used. (For historical reasons we use the symbol 's' rather than 'lambda' to reference this parameter)
eval_data	If train, we evaluate the contribution of data views based on training data using cross validation error; if test, we evaluate the contribution of data views based on test data. Default is train. If set to test, users need to provide the test data, i.e. x_list_test and y_test.
weights	Observation weights; defaults to 1 per observation
offset	Offset vector (matrix) as in multiview
mvlambda	A user supplied lambda sequence, default NULL. Typical usage is to have the program compute its own mvlambda sequence. This sequence, in general, is different from that used in the glmnet::glmnet() call (named lambda). Note that this is done for the full model (master sequence), and separately for each fold. The fits are then aligned using the glmnet lambda sequence associated with the master sequence (see the alignment argument for additional details). Adapting mvlambda for each fold leads to better convergence. When mvlambda is supplied, the same sequence is used everywhere, but in some GLMs can lead to convergence issues.
type.measure	loss to use for cross-validation. Currently five options, not all available for all models. The default is type.measure="deviance", which uses squared-error for gaussian models (a.k.a type.measure="mse" there), deviance for logistic and poisson regression, and partial-likelihood for the Cox model. type.measure="class" applies to binomial and multinomial logistic regression only, and gives misclassification error. type.measure="auc" is for two-class logistic regression only, and gives area under the ROC curve. type.measure="mse" or type.measure="mae" (mean absolute error) can be used by all models except the "cox"; they measure the deviation from the fitted mean to the response. type.measure="C" is Harrel's concordance measure, only available for cox models.
x_list_test	A list of x matrices in the test data for evaluation.
test_y	The quantitative response in the test data with length equal to the number of rows in each x matrix of the test data.
nfolds	number of folds - default is 10. Although nfolds can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is nfolds=3
foldid	an optional vector of values between 1 and nfold identifying what fold each observation is in. If supplied, nfold can be missing.
force	If NULL, the data view contribution is benchmarked by the null model. If users want to benchmark by the model fit on a specified list of data views, force needs to be set to this list of benchmarking data views, i.e. a list of x matrices. The function then evaluates the marginal contribution of each additional data, i.e. the data views in x_list but not in force, on top of the benchmarking views.
...	Other arguments that can be passed to multiview

Value

a data frame consisting of the view, error metric, and percentage improvement.

Examples

```
set.seed(3)
# Simulate data based on the factor model
x = matrix(rnorm(200*20), 200, 20)
z = matrix(rnorm(200*20), 200, 20)
w = matrix(rnorm(200*20), 200, 20)
U = matrix(rep(0, 200*10), 200, 10) # latent factors
for (m in seq(10)){
    u = rnorm(200)
    x[, m] = x[, m] + u
    z[, m] = z[, m] + u
    w[, m] = w[, m] + u
    U[, m] = U[, m] + u}
beta_U = c(rep(2, 5),rep(-2, 5))
y = U %*% beta_U + 3 * rnorm(100)

# Split training and test sets
smp_size_train = floor(0.9 * nrow(x))
train_ind = sort(sample(seq_len(nrow(x)), size = smp_size_train))
test_ind = setdiff(seq_len(nrow(x)), train_ind)
train_X = scale(x[train_ind, ])
test_X = scale(x[test_ind, ])
train_Z <- scale(z[train_ind, ])
test_Z <- scale(z[test_ind, ])
train_W <- scale(w[train_ind, ])
test_W <- scale(w[test_ind, ])
train_y <- y[train_ind, ]
test_y <- y[test_ind, ]
foldid = sample(rep_len(1:10, dim(train_X)[1]))

# Benchmarked by the null model:
rho = 0.3
view.contribution(x_list=list(x=train_X,z=train_Z), train_y, rho = rho,
                  eval_data = 'train', family = gaussian())
view.contribution(x_list=list(x=train_X,z=train_Z), train_y, rho = rho,
                  eval_data = 'test', family = gaussian(),
                  x_list_test=list(x=test_X,z=test_Z), test_y=test_y)

# Force option -- benchmarked by the model train on a specified list of data views:
view.contribution(x_list=list(x=train_X,z=train_Z,w=train_W), train_y, rho = rho,
                  eval_data = 'train', family = gaussian(), force=list(x=train_X))
```

`weighted_mean_sd` *Helper function to compute weighted mean and standard deviation*

Description

Helper function to compute weighted mean and standard deviation. Deals gracefully whether x is sparse matrix or not.

Usage

```
weighted_mean_sd(x, weights = rep(1, nrow(x)))
```

Arguments

x	Observation matrix.
weights	Optional weight vector.

Value

A list with components.

mean	vector of weighted means of columns of x
sd	vector of weighted standard deviations of columns of x

Index