Package 'SuperLearner'

February 4, 2016

Type Package

Title Super Learner Prediction

Version 2.0-19

Date 2016-02-02

Author Eric Polley, Erin LeDell, Mark van der Laan

Maintainer Eric Polley <polley.eric@mayo.edu>

Description Implements the super learner prediction method and contains a library of prediction algorithms to be used in the super learner.

License GPL-3

URL https://github.com/ecpolley/SuperLearner

Depends R (>= 2.14.0), nnls

Imports cvAUC

Suggests arm, caret, class, e1071, earth, gam, gbm, genefilter, ggplot2, glmnet, Hmisc, ipred, lattice, LogicReg, MASS, mda, mlbench, nloptr, nnet, party, polspline, quadprog, randomForest, ROCR, rpart, SIS, spls, stepPlr, sva

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2016-02-04 14:56:53

R topics documented:

`CV.SuperLearner` *Function to get V-fold cross-validated risk estimate for super learner*

Description

Function to get V-fold cross-validated risk estimate for super learner. This function simply splits the data into V folds and then calls SuperLearner. Most of the arguments are passed directly to SuperLearner.

Usage

```
CV.SuperLearner(Y, X, V = 20, family = gaussian(), SL.library,
  method = "method.NNLS", id = NULL, verbose = FALSE,
  control = list(saveFitLibrary = FALSE), cvControl = list(),
  obsWeights = NULL, saveAll = TRUE, parallel = "seq")
```

Arguments

`Y`	The outcome.
`X`	The covariates.
`V`	The number of folds for `CV.SuperLearner`. This is not the number of folds for `SuperLearner`. The number of folds for `SuperLearner` is controlled with `cvControl`.
`family`	Currently allows `gaussian` or `binomial` to describe the error distribution. Link function information will be ignored and should be contained in the method argument below.
`SL.library`	Either a character vector of prediction algorithms or a list containing character vectors. See details below for examples on the structure. A list of functions included in the SuperLearner package can be found with `listWrappers()`.
`method`	A list (or a function to create a list) containing details on estimating the coefficients for the super learner and the model to combine the individual algorithms in the library. See `?method.template` for details. Currently, the built in options are either "method.NNLS" (the default), "method.NNLS2", "method.NNloglik",

"method.CC_LS", "method.CC_nloglik", or "method.AUC". NNLS and NNLS2 are non-negative least squares based on the Lawson-Hanson algorithm and the dual method of Goldfarb and Idnani, respectively. NNLS and NNLS2 will work for both gaussian and binomial outcomes. NNloglik is a non-negative binomial likelihood maximization using the BFGS quasi-Newton optimization method. NN* methods are normalized so weights sum to one. CC_LS uses Goldfarb and Idnani's quadratic programming algorithm to calculate the best convex combination of weights to minimize the squared error loss. CC_nloglik calculates the convex combination of weights that minimize the negative binomial log likelihood on the logistic scale using the sequential quadratic programming algorithm. AUC, which only works for binary outcomes, uses the Nelder-Mead method via the optim function to minimize rank loss (equivalent to maximizing AUC).

`id` Optional cluster identification variable. For the cross-validation splits, `id` forces observations in the same cluster to be in the same validation fold. `id` is passed to the prediction and screening algorithms in SL.library, but be sure to check the individual wrappers as many of them ignore the information.

`verbose` Logical; TRUE for printing progress during the computation (helpful for debugging).

`control` A list of parameters to control the estimation process. Parameters include `saveFitLibrary` and `trimLogit`. See `SuperLearner.control` for details.

`cvControl` A list of parameters to control the cross-validation process. Parameters include `V`, `stratifyCV`, `shuffle` and `validRows`. See `SuperLearner.CV.control` for details.

`obsWeights` Optional observation weights variable. As with `id` above, `obsWeights` is passed to the prediction and screening algorithms, but many of the built in wrappers ignore (or can't use) the information. If you are using observation weights, make sure the library you specify uses the information.

`saveAll` Logical; Should the entire `SuperLearner` object be saved for each fold?

`parallel` Options for parallel computation of the V-fold step. Use "seq" (the default) for sequential computation. `parallel = 'multicore'` to use `mclapply` for the V-fold step (but note that `SuperLearner()` will still be sequential). The default for mclapply is to check the `mc.cores` option, and if not set to default to 2 cores. Be sure to set `options()$mc.cores` to the desired number of cores if you don't want the default. Or `parallel` can be the name of a snow cluster and will use `parLapply` for the V-fold step. For both multicore and snow, the inner `SuperLearner` calls will be sequential.

Details

The `SuperLearner` function builds a estimator, but does not contain an estimate on the performance of the estimator. Various methods exist for estimator performance evaluation. If you are familiar with the super learner algorithm, it should be no surprise we recommend using cross-validation to evaluate the honest performance of the super learner estimator. The function `CV.SuperLearner` computes the usual V-fold cross-validated risk estimate for the super learner (and all algorithms in `SL.library` for comparison).

Value

An object of class `CV.SuperLearner` (a list) with components:

`call`	The matched call.
`AllSL`	If `saveAll = TRUE`, a list with output from each call to `SuperLearner`, otherwise NULL.
`SL.predict`	The predicted values from the super learner when each particular row was part of the validation fold.
`discreteSL.predict`	The traditional cross-validated selector. Picks the algorithm with the smallest cross-validated risk (in super learner terms, gives that algorithm coefficient 1 and all others 0).
`whichDiscreteSL`	A list of length `V`. The elements in the list are the algorithm that had the smallest cross-validated risk estimate for that fold.
`library.predict`	A matrix with the predicted values from each algorithm in `SL.library`. The columns are the algorithms in `SL.library` and the rows represent the predicted values when that particular row was in the validation fold (i.e. not used to fit that estimator).
`coef`	A matrix with the coefficients for the super learner on each fold. The columns are the algorithms in `SL.library` the rows are the folds.
`folds`	A list containing the row numbers for each validation fold.
`V`	Number of folds for `CV.SuperLearner`.
`libraryNames`	A character vector with the names of the algorithms in the library. The format is 'predictionAlgorithm_screeningAlgorithm' with '_All' used to denote the prediction algorithm run on all variables in X.
`SL.library`	Returns `SL.library` in the same format as the argument with the same name above.
`method`	A list with the method functions.
`Y`	The outcome

Author(s)

Eric C Polley <polley.eric@mayo.edu>

See Also

`SuperLearner`

Examples

```
set.seed(23432)
## training set
n <- 500
p <- 50
```

```
X <- matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X) <- paste("X", 1:p, sep="")
X <- data.frame(X)
Y <- X[, 1] + sqrt(abs(X[, 2] * X[, 3])) + X[, 2] - X[, 3] + rnorm(n)

# build Library and run Super Learner
SL.library <- c("SL.glm", "SL.randomForest", "SL.gam", "SL.polymars", "SL.mean")
## Not run:
test <- CV.SuperLearner(Y = Y, X = X, V = 10, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS")
test
summary(test)
# Look at the coefficients across folds
coef(test)

## End(Not run)
```

CVFolds *Generate list of row numbers for each fold in the cross-validation*

Description

Generate list of row numbers for each fold in the cross-validation. CVFolds is used in the SuperLearner to create the cross-validation splits.

Usage

```
CVFolds(N, id, Y, cvControl)
```

Arguments

N	Sample size
id	Optional cluster id variable. If present, all observations in the same cluster will always be in the same split.
Y	outcome
cvControl	Control parameters for the cross-validation step. See SuperLearner.CV.control for details.

Value

validRows	A list of length V where each element in the list is a vector with the row numbers of the corresponding validation sample.

Author(s)

Eric C Polley <polley.eric@mayo.edu>

listWrappers *list all wrapper functions in SuperLearner*

Description

List all wrapper functions in SuperLearner package

Usage

```
listWrappers(what = "both")
```

Arguments

what	What list to return. Can be both for both prediction algorithms and screening algorithms, SL for the prediction algorithms, screen for the screening algorithms, method for the estimation method details, or anything else will return a list of all (exported) functions in the SuperLearner package. Additional wrapper functions are available at https://github.com/ecpolley/SuperLearnerExtra.

Value

Invisible character vector with all exported functions in the SuperLearner package

Author(s)

Eric C Polley <eric.polley@nih.gov>

See Also

SuperLearner

Examples

```
listWrappers(what = "SL")
listWrappers(what = "screen")
```

plot.CV.SuperLearner *Graphical display of the V-fold CV risk estimates*

Description

The function plots the V-fold cross-validated risk estimates for the super learner, the discrete super learner and each algorithm in the library. By default the estimates will be sorted and include an asymptotic 95% confidence interval.

Usage

```
## S3 method for class 'CV.SuperLearner'
plot(x, package = "ggplot2", constant = qnorm(0.975), sort = TRUE, ...)
```

Arguments

x	The output from CV.SuperLearner.
package	Either "ggplot2" or "lattice". The package selected must be available.
constant	A numeric value. The confidence interval is defined as p +/- constant * se, where p is the point estimate and se is the standard error. The default is the quantile of the standard normal corresponding to a 95% CI.
sort	Logical. Should the rows in the plot be sorted from the smallest to the largest point estimate. If FALSE, then the order is super learner, discrete super learner, then the estimators in SL.library.
...	Additional arguments for summary.CV.SuperLearner

Details

see summary.CV.SuperLearner for details on how the estimates are computed

Value

Returns the plot (either a ggplot2 object (class ggplot) or a lattice object (class trellis))

Author(s)

Eric C Polley <eric.polley@nih.gov>

See Also

summary.CV.SuperLearner and CV.SuperLearner

---

predict.SuperLearner *Predict method for SuperLearner object*

Description

Obtains predictions on a new data set from a SuperLearner fit. May require the original data if one of the library algorithms uses the original data in its predict method.

Usage

```
## S3 method for class 'SuperLearner'
predict(object, newdata, X = NULL, Y = NULL, onlySL = FALSE, ...)
```

Arguments

object	Fitted object from SuperLearner
newdata	New X values for prediction
X	Original data set used to fit object
Y	Original outcome used to fit object
onlySL	Logical. If TRUE, only compute predictions for algorithms with non-zero coefficients in the super learner object. Default is FALSE (computes predictions for all algorithms in library).
...	Additional arguments passed to the predict.SL.* functions

Details

If newdata is omitted the predicted values from object are returned. Each algorithm in the Super Learner library needs to have a corresponding prediction function with the "predict." prefixed onto the algorithm name (e.g. predict.SL.glm for SL.glm).

Value

pred	Predicted values from Super Learner fit
library.predict	Predicted values for each algorithm in library

Author(s)

Eric C Polley <eric.polley@nih.gov>

See Also

SuperLearner

---

recombineCVSL *Recombine a CV.SuperLearner fit using a new metalearning method*

Description

Function to re-compute the V-fold cross-validated risk estimate for super learner using a new metalearning method. This function takes as input an existing CV.SuperLearner fit and applies the recombineSL fit to each of the V Super Learner fits.

Usage

```
recombineCVSL(object, method = "method.NNloglik", verbose = FALSE,
  saveAll = TRUE, parallel = "seq")
```

Arguments

object	Fitted object from CV.SuperLearner.
method	A list (or a function to create a list) containing details on estimating the coefficients for the super learner and the model to combine the individual algorithms in the library. See ?method.template for details. Currently, the built in options are either "method.NNLS" (the default), "method.NNLS2", "method.NNloglik", "method.CC_LS", "method.CC_nloglik", or "method.AUC". NNLS and NNLS2 are non-negative least squares based on the Lawson-Hanson algorithm and the dual method of Goldfarb and Idnani, respectively. NNLS and NNLS2 will work for both gaussian and binomial outcomes. NNloglik is a non-negative binomial likelihood maximization using the BFGS quasi-Newton optimization method. NN* methods are normalized so weights sum to one. CC_LS uses Goldfarb and Idnani's quadratic programming algorithm to calculate the best convex combination of weights to minimize the squared error loss. CC_nloglik calculates the convex combination of weights that minimize the negative binomial log likelihood on the logistic scale using the sequential quadratic programming algorithm. AUC, which only works for binary outcomes, uses the Nelder-Mead method via the optim function to minimize rank loss (equivalent to maximizing AUC).
verbose	logical; TRUE for printing progress during the computation (helpful for debugging).
saveAll	Logical; Should the entire SuperLearner object be saved for each fold?
parallel	Options for parallel computation of the V-fold step. Use "seq" (the default) for sequential computation. parallel = 'multicore' to use mclapply for the V-fold step (but note that SuperLearner() will still be sequential). Or parallel can be the name of a snow cluster and will use parLapply for the V-fold step. For both multicore and snow, the inner SuperLearner calls will be sequential.

Details

The function recombineCVSL computes the usual V-fold cross-validated risk estimate for the super learner (and all algorithms in SL.library for comparison), using a newly specified metalearning method. The weights for each algorithm in SL.library are re-estimated using the new metalearner, however the base learner fits are not regenerated, so this function saves a lot of computation time as opposed to using the CV.SuperLearner function with a new method argument. The output is identical to the output from the CV.SuperLearner function.

Value

An object of class CV.SuperLearner (a list) with components:

call	The matched call.
AllSL	If saveAll = TRUE, a list with output from each call to SuperLearner, otherwise NULL.
SL.predict	The predicted values from the super learner when each particular row was part of the validation fold.

`discreteSL.predict`
: The traditional cross-validated selector. Picks the algorithm with the smallest cross-validated risk (in super learner terms, gives that algorithm coefficient 1 and all others 0).

`whichDiscreteSL`
: A list of length `V`. The elements in the list are the algorithm that had the smallest cross-validated risk estimate for that fold.

`library.predict`
: A matrix with the predicted values from each algorithm in `SL.library`. The columns are the algorithms in `SL.library` and the rows represent the predicted values when that particular row was in the validation fold (i.e. not used to fit that estimator).

`coef`
: A matrix with the coefficients for the super learner on each fold. The columns are the algorithms in `SL.library` the rows are the folds.

`folds`
: A list containing the row numbers for each validation fold.

`V`
: Number of folds for `CV.SuperLearner`.

`libraryNames`
: A character vector with the names of the algorithms in the library. The format is 'predictionAlgorithm_screeningAlgorithm' with '_All' used to denote the prediction algorithm run on all variables in X.

`SL.library`
: Returns `SL.library` in the same format as the argument with the same name above.

`method`
: A list with the method functions.

`Y`
: The outcome

Author(s)

Erin LeDell <ledell@berkeley.edu>

See Also

`recombineSL`

Examples

```
## Not run:

# Binary outcome example adapted from SuperLearner examples

set.seed(1)
N <- 200
X <- matrix(rnorm(N*10), N, 10)
X <- as.data.frame(X)
Y <- rbinom(N, 1, plogis(.2*X[, 1] + .1*X[, 2] - .2*X[, 3] +
  .1*X[, 3]*X[, 4] - .2*abs(X[, 4])))

SL.library <- c("SL.glmnet", "SL.glm", "SL.knn", "SL.gam", "SL.mean")

# least squares loss function
```

```
set.seed(1) # for reproducibility
cvfit_nnls <- CV.SuperLearner(Y = Y, X = X, V = 10, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS", family = binomial())
cvfit_nnls$coef
#    SL.glmnet_All SL.glm_All  SL.knn_All SL.gam_All SL.mean_All
# 1      0.0000000 0.00000000 0.000000000  0.4143862   0.5856138
# 2      0.0000000 0.00000000 0.304802397  0.3047478   0.3904498
# 3      0.0000000 0.00000000 0.002897533  0.5544075   0.4426950
# 4      0.0000000 0.20322642 0.000000000  0.1121891   0.6845845
# 5      0.1743973 0.00000000 0.032471026  0.3580624   0.4350693
# 6      0.0000000 0.00000000 0.099881535  0.3662309   0.5338876
# 7      0.0000000 0.00000000 0.234876082  0.2942472   0.4708767
# 8      0.0000000 0.06424676 0.113988158  0.5600208   0.2617443
# 9      0.0000000 0.00000000 0.338030342  0.2762604   0.3857093
# 10     0.3022442 0.00000000 0.294226204  0.1394534   0.2640762


# negative log binomial likelihood loss function
cvfit_nnloglik <- recombineCVSL(cvfit_nnls, method = "method.NNloglik")
cvfit_nnloglik$coef
#    SL.glmnet_All SL.glm_All SL.knn_All SL.gam_All SL.mean_All
# 1      0.0000000  0.0000000 0.00000000  0.5974799  0.40252010
# 2      0.0000000  0.0000000 0.31177345  0.6882266  0.00000000
# 3      0.0000000  0.0000000 0.01377469  0.8544238  0.13180152
# 4      0.0000000  0.1644188 0.00000000  0.2387919  0.59678930
# 5      0.2142254  0.0000000 0.00000000  0.3729426  0.41283197
# 6      0.0000000  0.0000000 0.00000000  0.5847150  0.41528502
# 7      0.0000000  0.0000000 0.47538172  0.5080311  0.01658722
# 8      0.0000000  0.0000000 0.00000000  1.0000000  0.00000000
# 9      0.0000000  0.0000000 0.45384961  0.2923480  0.25380243
# 10     0.3977816  0.0000000 0.27927906  0.1606384  0.16230097

# If we use the same seed as the original `cvfit_nnls`, then
# the recombineCVSL and CV.SuperLearner results will be identical
# however, the recombineCVSL version will be much faster since
# it doesn't have to re-fit all the base learners, V times each.
set.seed(1)
cvfit_nnloglik2 <- CV.SuperLearner(Y = Y, X = X, V = 10, SL.library = SL.library,
  verbose = TRUE, method = "method.NNloglik", family = binomial())
cvfit_nnloglik2$coef
#    SL.glmnet_All SL.glm_All SL.knn_All SL.gam_All SL.mean_All
# 1      0.0000000  0.0000000 0.00000000  0.5974799  0.40252010
# 2      0.0000000  0.0000000 0.31177345  0.6882266  0.00000000
# 3      0.0000000  0.0000000 0.01377469  0.8544238  0.13180152
# 4      0.0000000  0.1644188 0.00000000  0.2387919  0.59678930
# 5      0.2142254  0.0000000 0.00000000  0.3729426  0.41283197
# 6      0.0000000  0.0000000 0.00000000  0.5847150  0.41528502
# 7      0.0000000  0.0000000 0.47538172  0.5080311  0.01658722
# 8      0.0000000  0.0000000 0.00000000  1.0000000  0.00000000
# 9      0.0000000  0.0000000 0.45384961  0.2923480  0.25380243
# 10     0.3977816  0.0000000 0.27927906  0.1606384  0.16230097
```

```
## End(Not run)
```

`recombineSL` *Recombine a SuperLearner fit using a new metalearning method*

Description

The `recombineSL` function takes an existing SuperLearner fit and a new metalearning method and returns a new SuperLearner fit with updated base learner weights.

Usage

```
recombineSL(object, Y, method = "method.NNloglik", verbose = FALSE)
```

Arguments

`object`	Fitted object from `SuperLearner`.
`Y`	The outcome in the training data set. Must be a numeric vector.
`method`	A list (or a function to create a list) containing details on estimating the coefficients for the super learner and the model to combine the individual algorithms in the library. See `?method.template` for details. Currently, the built in options are either "method.NNLS" (the default), "method.NNLS2", "method.NNloglik", "method.CC_LS", "method.CC_nloglik", or "method.AUC". NNLS and NNLS2 are non-negative least squares based on the Lawson-Hanson algorithm and the dual method of Goldfarb and Idnani, respectively. NNLS and NNLS2 will work for both gaussian and binomial outcomes. NNloglik is a non-negative binomial likelihood maximization using the BFGS quasi-Newton optimization method. NN* methods are normalized so weights sum to one. CC_LS uses Goldfarb and Idnani's quadratic programming algorithm to calculate the best convex combination of weights to minimize the squared error loss. CC_nloglik calculates the convex combination of weights that minimize the negative binomial log likelihood on the logistic scale using the sequential quadratic programming algorithm. AUC, which only works for binary outcomes, uses the Nelder-Mead method via the optim function to minimize rank loss (equivalent to maximizing AUC).
`verbose`	logical; TRUE for printing progress during the computation (helpful for debugging).

Details

`recombineSL` re-fits the super learner prediction algorithm using a new metalearning method. The weights for each algorithm in `SL.library` are re-estimated using the new metalearner, however the base learner fits are not regenerated, so this function saves a lot of computation time as opposed to using the `SuperLearner` function with a new `method` argument. The output is identical to the output from the `SuperLearner` function.

Value

call	The matched call.
libraryNames	A character vector with the names of the algorithms in the library. The format is 'predictionAlgorithm_screeningAlgorithm' with '_All' used to denote the prediction algorithm run on all variables in X.
SL.library	Returns SL.library in the same format as the argument with the same name above.
SL.predict	The predicted values from the super learner for the rows in newX.
coef	Coefficients for the super learner.
library.predict	A matrix with the predicted values from each algorithm in SL.library for the rows in newX.
Z	The Z matrix (the cross-validated predicted values for each algorithm in SL.library).
cvRisk	A numeric vector with the V-fold cross-validated risk estimate for each algorithm in SL.library. Note that this does not contain the CV risk estimate for the SuperLearner, only the individual algorithms in the library.
family	Returns the family value from above
fitLibrary	A list with the fitted objects for each algorithm in SL.library on the full training data set.
varNames	A character vector with the names of the variables in X.
validRows	A list containing the row numbers for the V-fold cross-validation step.
method	A list with the method functions.
whichScreen	A logical matrix indicating which variables passed each screening algorithm.
control	The control list.
cvControl	The cvControl list.
errorsInCVLibrary	A logical vector indicating if any algorithms experienced an error within the CV step.
errorsInLibrary	A logical vector indicating if any algorithms experienced an error on the full data.

Author(s)

Erin LeDell <ledell@berkeley.edu>

References

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2008) Super Learner, *Statistical Applications of Genetics and Molecular Biology*, **6**, article 25. http://www.bepress.com/sagmb/vol6/iss1/art25

Examples

```
## Not run:

# Binary outcome example adapted from SuperLearner examples

set.seed(1)
N <- 200
X <- matrix(rnorm(N*10), N, 10)
X <- as.data.frame(X)
Y <- rbinom(N, 1, plogis(.2*X[, 1] + .1*X[, 2] - .2*X[, 3] +
  .1*X[, 3]*X[, 4] - .2*abs(X[, 4])))

SL.library <- c("SL.glmnet", "SL.glm", "SL.knn", "SL.gam", "SL.mean")

# least squares loss function
set.seed(1) # for reproducibility
fit_nnls <- SuperLearner(Y = Y, X = X, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS", family = binomial())
fit_nnls
#                    Risk       Coef
# SL.glmnet_All 0.2439433 0.01293059
# SL.glm_All    0.2461245 0.08408060
# SL.knn_All    0.2604000 0.09600353
# SL.gam_All    0.2471651 0.40761918
# SL.mean_All   0.2486049 0.39936611


# negative log binomial likelihood loss function
fit_nnloglik <- recombineSL(fit_nnls, Y = Y, method = "method.NNloglik")
fit_nnloglik
#                    Risk      Coef
# SL.glmnet_All 0.6815911 0.1577228
# SL.glm_All    0.6918926 0.0000000
# SL.knn_All          Inf 0.0000000
# SL.gam_All    0.6935383 0.6292881
# SL.mean_All   0.6904050 0.2129891

# If we use the same seed as the original `fit_nnls`, then
# the recombineSL and SuperLearner results will be identical
# however, the recombineSL version will be much faster since
# it doesn't have to re-fit all the base learners.
set.seed(1)
fit_nnloglik2 <- SuperLearner(Y = Y, X = X, SL.library = SL.library,
  verbose = TRUE, method = "method.NNloglik", family = binomial())
fit_nnloglik2
#                    Risk      Coef
# SL.glmnet_All 0.6815911 0.1577228
# SL.glm_All    0.6918926 0.0000000
# SL.knn_All          Inf 0.0000000
# SL.gam_All    0.6935383 0.6292881
# SL.mean_All   0.6904050 0.2129891
```

```
## End(Not run)
```

`SampleSplitSuperLearner` *Super Learner Prediction Function*

Description

A Prediction Function for the Super Learner. The `SuperLearner` function takes a training set pair (X,Y) and returns the predicted values based on a validation set. SampleSplitSuperLearner uses sample split validation whereas SuperLearner uses V-fold cross-validation.

Usage

```
SampleSplitSuperLearner(Y, X, newX = NULL, family = gaussian(), SL.library,
  method = "method.NNLS", id = NULL, verbose = FALSE,
  control = list(), split = 0.8, obsWeights = NULL)
```

Arguments

`Y`	The outcome in the training data set. Must be a numeric vector.
`X`	The predictor variables in the training data set, usually a data.frame.
`newX`	The predictor variables in the validation data set. The structure should match X. If missing, uses X for newX.
`SL.library`	Either a character vector of prediction algorithms or a list containing character vectors. See details below for examples on the structure. A list of functions included in the SuperLearner package can be found with `listWrappers()`.
`verbose`	logical; TRUE for printing progress during the computation (helpful for debugging).
`family`	Currently allows `gaussian` or `binomial` to describe the error distribution. Link function information will be ignored and should be contained in the method argument below.
`method`	A list (or a function to create a list) containing details on estimating the coefficients for the super learner and the model to combine the individual algorithms in the library. See `?method.template` for details. Currently, the built in options are either "method.NNLS" (the default), "method.NNLS2", "method.NNloglik", "method.CC_LS", or "method.CC_nloglik". NNLS and NNLS2 are non-negative least squares based on the Lawson-Hanson algorithm and the dual method of Goldfarb and Idnani, respectively. NNLS and NNLS2 will work for both gaussian and binomial outcomes. NNloglik is a non-negative binomial likelihood maximization using the BFGS quasi-Newton optimization method. NN* methods are normalized so weights sum to one. CC_LS uses Goldfarb and Idnani's quadratic programming algorithm to calculate the best convex combination of weights to minimize the squared error loss. CC_nloglik calculates the convex combination of weights that minimize the negative binomial log likelihood on the logistic scale using the sequential quadratic programming algorithm.

id	Optional cluster identification variable. For the cross-validation splits, id forces observations in the same cluster to be in the same validation fold. id is passed to the prediction and screening algorithms in SL.library, but be sure to check the individual wrappers as many of them ignore the information.
obsWeights	Optional observation weights variable. As with id above, obsWeights is passed to the prediction and screening algorithms, but many of the built in wrappers ignore (or can't use) the information. If you are using observation weights, make sure the library you specify uses the information.
control	A list of parameters to control the estimation process. Parameters include saveFitLibrary and trimLogit. See SuperLearner.control for details.
split	Either a single value between 0 and 1 indicating the fraction of the samples for the training split. A value of 0.8 will randomly assign 80 percent of the samples to the training split and the other 20 percent to the validation split. Alternatively, split can be a numeric vector with the row numbers of X corresponding to the validation split. All other rows not in the vector will be considered in the training split.

Details

SuperLearner fits the super learner prediction algorithm. The weights for each algorithm in SL.library is estimated, along with the fit of each algorithm.

The prescreen algorithms. These algorithms first rank the variables in X based on either a univariate regression p-value of the randomForest variable importance. A subset of the variables in X is selected based on a pre-defined cut-off. With this subset of the X variables, the algorithms in SL.library are then fit.

The SuperLearner package contains a few prediction and screening algorithm wrappers. The full list of wrappers can be viewed with listWrappers(). The design of the SuperLearner package is such that the user can easily add their own wrappers. We also maintain a website with additional examples of wrapper functions at https://github.com/ecpolley/SuperLearnerExtra.

Value

call	The matched call.
libraryNames	A character vector with the names of the algorithms in the library. The format is 'predictionAlgorithm_screeningAlgorithm' with '_All' used to denote the prediction algorithm run on all variables in X.
SL.library	Returns SL.library in the same format as the argument with the same name above.
SL.predict	The predicted values from the super learner for the rows in newX.
coef	Coefficients for the super learner.
library.predict	A matrix with the predicted values from each algorithm in SL.library for the rows in newX.
Z	The Z matrix (the cross-validated predicted values for each algorithm in SL.library).

cvRisk	A numeric vector with the V-fold cross-validated risk estimate for each algorithm in SL.library. Note that this does not contain the CV risk estimate for the SuperLearner, only the individual algorithms in the library.
family	Returns the family value from above
fitLibrary	A list with the fitted objects for each algorithm in SL.library on the full training data set.
varNames	A character vector with the names of the variables in X.
validRows	A list containing the row numbers for the V-fold cross-validation step.
method	A list with the method functions.
whichScreen	A logical matrix indicating which variables passed each screening algorithm.
control	The control list.
split	The split value.
errorsInCVLibrary	A logical vector indicating if any algorithms experienced an error within the CV step.
errorsInLibrary	A logical vector indicating if any algorithms experienced an error on the full data.

Author(s)

Eric C Polley <eric.polley@nih.gov>

References

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2008) Super Learner, *Statistical Applications of Genetics and Molecular Biology*, **6**, article 25. http://www.bepress.com/sagmb/vol6/iss1/art25

Examples

```
## Not run:
## simulate data
set.seed(23432)
## training set
n <- 500
p <- 50
X <- matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X) <- paste("X", 1:p, sep="")
X <- data.frame(X)
Y <- X[, 1] + sqrt(abs(X[, 2] * X[, 3])) + X[, 2] - X[, 3] + rnorm(n)

## test set
m <- 1000
newX <- matrix(rnorm(m*p), nrow = m, ncol = p)
colnames(newX) <- paste("X", 1:p, sep="")
newX <- data.frame(newX)
newY <- newX[, 1] + sqrt(abs(newX[, 2] * newX[, 3])) + newX[, 2] -
```

```
  newX[, 3] + rnorm(m)

# generate Library and run Super Learner
SL.library <- c("SL.glm", "SL.randomForest", "SL.gam",
  "SL.polymars", "SL.mean")
test <- SampleSplitSuperLearner(Y = Y, X = X, newX = newX, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS")
test

# library with screening
SL.library <- list(c("SL.glmnet", "All"), c("SL.glm", "screen.randomForest",
  "All", "screen.SIS"), "SL.randomForest", c("SL.polymars", "All"), "SL.mean")
test <- SuperLearner(Y = Y, X = X, newX = newX, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS")
test

# binary outcome
set.seed(1)
N <- 200
X <- matrix(rnorm(N*10), N, 10)
X <- as.data.frame(X)
Y <- rbinom(N, 1, plogis(.2*X[, 1] + .1*X[, 2] - .2*X[, 3] +
  .1*X[, 3]*X[, 4] - .2*abs(X[, 4])))

SL.library <- c("SL.glmnet", "SL.glm", "SL.knn", "SL.gam", "SL.mean")

# least squares loss function
test.NNLS <- SampleSplitSuperLearner(Y = Y, X = X, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS", family = binomial())
test.NNLS

## End(Not run)
```

`summary.CV.SuperLearner` *Summary Function for Cross-Validated Super Learner*

Description

summary method for the `CV.SuperLearner` function

Usage

```
## S3 method for class 'CV.SuperLearner'
summary(object, obsWeights = NULL, ...)

## S3 method for class 'summary.CV.SuperLearner'
print(x, digits, ...)
```

Arguments

object	An object of class "CV.SuperLearner", the result of a call to CV.SuperLearner.
x	An object of class "summary.CV.SuperLearner", the result of a call to summary.CV.SuperLearner.
obsWeights	Optional vector for observation weights.
digits	The number of significant digits to use when printing.
...	additional arguments ...

Details

Summary method for CV.SuperLearner. Calculates the V-fold cross-validated estimate of either the mean squared error or the -2*log(L) depending on the loss function used.

Value

summary.CV.SuperLearner returns a list with components

call	The function call from CV.SuperLearner
method	Describes the loss function used. Currently either least squares of negative log Likelihood.
V	Number of folds
Risk.SL	Risk estimate for the super learner
Risk.dSL	Risk estimate for the discrete super learner (the cross-validation selector)
Risk.library	A matrix with the risk estimates for each algorithm in the library
Table	A table with the mean risk estimate and standard deviation across the folds for the super learner and all algorithms in the library

Author(s)

Eric C Polley <eric.polley@nih.gov>

See Also

CV.SuperLearner

SuperLearner *Super Learner Prediction Function*

Description

A Prediction Function for the Super Learner. The SuperLearner function takes a training set pair (X,Y) and returns the predicted values based on a validation set.

Usage

```
SuperLearner(Y, X, newX = NULL, family = gaussian(), SL.library,
  method = "method.NNLS", id = NULL, verbose = FALSE,
  control = list(), cvControl = list(), obsWeights = NULL)
```

Arguments

Y	The outcome in the training data set. Must be a numeric vector.
X	The predictor variables in the training data set, usually a data.frame.
newX	The predictor variables in the validation data set. The structure should match X. If missing, uses X for newX.
SL.library	Either a character vector of prediction algorithms or a list containing character vectors. See details below for examples on the structure. A list of functions included in the SuperLearner package can be found with `listWrappers()`.
verbose	logical; TRUE for printing progress during the computation (helpful for debugging).
family	Currently allows `gaussian` or `binomial` to describe the error distribution. Link function information will be ignored and should be contained in the method argument below.
method	A list (or a function to create a list) containing details on estimating the coefficients for the super learner and the model to combine the individual algorithms in the library. See `?method.template` for details. Currently, the built in options are either "method.NNLS" (the default), "method.NNLS2", "method.NNloglik", "method.CC_LS", "method.CC_nloglik", or "method.AUC". NNLS and NNLS2 are non-negative least squares based on the Lawson-Hanson algorithm and the dual method of Goldfarb and Idnani, respectively. NNLS and NNLS2 will work for both gaussian and binomial outcomes. NNloglik is a non-negative binomial likelihood maximization using the BFGS quasi-Newton optimization method. NN* methods are normalized so weights sum to one. CC_LS uses Goldfarb and Idnani's quadratic programming algorithm to calculate the best convex combination of weights to minimize the squared error loss. CC_nloglik calculates the convex combination of weights that minimize the negative binomial log likelihood on the logistic scale using the sequential quadratic programming algorithm. AUC, which only works for binary outcomes, uses the Nelder-Mead method via the optim function to minimize rank loss (equivalent to maximizing AUC).
id	Optional cluster identification variable. For the cross-validation splits, `id` forces observations in the same cluster to be in the same validation fold. `id` is passed to the prediction and screening algorithms in SL.library, but be sure to check the individual wrappers as many of them ignore the information.
obsWeights	Optional observation weights variable. As with `id` above, `obsWeights` is passed to the prediction and screening algorithms, but many of the built in wrappers ignore (or can't use) the information. If you are using observation weights, make sure the library you specify uses the information.
control	A list of parameters to control the estimation process. Parameters include `saveFitLibrary` and `trimLogit`. See `SuperLearner.control` for details.

cvControl — A list of parameters to control the cross-validation process. Parameters include V, stratifyCV, shuffle and validRows. See SuperLearner.CV.control for details.

Details

SuperLearner fits the super learner prediction algorithm. The weights for each algorithm in SL.library is estimated, along with the fit of each algorithm.

The prescreen algorithms. These algorithms first rank the variables in X based on either a univariate regression p-value of the randomForest variable importance. A subset of the variables in X is selected based on a pre-defined cut-off. With this subset of the X variables, the algorithms in SL.library are then fit.

The SuperLearner package contains a few prediction and screening algorithm wrappers. The full list of wrappers can be viewed with listWrappers(). The design of the SuperLearner package is such that the user can easily add their own wrappers. We also maintain a website with additional examples of wrapper functions at https://github.com/ecpolley/SuperLearnerExtra.

Value

call — The matched call.

libraryNames — A character vector with the names of the algorithms in the library. The format is 'predictionAlgorithm_screeningAlgorithm' with '_All' used to denote the prediction algorithm run on all variables in X.

SL.library — Returns SL.library in the same format as the argument with the same name above.

SL.predict — The predicted values from the super learner for the rows in newX.

coef — Coefficients for the super learner.

library.predict — A matrix with the predicted values from each algorithm in SL.library for the rows in newX.

Z — The Z matrix (the cross-validated predicted values for each algorithm in SL.library).

cvRisk — A numeric vector with the V-fold cross-validated risk estimate for each algorithm in SL.library. Note that this does not contain the CV risk estimate for the SuperLearner, only the individual algorithms in the library.

family — Returns the family value from above

fitLibrary — A list with the fitted objects for each algorithm in SL.library on the full training data set.

varNames — A character vector with the names of the variables in X.

validRows — A list containing the row numbers for the V-fold cross-validation step.

method — A list with the method functions.

whichScreen — A logical matrix indicating which variables passed each screening algorithm.

control — The control list.

cvControl — The cvControl list.

`errorsInCVLibrary`
: A logical vector indicating if any algorithms experienced an error within the CV step.

`errorsInLibrary`
: A logical vector indicating if any algorithms experienced an error on the full data.

Author(s)

Eric C Polley <polley.eric@mayo.edu>

References

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2008) Super Learner, *Statistical Applications of Genetics and Molecular Biology*, **6**, article 25. http://www.bepress.com/sagmb/vol6/iss1/art25

Examples

```
## Not run:
## simulate data
set.seed(23432)
## training set
n <- 500
p <- 50
X <- matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X) <- paste("X", 1:p, sep="")
X <- data.frame(X)
Y <- X[, 1] + sqrt(abs(X[, 2] * X[, 3])) + X[, 2] - X[, 3] + rnorm(n)

## test set
m <- 1000
newX <- matrix(rnorm(m*p), nrow = m, ncol = p)
colnames(newX) <- paste("X", 1:p, sep="")
newX <- data.frame(newX)
newY <- newX[, 1] + sqrt(abs(newX[, 2] * newX[, 3])) + newX[, 2] -
  newX[, 3] + rnorm(m)

# generate Library and run Super Learner
SL.library <- c("SL.glm", "SL.randomForest", "SL.gam",
  "SL.polymars", "SL.mean")
test <- SuperLearner(Y = Y, X = X, newX = newX, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS")
test

# library with screening
SL.library <- list(c("SL.glmnet", "All"), c("SL.glm", "screen.randomForest",
  "All", "screen.SIS"), "SL.randomForest", c("SL.polymars", "All"), "SL.mean")
test <- SuperLearner(Y = Y, X = X, newX = newX, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS")
test
```

```
# binary outcome
set.seed(1)
N <- 200
X <- matrix(rnorm(N*10), N, 10)
X <- as.data.frame(X)
Y <- rbinom(N, 1, plogis(.2*X[, 1] + .1*X[, 2] - .2*X[, 3] +
  .1*X[, 3]*X[, 4] - .2*abs(X[, 4])))

SL.library <- c("SL.glmnet", "SL.glm", "SL.knn", "SL.gam", "SL.mean")

# least squares loss function
test.NNLS <- SuperLearner(Y = Y, X = X, SL.library = SL.library,
  verbose = TRUE, method = "method.NNLS", family = binomial())
test.NNLS

# negative log binomial likelihood loss function
test.NNloglik <- SuperLearner(Y = Y, X = X, SL.library = SL.library,
  verbose = TRUE, method = "method.NNloglik", family = binomial())
test.NNloglik

# 1 - AUC loss function
test.AUC <- SuperLearner(Y = Y, X = X, SL.library = SL.library,
  verbose = TRUE, method = "method.AUC", family = binomial())
test.AUC

# 2
# adapted from library(SIS)
set.seed(1)
# training
b <- c(2, 2, 2, -3*sqrt(2))
n <- 150
p <- 200
truerho <- 0.5
corrmat <- diag(rep(1-truerho, p)) + matrix(truerho, p, p)
corrmat[, 4] = sqrt(truerho)
corrmat[4, ] = sqrt(truerho)
corrmat[4, 4] = 1
cholmat <- chol(corrmat)
x <- matrix(rnorm(n*p, mean=0, sd=1), n, p)
x <- x
feta <- x[, 1:4]
fprob <- exp(feta) / (1 + exp(feta))
y <- rbinom(n, 1, fprob)

# test
m <- 10000
newx <- matrix(rnorm(m*p, mean=0, sd=1), m, p)
newx <- newx
newfeta <- newx[, 1:4]
newfprob <- exp(newfeta) / (1 + exp(newfeta))
newy <- rbinom(m, 1, newfprob)

DATA2 <- data.frame(Y = y, X = x)
```

```
newDATA2 <- data.frame(Y = newy, X=newx)

create.SL.knn <- function(k = c(20, 30)) {
  for(mm in seq(length(k))){
    eval(parse(text = paste('SL.knn.', k[mm], '<- function(..., k = ', k[mm],
      ') SL.knn(..., k = k)', sep = '')), envir = .GlobalEnv)
  }
  invisible(TRUE)
}
create.SL.knn(c(20, 30, 40, 50, 60, 70))

# library with screening
SL.library <- list(c("SL.glmnet", "All"), c("SL.glm", "screen.randomForest"),
  "SL.randomForest", "SL.knn", "SL.knn.20", "SL.knn.30", "SL.knn.40",
  "SL.knn.50", "SL.knn.60", "SL.knn.70",
  c("SL.polymars", "screen.randomForest"))
test <- SuperLearner(Y = DATA2$Y, X = DATA2[, -1], newX = newDATA2[, -1],
  SL.library = SL.library, verbose = TRUE, family = binomial())
test

## examples with multicore
set.seed(23432)
## training set
n <- 500
p <- 50
X <- matrix(rnorm(n*p), nrow = n, ncol = p)
colnames(X) <- paste("X", 1:p, sep="")
X <- data.frame(X)
Y <- X[, 1] + sqrt(abs(X[, 2] * X[, 3])) + X[, 2] - X[, 3] + rnorm(n)

## test set
m <- 1000
newX <- matrix(rnorm(m*p), nrow = m, ncol = p)
colnames(newX) <- paste("X", 1:p, sep="")
newX <- data.frame(newX)
newY <- newX[, 1] + sqrt(abs(newX[, 2] * newX[, 3])) + newX[, 2] - newX[, 3] + rnorm(m)

# generate Library and run Super Learner
SL.library <- c("SL.glm", "SL.randomForest", "SL.gam",
  "SL.polymars", "SL.mean")

testMC <- mcSuperLearner(Y = Y, X = X, newX = newX, SL.library = SL.library,
  method = "method.NNLS")
testMC

## examples with snow
library(parallel)
cl <- makeCluster(2, type = "PSOCK") # can use different types here
clusterSetRNGStream(cl, iseed = 2343)
testSNOW <- snowSuperLearner(cluster = cl, Y = Y, X = X, newX = newX,
  SL.library = SL.library, method = "method.NNLS")
testSNOW
stopCluster(cl)
```

```
## snow example with user-generated wrappers
# If you write your own wrappers and are using snowSuperLearner()
# These new wrappers need to be added to the SuperLearner namespace and exported to the clusters
# Using a simple example here, but can define any new SuperLearner wrapper
my.SL.wrapper <- function(...) SL.glm(...)
# assign function into SuperLearner namespace
environment(my.SL.wrapper) <-asNamespace("SuperLearner")

cl <- makeCluster(2, type = "PSOCK") # can use different types here
clusterSetRNGStream(cl, iseed = 2343)
clusterExport(cl, c("my.SL.wrapper"))  # copy the function to all clusters
testSNOW <- snowSuperLearner(cluster = cl, Y = Y, X = X, newX = newX,
  SL.library = c("SL.glm", "SL.mean", "my.SL.wrapper"), method = "method.NNLS")
testSNOW
stopCluster(cl)

## timing
replicate(5, system.time(SuperLearner(Y = Y, X = X, newX = newX,
  SL.library = SL.library, method = "method.NNLS")))

replicate(5, system.time(mcSuperLearner(Y = Y, X = X, newX = newX,
  SL.library = SL.library, method = "method.NNLS")))

cl <- makeCluster(2, type = 'PSOCK')
replicate(5, system.time(snowSuperLearner(cl, Y = Y, X = X, newX = newX,
  SL.library = SL.library, method = "method.NNLS")))
stopCluster(cl)


## End(Not run)
```

`SuperLearner.control` *Control parameters for the SuperLearner*

Description

Control parameters for the `SuperLearner`

Usage

```
SuperLearner.control(saveFitLibrary = TRUE, trimLogit = 0.001)
```

Arguments

`saveFitLibrary`	Logical. Should the fit for each algorithm be saved in the output from `SuperLearner`.
`trimLogit`	number between 0.0 and 0.5. What level to truncate the logit transformation to maintain a bounded loss function when using the NNloglik method.

Value

A list containing the control parameters.

`SuperLearner.CV.control` *Control parameters for the cross validation steps in* `SuperLearner`

Description

Control parameters for the cross validation steps in `SuperLearner`

Usage

```
SuperLearner.CV.control(V = 10L, stratifyCV = FALSE, shuffle = TRUE,
  validRows = NULL)
```

Arguments

`V`	Integer. Number of splits for the V-fold cross-validation step. The default is 10. In most cases, between 10 and 20 splits works well.
`stratifyCV`	Logical. Should the data splits be stratified by a binary response? Attempts to maintain the same ratio in each training and validation sample.
`shuffle`	Logical. Should the rows of `X` be shuffled before creating the splits.
`validRows`	A List. Use this to pass pre-specified rows for the sample splits. The length of the list should be `V` and each entry in the list should contain a vector with the row numbers of the corresponding validation sample.

Value

A list containing the control parameters

`SuperLearnerNews` *Show the NEWS file for the SuperLearner package*

Description

Show the NEWS file of the SuperLearner package. The function is simply a wrapper for the `RShowDoc` function

Usage

```
SuperLearnerNews(...)
SuperLearnerDocs(what = 'SuperLearnerR.pdf', ...)
```

Arguments

...	additional arguments passed to RShowDoc
what	specify what document to open. Currently supports the NEWS file and the PDF files 'SuperLearner.pdf' and 'SuperLearnerR.pdf'.

Value

A invisible character string given the path to the SuperLearner NEWS file

---

trimLogit *truncated-probabilities logit transformation*

Description

computes the logit transformation on the truncated probabilities

Usage

```
trimLogit(x, trim = 1e-05)
```

Arguments

x	vector of probabilities.
trim	value to truncate probabilities at. Currently symmetric truncation (trim and 1-trim).

Value

logit transformed values

Examples

```
x <- c(0.00000001, 0.0001, 0.001, 0.01, 0.1, 0.3, 0.7, 0.9, 0.99,
  0.999, 0.9999, 0.99999999)
trimLogit(x, trim = 0.001)
data.frame(Prob = x, Logit = qlogis(x), trimLogit = trimLogit(x, 0.001))
```

`write.method.template` *Method to estimate the coefficients for the super learner*

Description

These functions contain the information on the loss function and the model to combine algorithms

Usage

```
write.method.template(file = "", ...)

## a few built in options:
method.NNLS()
method.NNLS2()
method.NNloglik()
method.CC_LS()
method.CC_nloglik()
method.AUC(optim_method = "Nelder-Mead")
```

Arguments

`file`	A connection, or a character string naming a file to print to. Passed to `cat`.
`optim_method`	Passed to the `optim` call method. See `optim` for details.
`...`	Additional arguments passed to `cat`.

Details

A `SuperLearner` method must be a list (or a function to create a list) with exactly 3 elements. The 3 elements must be named `require`, `computeCoef` and `computePred`.

Value

A list containing 3 elements:

`require`	A character vector listing any required packages. Use `NULL` if no additional packages are required
`computeCoef`	A function. The arguments are: `Z`, `Y`, `libraryNames`, `obsWeights`, `control`, `verbose`. The value is a list with two items: `cvRisk` and `coef`. This function computes the coefficients of the super learner. As the super learner minimizes the cross-validated risk, the loss function information is contained in this function as well as the model to combine the algorithms in `SL.library`.
`computePred`	A function. The arguments are: `predY`, `coef`, `control`. The value is a numeric vector with the super learner predicted values.

Author(s)

Eric C Polley <eric.polley@nih.gov>

See Also

SuperLearner

Examples

```
write.method.template(file = '')
```

`write.screen.template` *screening algorithms for SuperLearner*

Description

Screening algorithms for `SuperLearner` to be used with `SL.library`.

Usage

```
write.screen.template(file = "", ...)
```

Arguments

`file`	A connection, or a character string naming a file to print to. Passed to `cat`.
`...`	Additional arguments passed to `cat`

Details

Explain structure of a screening algorithm here:

Value

`whichVariable`	A logical vector with the length equal to the number of columns in `X`. TRUE indicates the variable (column of X) should be included.

Author(s)

Eric C Polley <eric.polley@nih.gov>

See Also

SuperLearner

Examples

```
write.screen.template(file = '')
```

write.SL.template *Wrapper functions for prediction algorithms in SuperLearner*

Description

Template function for SuperLearner prediction wrappers and built in options.

Usage

```
write.SL.template(file = "", ...)
```

Arguments

file	A connection, or a character string naming a file to print to. Passed to cat.
...	Additional arguments passed to cat

Details

Describe SL.* structure here

Value

A list with two elements:

pred	The predicted values for the rows in newX.
fit	A list. Contains all objects necessary to get predictions for new observations from specific algorithm.

Author(s)

Eric C Polley <eric.polley@nih.gov>

See Also

SuperLearner

Examples

```
write.SL.template(file = '')
```

Index