# Package ‘ck37r’

June 3, 2017

**Title** Chris Kennedy's R Toolkit

**Version** 1.0.0

**Description** Toolkit for statistical, machine learning, and targeted learning analyses. Functionality includes loading & auto-installing packages, standardizing datasets, creating missingness indicators, imputing missing values, creating multicore or multinode clusters, automatic SLURM integration, enhancing SuperLearner and TMLE with automatic parallelization, and many other SuperLearner analysis & plotting enhancements.

**Depends** R (>= 3.1.0)

**Imports** caret, cvAUC, doParallel, doSNOW, foreach, ggplot2, methods, parallel, pryr, randomForest, RANN, reader, RhpcBLASctl, ROCR, snow, stringr, SuperLearner, tmle

**Suggests** doMC, glmnet, MASS, mlbench, testthat

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** Chris Kennedy [aut, cre]

**Maintainer** Chris Kennedy <chrisken@gmail.com>

**Repository** CRAN

**Date/Publication** 2017-06-03 17:55:34 UTC

## R topics documented:

---

## `cvsl_auc` *Calculate cross-validated AUC from CV.SuperLearner result*

---

### Description

Also calculates confidence interval. Based on initial code by Alan Hubbard.

### Usage

```
cvsl_auc(cvsl)
```

### Arguments

| | |
|---|---|
| `cvsl` | CV.SuperLearner object |

### Value

List with cvAUC and ci elements.

### References

LeDell, E., Petersen, M., & van der Laan, M. (2015). Computationally efficient confidence intervals for cross-validated area under the ROC curve estimates. Electronic journal of statistics, 9(1), 1583.

Polley EC, van der Laan MJ (2010) Super Learner in Prediction. U.C. Berkeley Division of Biostatistics Working Paper Series. Paper 226. http://biostats.bepress.com/ucbbiostat/paper266/

Sing, T., Sander, O., Beerenwinkel, N., & Lengauer, T. (2005). ROCR: visualizing classifier performance in R. Bioinformatics, 21(20), 3940-3941.

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007) Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml

## See Also

sl_auc_table cvsl_plot_roc ci.cvAUC

## Examples

```
library(SuperLearner)
library(ck37r)
data(Boston, package = "MASS")

set.seed(1, "L'Ecuyer-CMRG")

# Subset rows to speed up example computation.
row_subset = sample(nrow(Boston), 100)

Boston = Boston[row_subset, ]
X = subset(Boston, select = -chas)

cvsl = CV.SuperLearner(Boston$chas, X[, 1:2], family = binomial(),
                       cvControl = list(V = 2, stratifyCV = TRUE),
                       SL.library = c("SL.mean", "SL.glm"))
cvsl_auc(cvsl)
```

---

`cvsl_plot_roc` *Plot a ROC curve from cross-validated AUC from CV.SuperLearner*

---

## Description

Based on initial code by Alan Hubbard.

## Usage

```
cvsl_plot_roc(cvsl, Y = cvsl$Y,
  title = "CV-SuperLearner cross-validated ROC", digits = 4)
```

## Arguments

| | |
|---|---|
| `cvsl` | CV.SuperLearner object |
| `Y` | Outcome vector if not already included in the SL object. |
| `title` | Title to use in the plot. |
| `digits` | Digits to use when rounding AUC and CI for plot. |

## Value

List with the AUC plus standard error and confidence interval.

## References

LeDell, E., Petersen, M., & van der Laan, M. (2015). Computationally efficient confidence intervals for cross-validated area under the ROC curve estimates. Electronic journal of statistics, 9(1), 1583.

Polley EC, van der Laan MJ (2010) Super Learner in Prediction. U.C. Berkeley Division of Biostatistics Working Paper Series. Paper 226. http://biostats.bepress.com/ucbbiostat/paper266/

Sing, T., Sander, O., Beerenwinkel, N., & Lengauer, T. (2005). ROCR: visualizing classifier performance in R. Bioinformatics, 21(20), 3940-3941.

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007) Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml

## See Also

cvsl_auc sl_plot_roc ci.cvAUC

## Examples

```
library(SuperLearner)
library(ck37r)

data(Boston, package = "MASS")

set.seed(1, "L'Ecuyer-CMRG")

# Subset rows to speed up example computation.
row_subset = sample(nrow(Boston), 100)

Boston = Boston[row_subset, ]
X = subset(Boston, select = -chas)

cvsl = CV.SuperLearner(Boston$chas, X[, 1:2], family = binomial(),
                       cvControl = list(V = 2, stratifyCV = TRUE),
                       SL.library = c("SL.mean", "SL.glm"))
cvsl_plot_roc(cvsl)
```

---

## cvsl_weights *Create a table of meta-weights from a CV.SuperLearner*

---

## Description

Returns summary statistics (mean, sd, min, max) on the distribution of the weights assigned to each learner across SuperLearner ensembles. This makes it easier to understand the stochastic nature of the SL learner weights and to see how often certain learners are used. This function may eventually be moved into the SuperLearner package.

## Usage

```
cvsl_weights(cvsl, sort = T, nonzero = F, clean_names = T, rank = T,
  digits = 5)
```

## Arguments

| | |
|---|---|
| `cvsl` | CV.SuperLearner result object |
| `sort` | If TRUE, sort rows (learners) in descending order by mean weight. |
| `nonzero` | Restrict to learners with a non-zero mean weight. |
| `clean_names` | Remove "SL." from the front and "_All" from the end of learner names. |
| `rank` | Adding the learner rank to the table. |
| `digits` | Number of digits to round the results. Set to NULL to disable. |

## Value

Table in data frame form with each learner's mean, sd, min, and max meta-weight in the ensemble of each learner.

## References

Polley EC, van der Laan MJ (2010) Super Learner in Prediction. U.C. Berkeley Division of Biostatistics Working Paper Series. Paper 226. http://biostats.bepress.com/ucbbiostat/paper266/

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007) Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml

## Examples

```
library(SuperLearner)
library(ck37r)

data(Boston, package = "MASS")

set.seed(1, "L'Ecuyer-CMRG")

# Subset rows to speed up example computation.
row_subset = sample(nrow(Boston), 100)

Boston = Boston[row_subset, ]
X = subset(Boston, select = -chas)

cvsl = CV.SuperLearner(Boston$chas, X[, 1:2], family = binomial(),
                       cvControl = list(V = 2, stratifyCV = TRUE),
                       SL.library = c("SL.mean", "SL.glm"))
cvsl_weights(cvsl)
```

## gen_superlearner *Setup a SuperLearner() based on parallel configuration.*

### Description

Create custom SL and CV.SL functions that automatically use parallelization based on the provided configuration.

### Usage

```
gen_superlearner(parallel = "multicore", cluster = NULL,
  outer_cv_folds = NULL, verbose = T)
```

### Arguments

| | |
|---|---|
| `parallel` | Can be "multicore", "doSNOW"/"snow" or "seq"/NULL. |
| `cluster` | Optional cluster, e.g. from SNOW. |
| `outer_cv_folds` | How many folds to use for CV.SuperLearner |
| `verbose` | If TRUE will output additional details during execution. |

### Value

A list with a SL and CV.SL function.

## import_csvs *Import all CSV files in a given directory and save them to a list.*

### Description

Import all CSV files in a given directory and save them to a list.

### Usage

```
import_csvs(directory = "", file_pattern = "\\.csv$", recursive = T,
  verbose = T)
```

### Arguments

| | |
|---|---|
| `directory` | Directory to search for files. |
| `file_pattern` | File pattern to match. |
| `recursive` | Whether or not recurse into subdirectories, default T. |
| `verbose` | If True display additional information during execution. |

## Value

List with files; filenames are the names of the list elements (with extension removed).

## Examples

```
library(ck37r)

files = import_csvs("extdata")

names(files)
```

---

`impute_missing_values` *Impute missing values in a dataframe and add missingness indicators.*

---

## Description

Impute missing values, using knn by default or alternatively median-impute numerics, mode-impute factors. Add missingness indicators.

## Usage

```
impute_missing_values(data, type = "standard", add_indicators = T,
  prefix = "miss_", skip_vars = c(), verbose = F)
```

## Arguments

| | |
|---|---|
| `data` | Dataframe or matrix. |
| `type` | "knn" or "standard" (median/mode). NOTE: knn will result in the data being centered and scaled! |
| `add_indicators` | Add a series of missingness indicators. |
| `prefix` | String to add at the beginning of the name of each missingness indicator. |
| `skip_vars` | List of variable names to exclude from the imputation. |
| `verbose` | If True display extra information during execution. |

## Value

List with the following elements:

- $data - imputed dataset.
- $impute_info - if knn, caret preprocess element for imputing test data.
- $impute_values - if standard, list of imputation values for each variable.

## See Also

`missingness_indicators` `preProcess`

## Examples

```
# Load a test dataset.
data(PimaIndiansDiabetes2, package = "mlbench")

# Check for missing values.
colSums(is.na(PimaIndiansDiabetes2))

# Impute missing data and add missingness indicators.
# Don't impute the outcome though.
result = impute_missing_values(PimaIndiansDiabetes2, skip_vars = "diabetes")

# Confirm we have no missing data.
colSums(is.na(result$data))


#############
# K-nearest neighbors imputation

result2 = impute_missing_values(PimaIndiansDiabetes2, type = "knn", skip_vars = "diabetes")

# Confirm we have no missing data.
colSums(is.na(result2$data))
```

---

`load_all_code` *Load all R files in a library directory.*

---

## Description

Load all R files in a library directory.

## Usage

```
load_all_code(lib_dir = "lib", exclude_files = c("function_library.R"),
  file_pattern = "\\.R$", recursive = T, verbose = T)
```

## Arguments

| | |
|---|---|
| `lib_dir` | Directory contains the source code files. |
| `exclude_files` | Exclude a list of files; exclude function_library.R by default because we presume that is the main R library file. |
| `file_pattern` | Regular expression for files to load, defaults to *.R |
| `recursive` | If TRUE also recurse into subdirectories. |
| `verbose` | If TRUE display additional output during execution. |

## Examples

```
library(ck37r)

# Here R is a subdirectory with a bunch of .R files to load.
load_all_code("R")
```

---

`load_packages` *Load a list of packages.*

---

## Description

Load packages and install them from CRAN if they aren't already available.

## Usage

```
load_packages(pkgs = NULL, auto_install = F, update = F, verbose = F,
  ...)
```

## Arguments

| | |
|---|---|
| `pkgs` | Character vector of packages to load. |
| `auto_install` | Install any packages that could not be loaded. |
| `update` | Update packages where possible. |
| `verbose` | If T display more detailed information during execution. |
| `...` | Any additional parameters to pass through to install.packages() |

## Examples

```
# Load these 4 packages and install them if necessary.
load_packages(c("MASS", "SuperLearner", "tmle", "doParallel"), auto_install = TRUE)
```

## missingness_indicators

*Return matrix of missingness indicators for a dataframe or matrix.*

### Description

Return matrix of missingness indicators for a dataframe or matrix. Removes constant or collinear indicators.

### Usage

```
missingness_indicators(data, prefix = "miss_", remove_constant = T,
  remove_collinear = T, skip_vars = c(), verbose = F)
```

### Arguments

| | |
|---|---|
| `data` | Dataframe or matrix to analyze for missingness. |
| `prefix` | Name prefix for new indicator columns. |
| `remove_constant` | Remove any indicators that are all 0 or all 1. |
| `remove_collinear` | Remove any indicators that are collinear with each other. |
| `skip_vars` | Vector of variable names to skip. |
| `verbose` | If TRUE, print additional information. |

### Value

Matrix of missingness indicators

### See Also

`impute_missing_values`

### Examples

```
# Load a test dataset.
data(PimaIndiansDiabetes2, package = "mlbench")

# Check for missing values.
colSums(is.na(PimaIndiansDiabetes2))

# Generate missingness indicators; skip outcome variable.
indicators = missingness_indicators(PimaIndiansDiabetes2, skip_vars = "diabetes")

# Check missingness.
colSums(indicators)
```

## Mode — *Compute the mode of a vector (can be multiple results).*

### Description

Via http://stackoverflow.com/questions/2547402/is-there-a-built-in-function-for-finding-the-mode

### Usage

```
Mode(x)
```

### Arguments

| | |
|---|---|
| x | vector |

### Value

Vector of modal values in arbitrary order.

### Examples

```

library(ck37r)

data(Boston, package = "MASS")

table(Boston$chas)

Mode(Boston$chas)
```

## parallelize — *Setup parallel processing, either multinode or multicore.*

### Description

By default it uses a multinode cluster if available, otherwise sets up multicore via doMC. Libraries required: parallel, doSNOW, doMC, RhpcBLASctl, foreach

### Usage

```
parallelize(type = "any", max_cores = NULL, allow_multinode = T,
  machine_list = Sys.getenv("SLURM_NODELIST"),
  cpus_per_node = as.numeric(Sys.getenv("SLURM_CPUS_ON_NODE")),
  outfile = "", verbose = F)
```

### Arguments

| | |
|---|---|
| `type` | "any", "cluster"/"doSNOW", "doParallel", "doMC", or "seq" |
| `max_cores` | Restrict to this many cores, even if more are available. |
| `allow_multinode` | If T will use multiple nodes if detected. If F will not use multiple machines even if they are available. |
| `machine_list` | List of networked computers for multinode computation. |
| `cpus_per_node` | Number of processes to run on each node, if using multinode parallelization. |
| `outfile` | File to collect output across workers. IF "" then results are printed to the console. |
| `verbose` | If TRUE display additional output during execution. |

### Value

obj Cluster object that can be passed to stop_cluster().

### See Also

stop_cluster

## `plot.SuperLearner` *Plot estimated risk and confidence interval for each learner*

### Description

Does not include SuperLearner or Discrete SL results as that requires CV.SuperLearner to estimate the standard errors.

### Usage

```
## S3 method for class 'SuperLearner'
plot(x, Y = x$Y, constant = qnorm(0.975),
  sort = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `x` | SuperLearner result object |
| `Y` | Outcome vector |
| `constant` | Multiplier of the standard error for confidence interval construction. |
| `sort` | If TRUE re-orders the results by risk estimate. |
| `...` | Any remaining arguments (unused). |

### Value

plot object; print to display.

### References

Polley EC, van der Laan MJ (2010) Super Learner in Prediction. U.C. Berkeley Division of Biostatistics Working Paper Series. Paper 226. http://biostats.bepress.com/ucbbiostat/paper266/

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007) Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml

### See Also

SuperLearner

### Examples

```
library(SuperLearner)
library(ck37r)

data(Boston, package = "MASS")

set.seed(1)
sl = SuperLearner(Boston$medv, subset(Boston, select = -medv), family = gaussian(),
                  SL.library = c("SL.mean", "SL.glmnet"))

sl
plot(sl, Y = Boston$chas)
```

---

## rf_count_terminal_nodes

*Count the terminal nodes in each tree from a random forest*

---

### Description

Returns a vector of terminal node counts for each tree in a random forest. The distribution of terminal node counts is helpful when seeking to optimize the maxnodes hyperparameter of the random forest. By default RF allows very large trees, which may result in overfitting. Optimizing the number of terminal nodes in a random forest is a more direct way of requiring simpler trees than the minimum node size hyperparameter.

### Usage

```
rf_count_terminal_nodes(rf)
```

### Arguments

| | |
|---|---|
| rf | Random Forest object |

## Value

vector of terminal node counts

## References

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

## See Also

getTree randomForest

## Examples

```
library(SuperLearner)
library(ck37r)

data(Boston, package = "MASS")

set.seed(1)

# Downsample to 100 observations speed up example.
Boston = Boston[sample(nrow(Boston), 100L), ]

sl = SuperLearner(Boston$medv, subset(Boston, select = -medv), family = gaussian(),
                  cvControl = list(V = 3),
                  SL.library = c("SL.mean", "SL.glmnet", "SL.randomForest"))

sl

summary(rf_count_terminal_nodes(sl$fitLibrary$SL.randomForest_All$object))

max_terminal_nodes = max(rf_count_terminal_nodes(sl$fitLibrary$SL.randomForest_All$object))

max_terminal_nodes

# Now run create.Learner() based on that maximum.

# It is often handy to convert to log scale of a hyperparameter before testing a ~linear grid.
# NOTE: -0.7 ~ 0.69 ~ log(0.5) which is the multiplier that yields sqrt(max)
maxnode_seq = unique(round(exp(log(max_terminal_nodes) * exp(c(-0.97, -0.7, -0.45, -0.15, 0)))))
maxnode_seq

rf = SuperLearner::create.Learner("SL.randomForest", detailed_names = TRUE, name_prefix = "rf",
                          params = list(ntree = 100), # fewer trees for testing speed only.
                                tune = list(maxnodes = maxnode_seq))

sl = SuperLearner(Boston$medv, subset(Boston, select = -medv), family = gaussian(),
                  cvControl = list(V = 3),
                  SL.library = c("SL.mean", "SL.glmnet", rf$names))

sl
```

## `setup_parallel_tmle` *Setup TMLE to run in parallel*

### Description

Starts a cluster and sets up TMLE and SuperLearner to use the cluster so that TMLE is conducted in parallel rather than using only one core.

### Usage

```
setup_parallel_tmle(parallel = "multicore", max_cores = NULL,
  allow_multinode = T, env = .GlobalEnv)
```

### Arguments

| | |
|---|---|
| `parallel` | "multicore", "doParallel", or "doSNOW" |
| `max_cores` | Restrict how many many cores will be used on a machine, rather than using all available cores. Useful if each core needs to use a substantial amount of memory. |
| `allow_multinode` | If T, will create a multinode cluster if it finds multiple machines listed in the "SLURM_NODELIST" environmental variable. If F, it will only use the current node even if multiple nodes are detected. |
| `env` | Environment in which to save the functions, defaulting to the global environment. Set to NULL to disable. |

### See Also

parallelize, tmle_parallel, gen_superlearner

## `sl_auc_table` *Table of cross-validated AUCs from SuperLearner result*

### Description

Calculates cross-validated AUC for each learner in the SuperLearner. Also calculates standard-error, confidence interval and p-value. Based on initial code by Alan Hubbard.

### Usage

```
sl_auc_table(sl, Y = sl$Y, sort = T)
```

## Arguments

| | |
|---|---|
| `sl` | CV.SuperLearner object |
| `Y` | Outcome vector, if not already added to SL object. |
| `sort` | Sort table by order of AUC. |

## Value

Dataframe table with auc, se, ci, and p-value (null hypothesis = 0.5).

## References

LeDell, E., Petersen, M., & van der Laan, M. (2015). Computationally efficient confidence intervals for cross-validated area under the ROC curve estimates. Electronic journal of statistics, 9(1), 1583.

Polley EC, van der Laan MJ (2010) Super Learner in Prediction. U.C. Berkeley Division of Biostatistics Working Paper Series. Paper 226. http://biostats.bepress.com/ucbbiostat/paper266/

Sing, T., Sander, O., Beerenwinkel, N., & Lengauer, T. (2005). ROCR: visualizing classifier performance in R. Bioinformatics, 21(20), 3940-3941.

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007) Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml

## See Also

`cvsl_auc` `sl_plot_roc` `ci.cvAUC`

## Examples

```
library(SuperLearner)
library(ck37r)

data(Boston, package = "MASS")

set.seed(1)
sl = SuperLearner(Boston$chas, subset(Boston, select = -chas), family = binomial(),
                  SL.library = c("SL.mean", "SL.glmnet"))

sl_auc_table(sl, Y = Boston$chas)
```

---

# `sl_plot_roc` *Plot a ROC curve from cross-validated AUC from SuperLearner*

---

## Description

Plots the ROC curve for a single learner from a SuperLearner object, defaulting to the minimum estimated risk learner. Based on code by Alan Hubbard.

## Usage

```
sl_plot_roc(sl, Y = sl$Y, learner = which.min(sl$cvRisk),
  title = paste0("SuperLearner cross-validated ROC: ",
  names(sl$cvRisk)[learner]), digits = 4)
```

## Arguments

| | |
|---|---|
| `sl` | SuperLearner object |
| `Y` | Outcome vector if not already included in the SL object. |
| `learner` | Which learner to plot - defaults to minimum risk learner. |
| `title` | Title to use in the plot. |
| `digits` | Digits to use when rounding AUC and CI for plot. |

## Value

List with plotted AUC & CI, table of AUC results for all learners, and the name of the best learner.

## References

LeDell, E., Petersen, M., & van der Laan, M. (2015). Computationally efficient confidence intervals for cross-validated area under the ROC curve estimates. Electronic journal of statistics, 9(1), 1583.

Polley EC, van der Laan MJ (2010) Super Learner in Prediction. U.C. Berkeley Division of Biostatistics Working Paper Series. Paper 226. http://biostats.bepress.com/ucbbiostat/paper266/

Sing, T., Sander, O., Beerenwinkel, N., & Lengauer, T. (2005). ROCR: visualizing classifier performance in R. Bioinformatics, 21(20), 3940-3941.

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007) Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml

## Examples

```
library(SuperLearner)
library(ck37r)

data(Boston, package = "MASS")

set.seed(1)
sl = SuperLearner(Boston$chas, subset(Boston, select = -chas),
                  family = binomial(), SL.library = c("SL.mean", "SL.glm"),
                  cvControl = list(V = 2))

sl

sl_plot_roc(sl, Y = Boston$chas)
```

## sl_stderr *Calculate the SE of individual SL learners*

### Description

This will help understand risk estimates of learners in SL, similar to CV.SL.

### Usage

```
sl_stderr(sl, Y, obsWeights = rep(1, length(Y)))
```

### Arguments

| | |
|---|---|
| sl | SuperLearner result object |
| Y | Outcome vector |
| obsWeights | Observation weights |

### Value

Vector of the standard errors of the risk estimate for each learner in the SL object.

### References

Dudoit, S., & van der Laan, M. J. (2005). Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. Statistical Methodology, 2(2), 131-154.

Polley EC, van der Laan MJ (2010) Super Learner in Prediction. U.C. Berkeley Division of Biostatistics Working Paper Series. Paper 226. http://biostats.bepress.com/ucbbiostat/paper266/

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007) Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml

### See Also

plot.SuperLearner summary.CV.SuperLearner

### Examples

```
library(SuperLearner)
library(ck37r)

data(Boston, package = "MASS")

set.seed(1)
sl = SuperLearner(Boston$medv, subset(Boston, select = -medv),
                  family = gaussian(), cvControl = list(V = 2),
                  SL.library = c("SL.mean", "SL.glm"))
```

```
sl

sl_stderr(sl, Y = Boston$medv)
```

---

standardize *Rescale variables, possibly excluding some columns*

---

## Description

Simple extension to base::scale() to skip columns.

## Usage

```
standardize(x, skip_vars = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | Dataframe or matrix, assumed to have column names. |
| `skip_vars` | List of names of variables not to scale. |
| `...` | Extra arguments passed-through to base::scale |

## Value

Data-frame with appropriate variables scaled.

## See Also

`scale`

## Examples

```
library(ck37r)

data(Boston, package = "MASS")

# Don't scale our outcome variable.
data = standardize(Boston, skip_vars = "medv")

summary(data)
```

| | |
|---|---|
| `stop_cluster` | *Stop the cluster if snow::makeCluster() was used, but nothing needed if doMC was used.* |

### Description

Stop the cluster if snow::makeCluster() was used, but nothing needed if doMC was used.

### Usage

```
stop_cluster(cluster_obj)
```

### Arguments

| | |
|---|---|
| `cluster_obj` | Cluster object generated by parallelize() |

| | |
|---|---|
| `tmle_parallel` | *Modify TMLE to support parallel computation for g and Q.* |

### Description

This is needed to use Savio or any multicore system effectively.

Another benefit is that the SuperLearner objects for the Q and g estimation are saved. This allows one to examine the risk estimates for example.

### Usage

```
tmle_parallel(Y, A, W, family, g.SL.library, Q.SL.library, id = 1:length(Y),
  verbose = F, V = 5, sl_fn = SuperLearner::SuperLearner,
  cvsl_fn = SuperLearner::CV.SuperLearner, cvQinit = F,
  conserve_memory = T, ...)
```

### Arguments

| | |
|---|---|
| `Y` | Outcome |
| `A` | Treatment indicator |
| `W` | Covariates |
| `family` | Gaussian or binomial |
| `g.SL.library` | SL library for estimating g |
| `Q.SL.library` | SL library for estimating Q |
| `id` | Optional list of subject-specific ids. |
| `verbose` | If TRUE outputs additional information during execution. |

| | |
|---|---|
| `V` | Number of cross-validation folds to use when estimating g and Q. Defaults to 5 as tmle package does. |
| `sl_fn` | SuperLearner function to use for estimation of g and possibly Q. By default this uses the normal SuperLearner function which is sequential. Other options would be to pass in mcSuperLearner, snowSuperLearner, or CV.SuperLearner. For functions that require additional arguments (e.g. the cluster argument of for snowSuperLearner) one should create a new function that overloads the call and sets that argument. This is what setup_parallel_tmle() does. |
| `cvsl_fn` | CV.SuperLearner equivalent, can be used for estimating Q. |
| `cvQinit` | If T, estimate Q using cvsl_fn, otherwise use sl_fn. |
| `conserve_memory` | If T, remove the fitLibrary elements to save memory after we have done the relevant prediction. |
| `...` | Remaining arguments are passed through to tmle::tmle(). |

## See Also

setup_parallel_tmle

# Index