Package: MLSurvey (via r-universe)

November 18, 2026

Title Machine Learning for Complex Survey Data

Version 1.0.2

Description Fits weighted machine learning models for complex survey data using a replicate weights approach that accounts for sampling weights, strata, and clusters. Weighted versions of Elastic Net, XGBoost, and random forest are provided for linear and logistic regression, along with parallelized tuning-parameter search and companion plotting methods.

License GPL (>= 3)

Encoding UTF-8

Imports glmnet, ggplot2, graphics, grDevices, parallel, randomForest, rlang, stats, survey, svyROC, utils, xgboost

Suggests Matrix, knitr, rmarkdown, RhpcBLASctl, patchwork, testthat (>= 3.0.0)

Config/testthat/edition 3

LazyData true

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

NeedsCompilation no

Author Hyeonju Kim [aut, cre] (ORCID: <https://orcid.org/0000-0002-2368-4850>)

Maintainer Hyeonju Kim <hyeonjukm01@gmail.com>

Depends R (>= 3.5.0)

Repository https://cran.r-universe.dev

Date/Publication 2026-11-18 09:50:21 UTC

RemoteUrl https://github.com/cran/MLSurvey

RemoteRef HEAD

RemoteSha 27c8fed7683d5827bd7f61565a65e95e73721f7f

Contents

autoplot.par.w.elnet . . . 2
autoplot.w.elnet . . . 4
ci_wauc . . . 5
eval_loss . . . 8
evalerror_bin . . . 9
evalerror_reg . . . 9
generate_wxgb_params . . . 10
nhanes2013_sbc . . . 11
optim_wxgb_para . . . 12
par_wElnet . . . 15
plot.par.w.elnet . . . 18
plot.w.elnet . . . 20
print.w.elnet . . . 21
replicate_weights . . . 22
wElnet . . . 25
wRandomforest . . . 28
wXGBoost . . . 32

Index **38**

`autoplot.par.w.elnet` *ggplot2 plotting method for* `par.w.elnet` *objects*

Description

A ggplot2-based counterpart to `plot.par.w.elnet()` which visualizes the results of `par_wElnet()`: the weighted error curve across the lambda grid for each alpha, and/or a summary of the best (minimum) error achieved by each alpha, with the overall best (alpha, lambda) combination highlighted.

```
Registered as an \code{\link[ggplot2]{autoplot}} method, so it dispatches
automatically via \code{autoplot(en.par)} once \code{ggplot2} is loaded,
without masking the base-\code{R} \code{plot()} method in [plot.par.w.elnet()].
```

Usage

```
## S3 method for class 'par.w.elnet'
autoplot(
  object,
  type = c("both", "profile", "summary"),
  log.lambda = TRUE,
  facet = FALSE,
  palette = "viridis",
  origin = TRUE,
  main.profile = NULL,
  xlab.profile = NULL,
  ylab.profile = NULL,
```

```
  main.summary = NULL,
  xlab.summary = NULL,
  ylab.summary = NULL,
  ...
)
```

Arguments

`object`	An object of class `par.w.elnet`, the output of `par_wElnet()`.
`type`	A string indicating what to plot. One of: • `"profile"`: error vs. `log(lambda)`, one curve per alpha, faceted or overlaid depending on `facet`. • `"summary"`: minimum error achieved vs. alpha, one point per alpha, with the overall best alpha highlighted. • `"both"` (default): both plots combined into one figure (requires `patchwork`; see Details).
`log.lambda`	A logical value. If `TRUE` (default), the x-axis of the `"profile"` plot is `log(lambda)`, matching `glmnet` convention. If `FALSE`, lambda is plotted on its original scale.
`facet`	A logical value. If `FALSE` (default), all alphas are overlaid on a single panel in the `"profile"` plot, colored by alpha. If `TRUE`, each alpha gets its own facet panel (useful when there are many alphas and overlaid curves are hard to distinguish).
`palette`	A string naming a `viridisLite`/`viridis`-style option passed to `ggplot2::scale_color_viridis_d()`, or `NULL` to use ggplot2's default discrete color scale. Default is `"viridis"`.
`origin`	A logical value. If `TRUE` (default), solid black reference lines are drawn at x=0 and y=0, and the plot's coordinate range is expanded if needed so the origin is always visible. If `FALSE`, no origin lines are drawn and the axes are scaled to the data range as usual.
`main.profile, xlab.profile, ylab.profile`	Optional strings overriding the title and axis labels of the `"profile"` panel. If `NULL` (default), sensible defaults are used ("Error profile by alpha", "log(lambda)" or "lambda", and "Weighted error").
`main.summary, xlab.summary, ylab.summary`	Optional strings overriding the title and axis labels of the `"summary"` panel. If `NULL` (default), sensible defaults are used ("Best error by alpha", "alpha", and "Minimum weighted error").
`...`	Currently unused; included for S3 consistency with `autoplot()`.

Details

When `type = "both"`, the two panels are combined side by side using the `patchwork` package. If `patchwork` is not installed, `autoplot()` falls back to returning a list of the two individual `ggplot` objects (`$profile` and `$summary`) instead of a single combined plot, along with a message explaining how to get the combined figure.

Since the return value(s) are ordinary `ggplot` objects, further customization beyond the `main.*`/`xlab.*`/`ylab.*` arguments (e.g. themes, color scales, facet arguments) can always be added afterward with `+`, e.g. `autoplot(en.par, type = "profile") + ggplot2::theme_bw()`.

Value

If type is "profile" or "summary", a single ggplot object. If type = "both" and patchwork is installed, a combined patchwork object. If type = "both" and patchwork is not installed, a named list of two ggplot objects: $profile and $summary.

Examples

```
library(MLSurvey)
library(ggplot2)
data(nhanes2013_sbc)
alpha <- c(0.000, 0.001, 0.008, 0.027, 0.064, 0.125, 0.216, 0.343, 0.512, 0.729, 1.000)

en.par <- par_wElnet(alpha = alpha,
                     data = nhanes2013_sbc,
                     col.y = "HBP", col.x = 2:61,
                     family = "binomial",
                     cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
                     method = "dCV", k = 10, R = 20)

autoplot(en.par)                       # both panels (needs patchwork)
autoplot(en.par, type = "profile")     # error-vs-lambda, overlaid by alpha
autoplot(en.par, type = "profile", facet = TRUE)  # one panel per alpha
autoplot(en.par, type = "summary")     # best-error-vs-alpha
autoplot(en.par, type = "profile", main.profile = "My custom title", xlab.profile = "Penalty")
```

---

autoplot.w.elnet *ggplot2 plotting method for* w.elnet *objects (single-alpha* wElnet() *output)*

Description

A ggplot2-based counterpart to plot.w.elnet() that visualizes the weighted error curve across the lambda grid for a single wElnet fit (i.e. a single, fixed alpha), marking the selected lambda.min. This is the single-alpha counterpart to autoplot.par.w.elnet(), which handles the multi-alpha output of par_wElnet().

Registered as an \code{\link[ggplot2]{autoplot}} method, so it dispatches automatically via \code{autoplot(en.dcv)} once \code{ggplot2} is loaded.

Usage

```
## S3 method for class 'w.elnet'
autoplot(
  object,
  log.lambda = TRUE,
  origin = TRUE,
```

```
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

`object`	An object of class `w.elnet`, the output of `wElnet`.
`log.lambda`	A logical value. If `TRUE` (default), the x-axis is `log(lambda)`, matching `glmnet` convention. If `FALSE`, lambda is plotted on its original scale.
`origin`	A logical value. If `TRUE` (default), solid black reference lines are drawn at x=0 and y=0, and the plot's coordinate range is expanded if needed so the origin is always visible. If `FALSE`, no origin lines are drawn and the axes are scaled to the data range as usual.
`main`, `xlab`, `ylab`	Optional strings overriding the plot title and axis labels. If `NULL` (default), sensible defaults are used ("Error profile (alpha = ...)", "log(lambda)" or "lambda", and "Weighted error").
`...`	Currently unused; included for S3 consistency with `autoplot()`.

Value

A single `ggplot` object.

Examples

```
library(MLSurvey)
library(ggplot2)
data(nhanes2013_sbc)

en.dcv <- wElnet(data = nhanes2013_sbc,
                 col.y = "HBP", col.x = 2:61,
                 family = "binomial", alpha = 0.729,
                 cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
                 method = "dCV", k = 10, R = 20)

autoplot(en.dcv)
autoplot(en.dcv, main = "My custom title", xlab = "Penalty", ylab = "Error")
```

`ci_wauc` *CI for weighted AUC (wAUC) by replicate weights*

Description

A function to estimate a confidence interval (CI) for wAUC implemented by `wROC::wauc()` with replicate weights by `survey::as.svrepdesign()`; `bootstrap` is Canty and Davison's bootstrap, `subbootstrap` is Rao and Wu's (n-1) bootstrap, and `JK1` and `JKn` are jackknife methods.

Usage

```
ci_wauc(
  y,
  yhat,
  tag.event = "1",
  tag.nonevent = "0",
  weights = NULL,
  cluster = NULL,
  strata = NULL,
  design = NULL,
  data = NULL,
  type = c("JK1", "JKn", "bootstrap", "subbootstrap"),
  B = 2000,
  conf.level = 0.95,
  fpc = NULL,
  fpctype = NULL,
  ...,
  compress = TRUE,
  mse = getOption("survey.replicates.mse")
)
```

Arguments

y	A vector of response variable or a string of its name (requires data to be supplied).
yhat	A vector of predicted probabilities of y or a string of its name (requires data to be supplied).
tag.event	A character of the event of interest in y, e.g. disease, or condition. Default is "1".
tag.nonevent	A character of the non-event in y, e.g. non-disease, or non-condition. Default is "0".
weights	A character string of a sampling weights' identifier. NULL for either a unweighted case or the sampling design inserted in the design argument.
cluster	A character string of a cluster identifier. It could be NULL if the sampling design were plugged in the design argument.
strata	A character string of a strata identifier. It could be NULL if the sampling design were plugged in the design argument.
design	An object of class survey.design generated by survey::svydesign(). It could be NULL if information about cluster, strata, weights and data were given.
data	A data frame including vectors of y, yhat, and (or) weights. cluster,strata can be included depending on the complex sampling structure. weights = NULL assumes equal probability generating 1's. If data=NULL, then specific numerical vectors must be included in y, yhat, or the sampling design should be plugged into the argument design.

type	Type of replicate weights. Choose one of "JK1", "JKn", "bootstrap", or "subbootstrap". Note: JKn for stratified, JK1 for unstratified designs.
B	The number of bootstrap replicates. Default is 2000.
conf.level	The width of confidence interval. Default is 0.95.
fpc, fpctype	Passed to jk1weights, jknweights, brrweights, bootweights, subbootweights, or mrbweights.
...	optional parameters to be passed to the low level function survey::as.svrepdesign().
compress	Use a compressed representation of the replicate weights matrix.
mse	if TRUE, compute variances from sums of squares around the point estimate, rather than the mean of the replicates.

Value

confidence interval (CI) estimates

See Also

svyROC::wauc() for detailed information on the implementation.

survey::as.svrepdesign() for detailed information on replicate weights methods.

Examples

```
data(nhanes2013_sbc)

# For ad hoc adjustment for a single PSU stratum not to contribute to the variance
options(survey.lonely.psu='remove')

# yhat would typically come from a fitted model (e.g. wElnet(), wXGBoost(),
# wRandomforest(), or any other classifier). Here we generate one directly
# from a simple weighted logistic regression for a self-contained example.
fit <- glm(HBP ~ INDFMPIR + BMXBMI + BMXARML + BMXLEG + BPXPLS,
          data = nhanes2013_sbc,
          family = binomial(), weights = nhanes2013_sbc$WTSAF2YR)
yhat <- predict(fit, type = "response")

# Conveniently, update a dataset including yhat (a vector) estimated
# from any machine learning methods. HBP, SDMVPSU, SDMVSTRA, and WTSAF2YR
# are referenced by name (not position) to avoid any risk of selecting
# the wrong columns.
data1 <- data.frame(
  nhanes2013_sbc[, c("HBP", "SDMVPSU", "SDMVSTRA", "WTSAF2YR")],
  yhat
)

# For estimating bootstrap 95\% CI for a model selected by dCV
wAUC.ci<- ci_wauc("HBP", "yhat",weights="WTSAF2YR",cluster="SDMVPSU",strata="SDMVSTRA",
                  data=data1,type="subbootstrap",B=2000)
#Or equivalently,
```

```
des <- survey::svydesign(ids=~SDMVPSU, strata = ~SDMVSTRA, weights = ~WTSAF2YR,
                                nest = TRUE, data = nhanes2013_sbc)

wAUC.ci<- ci_wauc(des$variables$HBP,data1$yhat,design = des,type = "subbootstrap",B=2000)
```

`eval_loss` *Weighted mean of log loss (binary cross-entropy loss) for complex survey data*

Description

A function to evaluate XGBoost models by weighted mean log loss for logistic regression. This function estimates predicted probabilities from the predicted binary outcomes and computes the weighted mean of log likelihood values for Bernoulli distribution. This function is a customized error function for complex survey data.

Usage

```
eval_loss(preds, dtrain)
```

Arguments

`preds`	Predicted binary values (`0`/`1`) from the model.
`dtrain`	An object of `xgb.DMatrix` including `label`(response variable or binary outcome), `data`,`weight`.

Value

A list of two components:

- `metric`: A string labelled as `weighted.mean.logloss`.
- `value` : A numeric value of weighted mean of log loss.

See Also

`xgboost::xgb.train()` for relevant arguments and return values in detail.

`wXGBoost()` for detailed information on implementation.

`optim_wxgb_para()` for detailed optimization and examples.

evalerror_bin *Weighted mean error of binary outcome for complex survey data*

Description

A function to evaluate XGBoost models by weighted mean error for binary outcome (`0/1`). This function is a customized error function to plug in the function `wXGBoost()` for complex survey data.

Usage

```
evalerror_bin(preds, dtrain)
```

Arguments

`preds`	Predicted binary values (`0/1`) from the model.
`dtrain`	An object of `xgboost::xgb.DMatrix` including `label`(response variable or binary outcome), `data`, and `weight`.

Value

A list of two components:

- `metric` : A string labelled as `weighted.error`.
- `value` : A numeric value of weighted mean error.

See Also

`xgboost::xgb.train()` for relevant arguments and return values in detail.

`wXGBoost()` for detailed information on implementation.

`optim_wxgb_para()` for detailed optimization and examples.

evalerror_reg *Weighted mean squared error for complex survey data*

Description

A function to evaluate XGBoost models by weighted mean squared error for linear regression. This function is a customized error function for complex survey data.

Usage

```
evalerror_reg(preds, dtrain)
```

Arguments

preds	Predicted values from the model
dtrain	An object of xgboost::xgb.DMatrix including label(response variable), data, and weight.

Value

A list of two components:

- metric: A string labelled as weighted.mse.
- value : A numeric value of weighted mean squared error (MSE).

See Also

xgboost::xgb.train() for relevant arguments and return values in detail.

wXGBoost() for detailed information on implementation.

optim_wxgb_para() for detailed optimization and examples.

---

generate_wxgb_params *Generate randomized wXGBoost hyperparameters for tuning*

Description

A helper function creating a complete list of valid wXGBoost hyperparameters. It combines core structural configurations, regularization parameters, and tree-building criteria. By default, it sets up a diverse random search space, but users can inject fixed overrides or custom sampling closures.

Usage

```
generate_wxgb_params(custom_space = list())
```

Arguments

custom_space	A named list of values or functions to override defaults. The list can contain static values (e.g., subsample = 0.8) or anonymous functions that return a single value when called (e.g., function() sample(3:10, 1)). Default objective is binary:logistic for binary classification; reg:squarederror is set for linear regression. The default search space is defined as follows:

```
defaults <- list(
  # General Parameters
  objective           = "binary:logistic",
  nthread             = 1,
  # Tree Booster Hyperparameters
  eta                 = runif(1, 0.01, 0.3),
  gamma               = runif(1, 0.0, 10.0),
  max_depth           = sample(3:10, 1),
```

```
  min_child_weight    = sample(1:20, 1),
max_delta_step    = 0,          # for very imbalanced case : (1:10) helps
  subsample           = runif(1, 0.5, 1.0),
  colsample_bytree    = runif(1, 0.5, 1.0),
  colsample_bylevel   = 1.0,
  colsample_bynode    = 1.0,
  alpha             = runif(1, 0.0, 1.0), # L1 regularization
  lambda            = runif(1, 1.0, 4.0), # L2 regularization
  tree_method         = "auto",
  scale_pos_weight    = 1.0,
  # Random Seed
  seed                = sample.int(100000, 1)
  )
```

Value

A named list of parameters ready to be parsed by `wXGBoost()`.

Note

The function is encapsulated in `optim_wxgb_para()` to dynamically create a (hyper)parameter set for `wXGBoost()` per iteration for optimzation. This function can also be used for `xgboost::xgb.train()` or `xgboost::xgb.cv()`.

Examples

```
# Generate using default search spaces
para.default<-generate_wxgb_params()

# Force a static max_depth and pass a custom eta sampler
custom.para<- generate_wxgb_params(list(
  max_depth = 6,
  eta = function() runif(1, 0.05, 0.15)
))
```

---

nhanes2013_sbc *2013-2014 cycle of NHANES data*

Description

The data was extracted from The National Center for Health Statistics (NCHS) in the Centers for Diseases Control and Prevention (CDC) and has been processed to a sparse matrix for memory efficiency.

Usage

```
data(nhanes2013_sbc)
```

Format

An object of class `data.frame` with 1510 obs. of 66 variables including

HBP Binary response variable labeled as 1 for hypertension or high blood pressure.

SEQN Respondent sequence number.

SDDSVYR Data Release Number indicating survey year.

SDMVPSU Masked Variance Pseudo-PSU, `cluster`.

SDMVSTRA Masked Variance Pseudo-Stratum, `strata`.

WTSAF2YR Fasting Subsample 2-year MEC Weight, `weights`.

Source

https://www.cdc.gov/nchs/nhanes/about_nhanes.htm

Examples

```
data(nhanes2013_sbc)
```

`optim_wxgb_para` *Optimal (hyper-) parameter search for weighted XGBoost (wXGBoost) for complex survey data*

Description

A function to automate an optimal search for (hyper-) parameter via random sampling and to perform cross validation by replicate weights methods, `wXGBoost`. The final model is not estimated (`final.model=FALSE`).

Usage

```
optim_wxgb_para(
  y,
  col.x,
  custom_space = list(),
  nitr = 50,
  nfolds = 10,
  R = 1,
  B = 200,
  nRounds = 10000,
  nstop = 5,
  seed = 120,
 method = c("dCV", "JKn", "bootstrap", "subbootstrap", "BRR", "split", "extrapolation"),
  train.prob = 0.7,
  method.split = c("dCV", "bootstrap", "subbootstrap"),
  missing = NA,
  maximize = FALSE,
```

```
  cluster = NULL,
  strata = NULL,
  weights = NULL,
  design = NULL,
  data = NULL,
  print_every_n = 50L,
  ...
)
```

Arguments

y	A numeric vector of response variable.
col.x	A numeric vector indicating indices of the covariates or a string vector indicating these column names.
custom_space	A named list passed downwards to modify structural parameters or samplers. Defaults to an empty `list()`. See also `generate_wxgb_params()`. The complete list of parameters is found on `xgboost::xgb.train()` or https://xgboost.readthedocs.io/en/latest/parameter.html.
nitr	The number of iterations in the `for`-loop to search for optimal (hyper-) parameters. Default is `50`.
nfolds	The number of folds to be defined for `dCV` method. Default is `nfolds=10`.
R	The number of times the sample is partitioned for the `dCV`, `split`, or `extrapolation` methods. Default is `R=1`.
B	The number of bootstrap re-samples for `bootstrap` and `subbootstrap` methods. Default is `B=200`, matching `wXGBoost`'s own default.
nRounds	The maximum number of boosting iterations.
nstop	`NULL` does not trigger the early stopping function. Setting to an integer k will stop training with a validation set unless the performance improve for k rounds. Setting this parameter engages the `cb.early.stop` callback.
seed	An integer to set the seed for random parameter reproducibility. This is not for `wXGBoost()` implementation but for `generate_wxgb_params()`.
method	A character string indicating a method of replicate weights. Choose one of these: `JKn`, `dCV`, `bootstrap`, `subbootstrap`, `BRR`, `split`, `extrapolation`.
train.prob	A numeric between 0 and 1, indicating the proportion of clusters (for the method `split`) or strata (for the method `extrapolation`) to be set in the training sets. Default is `train.prob = 0.7`, matching `wXGBoost`'s own default. Only applies to `split` and `extrapolation` methods.
method.split	A string of one the following replicate weights methods to be implemented under the `split` in the `method` argument: `dCV`, `bootstrap` or `subbootstrap`. Default is `"dCV"`, matching `wXGBoost`'s own default. Only applies to the `split` method.
missing	Default is set to `NA`, which means that `NA` values should be considered as 'missing' by the algorithm. Sometimes, `0` or other extreme values might be used to represent missing values. This parameter is only used when input is a dense matrix.

maximize	If nstop is set, then this parameter must be set as well. Default is FALSE since the weighted loss function is used for minimization. This parameter is passed to the cb.early.stop callback.
cluster	A character string of a cluster identifier. It could be NULL if the sampling design were plugged in the design argument.
strata	A character string of a strata identifier. It could be NULL if the sampling design were plugged in the design argument.
weights	A character string of a sampling weights' identifier. It could be NULL if the sampling design were plugged in the design argument.
design	An object of class survey.design generated by survey::svydesign(). It could be NULL if information about cluster, strata, weights and data were given.
data	A data frame with information about the response variable and covariates, as well as sampling weights, strata, and cluster indicators. It could be NULL if the sampling design were added to the design argument.
print_every_n	Print each n-th iteration evaluation messages when verbose>0. Default is 50L printing every 50th iteration message. This parameter is passed to the cb.print.evaluation callback.
...	optional parameters to be passed to the low level function xgboost::xgb.train() or xgboost::xgboost().

Value

A list of return values for optimal (hyper-)parameters as follows:

- nrounds : The best iteration number having the best evaluation metric value, i.e., minimum mean test error
- params : A list of optimal (hyper-) parameters selected among all the nitr models implemented by wxgboost(final.model=FALSE).
- min_test_errors: A table of all minimum weighted test errors and corresponding best iteration numbers for nitr iterations. One can check how nrounds has been chosen to be optimal by the minimum weighted test error.

See Also

xgboost::xgb.train() for relevant arguments and return values in detail.

wXGBoost(), replicate_weights() for detailed information on implementation.

Examples

```
## Set user defined (hyper-)parameters for random search:
custom  <- list(nthread=3,
              max_depth = 5,
              eta = function() runif(1, 0.05, 0.15),
              subsample = 0.5, # much smaller for large N
              max_delta_step = sample(1:10, 1) ) # for very imbalanced case
```

```
 ## search for optimal parameters

data(nhanes2013_sbc)
Mydesign <- survey::svydesign(ids=~SDMVPSU, strata = ~SDMVSTRA,
                    weights = ~WTSAF2YR,nest = TRUE, data = nhanes2013_sbc)
opt.par<- optim_wxgb_para(nhanes2013_sbc$HBP,2:61,custom,method="dCV",
                          design=Mydesign,print_every_n=250L)

## search using bootstrap instead, with a custom B:
opt.par.boot <- optim_wxgb_para(nhanes2013_sbc$HBP,2:61,custom,method="bootstrap",
                          B = 50, design=Mydesign,print_every_n=250L)


## fitting the optimal model

### The data needs to be a sparse matrix without intercept.
### Generate a sparse matrix from a regular dataset for training
unwtData<-nhanes2013_sbc[,-c(62:66)]   #excluding weight related vectors
Mat<- Matrix::sparse.model.matrix(HBP~. ,data = unwtData )[,-1]   #dropping intercept
wtData <- xgboost::xgb.DMatrix(data = nhanes2013_sbc,label=nhanes2013_sbc$HBP,
                      weight=nhanes2013_sbc$WTSAF2YR)

# Can include test data--it should be a xgb.DMatrix object--
# e.g., `eval=list(train=data1,test=data2)`.
wxgb<-xgboost::xgb.train(data=wtData, params=opt.par$params, nrounds=opt.par$nrounds,
          evals=list(train=wtData),
          custom_metric = evalerror_bin, verbose=0)
```

`par_wElnet` *Parallel cross-validated Weighted Elastic Net across a sequence of alpha values*

Description

Fits `wElnet()` once per value of `alpha`, assigning each alpha to its own CPU core via the `parallel` package, and then selects the overall best (alpha, lambda) combination by minimizing the average weighted error across all fits.

```
This function always uses a \code{"PSOCK"} cluster, on every
platform (Windows, Mac, Linux), because it is designed to run
safely inside RStudio. RStudio manages a graphics device and
other GUI state in the main R process; forking that process
(\code{"FORK"} clusters) is explicitly discouraged by R's own
\code{parallel} documentation in GUI-based sessions and can
crash the session. \code{"PSOCK"} workers are fresh, independent
R processes, so this risk does not apply, at the cost of a
small amount of serialization overhead when the cluster starts.
```

Because this function is part of the \code{MLSurvey} package, each worker simply attaches \code{MLSurvey} (along with \code{glmnet} and \code{survey}) via \code{library()}, which makes [wElnet()], [replicate_weights()], and all of their internal helper functions available on every worker automatically -- no manual \code{clusterExport()} of function definitions is required.

Usage

```
par_wElnet(
  alpha = c(0, 0.001, 0.008, 0.027, 0.064, 0.125, 0.216, 0.343, 0.512, 0.729, 1),
  n.cores = NULL,
  blas.threads = 1,
  data = NULL,
  col.y = NULL,
  col.x = NULL,
  cluster = NULL,
  strata = NULL,
  weights = NULL,
  design = NULL,
  family = c("gaussian", "binomial"),
  lambda = NULL,
  nlambda = 100,
  lambda.min.ratio = NULL,
 method = c("dCV", "JKn", "bootstrap", "subbootstrap", "BRR", "split", "extrapolation"),
  k = 10,
  R = 1,
  B = 200,
  dCV.sw.test = FALSE,
  train.prob = 0.7,
  method.split = c("dCV", "bootstrap", "subbootstrap"),
  print.rw = FALSE,
  standardize = TRUE,
  offset = NULL,
  ...
)
```

Arguments

alpha	A numeric vector of elastic net mixing parameters to evaluate in parallel, one per core. Default is the standard 11-point grid `c(0, 0.001, 0.008, 0.027, 0.064, 0.125, 0.216,0.343, 0.512, 0.729, 1)`.
n.cores	Integer number of CPU cores to use. Defaults to `min(parallel::detectCores() - 1, length(alpha))`.
blas.threads	Integer number of threads each worker's BLAS library is allowed to use. Defaults to 1, to avoid oversubscribing CPU cores when `n.cores` worker processes each also try to run multithreaded linear algebra inside `glmnet`. Requires the

RhpcBLASctl package; if it is not installed, this is silently skipped with a message.

data — A data frame with information about the response variable and covariates, as well as sampling weights, strata, and cluster indicators. Passed to wElnet() for every alpha. It could be NULL if the sampling design were supplied in the design argument instead.

col.y — A numeric value indicating the index of the response variable or a string of the response variable name.

col.x — A numeric vector indicating indices of covariates or a vector of strings indicating these column names.

cluster — A string of a cluster identifier. It could be NULL if the sampling design were plugged in the design argument.

strata — A string of a strata identifier. It could be NULL if the sampling design were plugged in the design argument.

weights — A string of a sampling weights' identifier. It could be NULL if the sampling design were included in the design argument.

design — An object of class survey.design generated by svydesign. It could be NULL if information about cluster, strata, weights, and data were given instead.

family — A string indicating a glm family object for fitting weighted elastic net models. Choose between gaussian (linear models) or binomial (logistic models).

lambda — A sequence of lambda supplied by a user. If NULL (default), wElnet() generates its own grid for each alpha.

nlambda — An integer. The number of lambda grid values to generate when lambda is NULL. Default is 100.

lambda.min.ratio — The smallest value for lambda, as a fraction of the maximum lambda. If NULL (default), wElnet()'s own internal default is used.

method — A string indicating a method of replicate weights. Choose one of: JKn, dCV, bootstrap, subbootstrap, BRR, split, extrapolation.

k — An integer. The number of folds for the dCV method. Default is k=10.

R — An integer. The number of times the sample is partitioned for dCV, split, or extrapolation methods. Default is R=1.

B — An integer. The number of bootstrap re-samples for bootstrap and subbootstrap methods. Default is B=200.

dCV.sw.test — A logical value indicating the method for estimating the error for the dCV method. Default is FALSE.

train.prob — A numeric between 0 and 1 setting the proportion of clusters (for split) or strata (for extrapolation) in training sets. Default is train.prob = 0.7.

method.split — A string indicating which replicate weights method to use under the split method: dCV, bootstrap, or subbootstrap.

print.rw — A logical value. If TRUE, each fitted w.elnet object's replicate-weights data set is retained in its output. Default FALSE.

standardize	A logical value passed to glmnet indicating whether predictors should be standardized before fitting. Default TRUE.
offset	An optional offset vector passed to glmnet. Default NULL.
...	Any further arguments accepted by glmnet (beyond standardize and offset), forwarded to every wElnet() call.

Value

An object of class par.w.elnet, a list with:

- results: a named list of w.elnet objects (the output of wElnet()), one per alpha.
- summary: a data frame with columns alpha, lambda.min, and min.error, one row per alpha, sorted in the order alpha was supplied.
- best: the single w.elnet object with the lowest min.error across all alphas.
- best.alpha: the alpha value corresponding to best.

See Also

wElnet() for the single-alpha model fit that this function parallelizes.

Examples

```
library(MLSurvey)
data(nhanes2013_sbc)
alpha <- c(0.000, 0.001, 0.008, 0.027, 0.064, 0.125, 0.216, 0.343, 0.512, 0.729, 1.000)

en.par <- par_wElnet(alpha = alpha,
                     data = nhanes2013_sbc,
                     col.y = "HBP", col.x = 2:61,
                     family = "binomial",
                     cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
                     method = "dCV", k = 10, R = 20)

en.par$summary      # error per alpha
en.par$best.alpha    # winning alpha
en.par$best          # winning w.elnet object (same structure as wElnet() output)
```

---

plot.par.w.elnet *Plot method for* par.w.elnet *objects*

Description

A plot method to visualize the results of par_wElnet(): the weighted error curve across the lambda grid for each alpha, and/or a summary of the best (minimum) error achieved by each alpha, with the overall best (alpha, lambda) combination highlighted.

```
For the single-alpha output of \code{\link{wElnet}} itself, see
[plot.w.elnet()].
```

Usage

```
## S3 method for class 'par.w.elnet'
plot(
  x,
  type = c("both", "profile", "summary"),
  log.lambda = TRUE,
  legend.pos = "topright",
  palette = NULL,
  main.profile = NULL,
  xlab.profile = NULL,
  ylab.profile = NULL,
  main.summary = NULL,
  xlab.summary = NULL,
  ylab.summary = NULL,
  ...
)
```

Arguments

x	An object of class par.w.elnet, the output of par_wElnet().
type	A string indicating what to plot. One of: • "profile": error vs. log(lambda), one curve per alpha (base plot, similar in spirit to glmnet's own CV plot, but overlaying every alpha on a single panel so they can be compared directly). • "summary": minimum error achieved vs. alpha, one point per alpha, with the overall best alpha highlighted. • "both" (default): both plots side by side.
log.lambda	A logical value. If TRUE (default), the x-axis of the "profile" plot is log(lambda), matching glmnet convention. If FALSE, lambda is plotted on its original scale.
legend.pos	A string passed to legend() for the "profile" panel. Default is "topright". Set to NULL to suppress the legend (useful when there are many alphas and the legend gets crowded).
palette	A vector of colors to cycle through for the different alphas in the "profile" plot. Defaults to grDevices::hcl.colors(n, "Zissou 1"), where n is the number of alphas.
main.profile, xlab.profile, ylab.profile	Optional strings overriding the title and axis labels of the "profile" panel. If NULL (default), sensible defaults are used ("Error profile by alpha", "log(lambda)" or "lambda", and "Weighted error").
main.summary, xlab.summary, ylab.summary	Optional strings overriding the title and axis labels of the "summary" panel. If NULL (default), sensible defaults are used ("Best error by alpha", "alpha", and "Minimum weighted error").
...	Additional graphical parameters (e.g. col, pch, cex, lwd) passed to the underlying plot() calls. Do not pass main, xlab, or ylab here – use the dedicated arguments above instead.

Value

Invisibly returns `x`. Called for its side effect of producing a plot.

Examples

```
library(MLSurvey)
data(nhanes2013_sbc)
alpha <- c(0.000, 0.001, 0.008, 0.027, 0.064, 0.125, 0.216, 0.343, 0.512, 0.729, 1.000)

en.par <- par_wElnet(alpha = alpha,
                     data = nhanes2013_sbc,
                     col.y = "HBP", col.x = 2:61,
                     family = "binomial",
                     cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
                     method = "dCV", k = 10, R = 20)

plot(en.par)                    # both panels
plot(en.par, type = "profile")  # just the error-vs-lambda curves
plot(en.par, type = "summary")  # just the best-error-vs-alpha curve
plot(en.par, type = "profile", main.profile = "My custom title", xlab.profile = "Penalty")
```

`plot.w.elnet` *Plot method for* `w.elnet` *objects (single-alpha* `wElnet()` *output)*

Description

A method to visualize the weighted error curve across the lambda grid for a single `wElnet` fit (i.e. a single, fixed alpha), marking the selected `lambda.min`. This is the single-alpha counterpart to `plot.par.w.elnet()`, which handles the multi-alpha output of `par_wElnet()`.

Usage

```
## S3 method for class 'w.elnet'
plot(x, log.lambda = TRUE, main = NULL, xlab = NULL, ylab = NULL, ...)
```

Arguments

`x`	An object of class `w.elnet`, the output of `wElnet`.
`log.lambda`	A logical value. If `TRUE` (default), the x-axis is `log(lambda)`, matching `glmnet` convention. If `FALSE`, lambda is plotted on its original scale.
`main`, `xlab`, `ylab`	Optional strings overriding the plot title and axis labels. If `NULL` (default), sensible defaults are used ("Error profile (alpha = ...)", "log(lambda)" or "lambda", and "Weighted error").
`...`	Additional graphical parameters (e.g. `col`, `lwd`, `lty`) passed to the underlying `plot()` call. Do not pass `main`, `xlab`, or `ylab` here – use the dedicated arguments above instead.

Value

Invisibly returns `x`. Called for its side effect of producing a plot.

Examples

```
library(MLSurvey)
data(nhanes2013_sbc)

en.dcv <- wElnet(data = nhanes2013_sbc,
                 col.y = "HBP", col.x = 2:61,
                 family = "binomial", alpha = 0.729,
                 cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
                 method = "dCV", k = 10, R = 20)

plot(en.dcv)
plot(en.dcv, main = "My custom title", xlab = "Penalty", ylab = "Error")
```

`print.w.elnet` *Print method for* `w.elnet` *objects (single-alpha* `wElnet()` *output)*

Description

Prints a short summary of a `wElnet` fit: the alpha used, the selected `lambda.min` and its weighted error, the number of nonzero coefficients (degrees of freedom) at that lambda, and the size of the lambda grid that was searched.

Usage

```
## S3 method for class 'w.elnet'
print(x, digits = 4, ...)
```

Arguments

`x`	An object of class `w.elnet`, the output of `wElnet`.
`digits`	Integer number of significant digits to use when printing numeric values. Default is 4.
`...`	Currently unused; included for S3 consistency with `print()`.

Value

Invisibly returns `x`. Called for its side effect of printing to the console.

Examples

```
library(MLSurvey)
data(nhanes2013_sbc)

en.dcv <- wElnet(data = nhanes2013_sbc,
                 col.y = "HBP", col.x = 2:61,
                 family = "binomial", alpha = 0.729,
                 cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
                 method = "dCV", k = 10, R = 20)

en.dcv          # or print(en.dcv)
```

`replicate_weights` *Replicate weights methods*

Description

A function calculating replicate weights.

Usage

```
replicate_weights(
  data,
 method = c("JKn", "dCV", "bootstrap", "subbootstrap", "BRR", "split", "extrapolation"),
  cluster = NULL,
  strata = NULL,
  weights = NULL,
  design = NULL,
  k = 10,
  R = 1,
  B = 200,
  train.prob = 0.7,
  method.split = c("dCV", "bootstrap", "subbootstrap"),
  rw.test = FALSE,
  dCV.sw.test = FALSE
)
```

Arguments

`data`	A data frame with information on (at least) cluster and strata indicators, and sampling weights. It could be `NULL` if the sampling design were indicated in the `design` argument (see `design`).
`method`	A string of a name of replicate weights methods. Choose one of these: `JKn`, `dCV`, `bootstrap`, `subbootstrap`, `BRR`, `split`, `extrapolation`.

cluster	A string of a cluster identifier. It could be NULL if the sampling design were plugged in the design argument.
strata	A string of a strata identifier. It could be NULL if the sampling design were plugged in the design argument.
weights	A string of a sampling weights' identifier. It could be NULL if the sampling design were included in the design argument.
design	An object of class survey.design generated by svydesign. It could be NULL if information about cluster, strata, weights and data were given.
k	An integer. The number of folds to be defined. Default is k=10. Only applies to the dCV method.
R	An integer. The number of times the sample is partitioned. Default is R=1. Only applies to dCV, split or extrapolation methods.
B	An integer. The number of bootstrap re-samples. Default is B=200. Only applies to bootstrap and subbootstrap methods.
train.prob	A numeric between 0 and 1 that sets the proportion of clusters (for the method split) or strata (for the method extrapolation) in training sets. Default is train.prob=0.7. Only applies to split and extrapolation methods.
method.split	A string of one the following replicate weights methods to be implemented under the split method in the method argument: dCV, bootstrap or subbootstrap.
rw.test	A logical value. If TRUE, the function returns in the output object the replicate weights to the corresponding test sets. If FALSE, only the replicate weights of the training sets are returned. Default is rw.test = FALSE.
dCV.sw.test	A logical value. If TRUE, original sampling weights for the units in the test sets are returned instead of the replicate weights. Default is dCV.sw.test = FALSE. Only applies to dCV method.

Details

This function from R-wlasso (https://github.com/aiparragirre/wlasso/, recently registered in CRAN: svyVarSel) is slightly modified for our package development. The following description is also obtained from the original source for convenience.

Some of replicate weights methods, such as JKn, bootstrap, subbootstrap and BRR, are implemented in R-survey, as.svrepdesign, from which the replicate_weights() function generates replicate weights for these options; dCV, split and extrapolation have been expressly defined in this function.

Selecting any of the above-mentioned methods returns a new data frame, in which new columns are added to the original data set; each column by replicate weights always creates different pairs of training sets and optionally the corresponding test sets by setting rw.test=TRUE. The number of new columns and their corresponding names are usually determined by numeric arguments, i.e.,k, or R, etc. and specifically by the selected replicate weights method. The new columns indicating training and test sets follow the similar structure for any selected method: rw_r_x_train_t, where x=1,...,R, and t=1,...,T indicate the x-th partition of the sample,the t-th training set, respectively. Similarly, the new columns for test sets are structured by rw_r_x_test_t or sw_r_x_test_t. For other methods, fold or set for each unit respectively labeled as fold_t or set_t is included in each partition depending on the method in a data set.

For JKn, bootstrap, and subbootstrap, the sw_r_1_test_* test columns (generated when rw.test = TRUE) represent an out-of-bag (OOB) style test set: for each replicate, a unit's original sampling weight is assigned to the test column if and only if that unit's PSU was not drawn into that particular replicate (resample multiplier of exactly 0), and 0 otherwise. For bootstrap/subbootstrap specifically, this means the OOB test fraction per replicate is random and not fixed in advance – in expectation roughly $1/e \approx 37\%$ of PSUs are held out per replicate for ordinary bootstrap, somewhat more for subbootstrap – rather than a predetermined split proportion as in dCV or split.

Value

The function returns a new data frame with new columns, each of which indicates replicate weights for different subsets.

Examples

```
data(nhanes2013_sbc)

## JKn ------------------------------------------------------------------
data_rw <- replicate_weights(data = nhanes2013_sbc,
                             method = "JKn",
                             cluster = "SDMVPSU",
                             strata = "SDMVSTRA",
                             weights = "WTSAF2YR",
                             rw.test = TRUE)

## dCV ------------------------------------------------------------------
data_rw <- replicate_weights(data = nhanes2013_sbc,
                             method = "dCV",
                             cluster = "SDMVPSU",
                             strata = "SDMVSTRA",
                             weights = "WTSAF2YR",
                             k = 10, R = 20,
                             rw.test = TRUE)

## subbootstrap ---------------------------------------------------------
# rw.test = TRUE now also returns out-of-bag ("sw_r_1_test_*") weights:
# units whose PSU was not drawn into a given replicate (resample multiplier
# == 0) are treated as that replicate's held-out test set.
data_rw <- replicate_weights(data = nhanes2013_sbc,
                             method = "subbootstrap",
                             cluster = "SDMVPSU",
                             strata = "SDMVSTRA",
                             weights = "WTSAF2YR",
                             B = 100,
                             rw.test = TRUE)

## BRR ------------------------------------------------------------------
data_rw <- replicate_weights(data = nhanes2013_sbc,
                             method = "BRR",
                             cluster = "SDMVPSU",
                             strata = "SDMVSTRA",
                             weights = "WTSAF2YR",
```

```
                              rw.test = TRUE)

## split ------------------------------------------------------------------

data_rw <- replicate_weights(data = nhanes2013_sbc,
                              method = "split",
                              cluster = "SDMVPSU",
                              strata = "SDMVSTRA",
                              weights = "WTSAF2YR",
                              R=20,
                              train.prob = 0.5,
                              method.split = "subbootstrap",
                              rw.test = TRUE)

## extrapolation ----------------------------------------------------------
data_rw <- replicate_weights(data = nhanes2013_sbc,
                             method = "extrapolation",
                             cluster = "SDMVPSU",
                             strata = "SDMVSTRA",
                             weights = "WTSAF2YR",
                             R=20,
                             train.prob = 0.5,
                             rw.test = TRUE)
```

`wElnet` *Weighted Elastic Net prediction models for complex survey data*

Description

A model selection function fitting Elastic Net (linear or logistic) models with sampling weights by a replicating weights method to select an optimal lambda for a given alpha by weighted error minimization. This is extended from `wlasso::wlasso()`, which is now officially available in `wlasso`. For an extended model selection for a sequence of alpha's by CPU parallel computing, use `par_wElnet()`.

Usage

```
wElnet(
  data = NULL,
  col.y = NULL,
  col.x = NULL,
  cluster = NULL,
  strata = NULL,
  weights = NULL,
  design = NULL,
  family = c("gaussian", "binomial"),
  lambda = NULL,
```

```
  alpha = 1,
  nlambda = 100,
  lambda.min.ratio = NULL,
 method = c("dCV", "JKn", "bootstrap", "subbootstrap", "BRR", "split", "extrapolation"),
  k = 10,
  R = 1,
  B = 200,
  dCV.sw.test = FALSE,
  train.prob = 0.7,
  method.split = c("dCV", "bootstrap", "subbootstrap"),
  print.rw = FALSE,
  standardize = TRUE,
  offset = NULL,
  ...
)
```

Arguments

`data`	A data frame with information about the response variable and covariates, as well as sampling weights, strata, and cluster indicators. It could be `NULL` if the sampling design were replaced in the `design` argument.
`col.y`	A numeric value indicating the index of the response variable or a string of the response variable name .
`col.x`	A numeric vector indicating indices of covariates or a vector of strings indicating these column names.
`cluster`	A string of a cluster identifier. It could be `NULL` if the sampling design were plugged in the `design` argument.
`strata`	A string of a strata identifier. It could be `NULL` if the sampling design were plugged in the `design` argument.
`weights`	A string of a sampling weights' identifier. It could be `NULL` if the sampling design were included in the `design` argument.
`design`	An object of class `survey.design` generated by `svydesign`. It could be `NULL` if information about `cluster`, `strata`, `weights` and `data` were given.
`family`	A string indicating a `glm` family object for fitting weighted elastic net models. Choose between `gaussian` (to fit linear models) or `binomial` (for logistic models).
`lambda`	A sequence of lambda supplied by a user.
`alpha`	A numeric value of the elastic net mixing parameter $$0 \leq \alpha \leq 1,$$ where `alpha = 1` for the LASSO penalty, and `alpha = 0` for the Ridge penalty.
`nlambda`	An integer. The number of lambda grids to be generated. Default is `nlambda = 100`.

`lambda.min.ratio`
: The smallest value for `lambda`, as a fraction of the maximum lambda. If `NULL` (default), this is computed internally: `0.01` if the number of observations is smaller than the number of covariates, otherwise `0.0001`. A very small value of `lambda.min.ratio` will lead to a saturated fit in the `nobs < nvars` case. This is undefined for a `"binomial"` model.

`method`
: A string indicating a method of replicate weights. Choose one of these: `JKn`, `dCV`, `bootstrap`, `subbootstrap`, `BRR`, `split`, `extrapolation`.

`k`
: An integer. The number of folds for the `dCV` method. Default is `k=10`.

`R`
: An integer. The number of times the sample is partitioned for `dCV`, `split` or `extrapolation` method. Default is `R=1`.

`B`
: An integer. The number of bootstrap re-samples for `bootstrap` and `subbootstrap` methods. Default is `B=200`.

`dCV.sw.test`
: A logical value indicating the method for estimating the error for `dCV` method. `FALSE`, (the default option) estimates the error for each test set and defines the cross-validated error based on the average strategy. Option `TRUE` estimates the cross-validated error based on the pooling strategy.

`train.prob`
: A numeric between 0 and 1 that sets the proportion of clusters (for the method `split`) or strata (for the method `extrapolation`) in training sets. Default is `train.prob = 0.7`.

`method.split`
: A string of one the following replicate weights methods to be implemented under the `split` method in the `method` argument: `dCV`, `bootstrap` or `subbootstrap`.

`print.rw`
: A logical value. If `TRUE`, the data set with the replicate weights is saved in the output object. Default `print.rw=FALSE`.

`standardize, offset, ...`
: Optional parameters to be passed to the low level function `glmnet::glmnet()`.

Value

The output object of the function `wElnet()` is an object of class `w.elnet`:

- `lambda`: A list containing information of two elements:
  - `grid`: A vector of all grid values for the tuning parameter.
  - `min`: A numeric value of an optimal tuning parameter minimizing the average error.
- `error`: A list containing information of two elements:
  - `average`: A numeric vector indicating the average error for each tuning parameter in `lambda$grid`.
  - `all`: A numeric matrix indicating the error of each test set for each tuning parameter.
- `alpha`: A numeric value of the elastic net mixing parameter used to fit this model (as supplied to the `alpha` argument).
- `model`: A list containing information of two elements in relation to the fitted models.
  - `grid`: A list containing information on models fitted by each tuning parameter in the `lambda$grid`.
    - `a0`: A numeric vector of model intercepts across the whole grid of tuning parameters (hence, of the same length as `lambda$grid`).

    - beta: A matrix of regression coefficients for all the covariates across the whole grid of tuning parameters (the dimension of the matrix is the size of covariates by the length of lambda$grid).
    - df: A numeric vector of the degrees of freedom (i.e., the number of coefficients different from zero) across the whole grid of tuning parameters (hence, of the same length as lambda$grid).
  - final_model: An object with S3 class glmnet found in r-glmnet. A list containing information on the final model fitted by the optimal tuning parameter, lambda$min. i.e., the optimal tuning parameter selected from lambda$grid.

- data.rw: A data frame containing the original data set and the replicate weights added to define training and test sets. Only included in the output object if print.rw=TRUE.
- call: The call that executes this function producing the object of w.elnet.

- Note that both model and final_model are fitted by the original data, not by one of the generated training sets by replicate weights.

See Also

svyVarSel::wlasso() for detailed arguments.

glmnet::glmnet() for detailed arguments and return values.

Examples

```
 data(nhanes2013_sbc)
# alpha can be selected based on `glmnet::glmnet()`:
alpha <- c(0.000, 0.001, 0.008, 0.027, 0.064, 0.125, 0.216, 0.343, 0.512, 0.729, 1.000)

en.dcv <- wElnet(data = nhanes2013_sbc,
              col.y = "HBP", col.x = 2:61,
              family = "binomial", alpha = 0.729,
              cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
              method = "dCV", k=10, R=20)

# Or equivalently:
des <- survey::svydesign(ids=~SDMVPSU, strata = ~SDMVSTRA, weights = ~WTSAF2YR,
                            nest = TRUE, data = nhanes2013_sbc)
en.dcv <- wElnet(col.y = "HBP", col.x = 2:61, design = des,
              family = "binomial", alpha = alpha[10],
              method = "dCV", k=10, R=20)
```

---

wRandomforest *Weighted Random Forest prediction models (wRF) for complex survey data*

Description

A model selection function for fitting wRF (linear or logistic) models for complex survey data with sampling weights by a replicating weights method to select optimal tuning parameters by weighted error minimization. Detailed arguments are referred to in R-randomForest::randomForest.

Usage

```
wRandomforest(
  data = NULL,
  col.x = NULL,
  y = NULL,
  xtest = NULL,
  ytest = NULL,
  cluster = NULL,
  strata = NULL,
  weights = NULL,
  design = NULL,
  method = c("dCV", "JKn", "BRR", "split", "extrapolation"),
  k = 10,
  R = 1,
  B = 200,
  dCV.sw.test = FALSE,
  train.prob = 0.7,
  method.split = c("dCV", "bootstrap", "subbootstrap"),
  print.rw = FALSE,
  ntree = 500,
 mtry = if (!is.null(y) && !is.factor(y)) max(floor(ncol(data[, col.x])/3), 1) else
    floor(sqrt(ncol(data[, col.x]))),
  replace = TRUE,
  classwt = NULL,
  cutoff,
  sampsize = if (replace) nrow(data) else ceiling(0.632 * nrow(data)),
  nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
  maxnodes = NULL,
  importance = FALSE,
  proximity = FALSE,
  oob.prox = proximity,
  norm.votes = TRUE,
  do.trace = FALSE,
  keep.forest = TRUE,
  corr.bias = FALSE,
  keep.inbag = FALSE,
  ...
)
```

Arguments

data	A data frame with information about independent variables, as well as sampling weights and strata and cluster indicators. It could be NULL if the sampling design were plugged in the design argument.
col.x	A numeric vector indicating indices of columns for covariates or independent variables or a vector of character strings indicating names of these columns.
y	A vector of the response variable. If a factor, classification is assumed, otherwise, regression is assumed. If omitted, randomForest will run in a unsupervised mode.
xtest	A data frame or matrix containing predictors for the test set.
ytest	A vector of response variable for the test set.
cluster	A character string indicating the name of cluster identifiers. It could be NULL if the sampling design were plugged in the design argument.
strata	A character string indicating the name of strata identifiers. It could be NULL if the sampling design were plugged in the design argument.
weights	A character string indicating the name of sampling weights. It could be NULL if the sampling design were plugged in the design argument.
design	An object of class survey.design generated by survey::svydesign(). It could be NULL if information about cluster, strata, weights and data were given.
method	A character string indicating a method of replicate weights. Choose one of these: JKn, dCV, BRR, split, extrapolation. Note: bootstrap and subbootstrap are not offered here (see Details).
k	An integer. The number of folds for the dCV method. Default is k=10.
R	An integer. The number of times the sample is partitioned for dCV, split or extrapolation method. Default is R=1.
B	Not used by wRandomforest(); retained in the signature only for interface consistency with wElnet()/wXGBoost(). See Details.
dCV.sw.test	A logical value indicating the method for estimating the error for dCV method. FALSE, (the default option) estimates the error for each test set and defines the cross-validated error based on the average strategy. Option TRUE estimates the cross-validated error based on the pooling strategy
train.prob	A numeric between 0 and 1, indicating the proportion of clusters (for the method split) or strata (for the method extrapolation) to be set in the training sets. Default is train.prob = 0.7. Only applies for split and extrapolation methods.
method.split	A string of one the following replicate weights methods to be implemented under the split method in the method argument: dCV, bootstrap or subbootstrap. (This still permits bootstrap/subbootstrap as the inner resampling scheme within a split's training partition, since that usage does not conflate with randomForest's own internal bagging the way a standalone top-level bootstrap/subbootstrap replicate-weight method would – see Details.)
print.rw	A logical value. If TRUE, the data set with the replicate weights is saved in the output object. Default print.rw=FALSE.

`ntree` The number of trees to grow. This should not set to be too small a number, to ensure that every input row gets predicted at least a few times.

`mtry` The number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)

`replace, classwt, cutoff, sampsize, nodesize, maxnodes, importance, proximity, oob.prox, norm.votes, do.trace, keep.forest, corr.bias, keep.inbag, ...`
Optional parameters to be passed to the low level function `randomForest::randomForest()`.

Details

Unlike `wElnet()` and `wXGBoost()`, `wRandomforest()` does not offer `method = "bootstrap"` or `method = "subbootstrap"` as a top-level replicate-weight resampling method. Random forests already perform bootstrap resampling internally for every tree (bagging is the defining mechanism of the algorithm), so layering an additional, outer bootstrap/subbootstrap replicate-weight scheme on top would double-resample in a way that has no clear interpretation for variance/error estimation, and would not serve the same tuning-parameter-selection purpose that `dCV`, `JKn`, `BRR`, `split`, or `extrapolation` do here. If `method = "bootstrap"` or `"subbootstrap"` is supplied, `wRandomforest()` stops with an informative error rather than silently proceeding.

Value

The output object of the function `wRandomforest()` is an object of class `w.randomforest`:

- `mtry`: A list containing information on `mtry` as a tuning parameter (the number of predictors sampled for splitting at each node):
  - `all`: A numeric vector of `k*R` `mtry`- values, i.e. `mtry` for each fold per replicate.
  - `optimal.mtry`: An optimal tuning parameter, `mtry` among `mtry$all` that minimizes the average error.
- `evaluation_log`: A list containing information of two elements:
  - `weighted.test.error`: A vector of the average errors corresponding to tuning parameters in `mtry$all`.
  - `min.weighted.test.error`: An minimum error for the optimal tuning parameter from `mtry$optimal.mtry` to select the final model.
- `model`: A list containing information on the fitted model by `optimal.mtry`: an object of class `randomForest`. Note that the selected model is fitted by the original `data`, not one of the generated training sets by replicate weights.
- `data.rw`: A data frame containing the original data set and the replicate weights added to define training and test sets. Only included in the output object if `print.rw=TRUE`.
- `call`: The call that executes this function producing the object of `w.randomforest`.

Note

`final.model` can leverage R-`randomForest` functionality for comprehensive data analysis for linear/logistic regression but will be extensively used for other models, i.e., Cox, Poisson, etc.

See Also

randomForest::randomForest for arguments and return values in detail.

Examples

```
 # Global options to avoid only one PSU in a stratum in a particular domain or subpopulation
options(survey.adjust.domain.lonely=TRUE)
options(survey.lonely.psu="adjust")


 # For classification: y should be a factor.
 # If a test set is provided, one can plug it into `xtest`, `ytest`
 # just as implemented in \code{randomForest::randomForest()}.

 data(nhanes2013_sbc)

dcv.rf <- wRandomforest(data = nhanes2013_sbc,
              y = as.factor(nhanes2013_sbc$HBP), col.x = 2:61,
              cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
              method = "dCV", k=5, R=5,importance=TRUE,proximity = TRUE)

# Or equivalently:
des <- survey::svydesign(ids=~SDMVPSU, strata = ~SDMVSTRA, weights = ~WTSAF2YR,
                              nest = TRUE, data =nhanes2013_sbc)
dcv.rf <- wRandomforest(y = as.factor(nhanes2013_sbc$HBP), col.x = 2:61, design =des,
                        importance=TRUE,proximity = TRUE,
                        method = "dCV", k=5, R=5)
```

`wXGBoost` *Weighted XGBoost prediction models (wXGBoost) for complex survey data*

Description

A model selection function for fitting XGBoost (linear or logistic) models for complex survey data with sampling weights by a replicating weights method to select optimal tuning parameters by weighted error minimization. Detailed arguments are referred to in `xgboost::xgboost()`.

Usage

```
wXGBoost(
  data = NULL,
  y = NULL,
  col.x = NULL,
  missing = NA,
  final.model = TRUE,
  test.data = NULL,
```

```
  cluster = NULL,
  strata = NULL,
  weights = NULL,
  design = NULL,
  params = list(),
  nrounds,
  verbose = 0,
  print_every_n = 1L,
  maximize = FALSE,
  early_stopping_rounds = 8,
 method = c("dCV", "JKn", "bootstrap", "subbootstrap", "BRR", "split", "extrapolation"),
  k = 10,
  R = 1,
  B = 200,
  dCV.sw.test = FALSE,
  train.prob = 0.7,
  method.split = c("dCV", "bootstrap", "subbootstrap"),
  print.rw = FALSE,
  save_period = NULL,
  save_name = "wxgboost.model",
  wxgb_model = NULL,
  callbacks = list()
)
```

Arguments

data	A data frame with information about the response variable and covariates, as well as sampling weights, strata, and cluster indicators. It could be NULL if the sampling design were plugged into the design argument.
y	A vector of response variable.
col.x	A numeric vector indicating indices of the covariates or a string vector indicating these column names.
missing	Default is set to NA, which means that NA values should be considered as 'missing' by the algorithm. Sometimes, 0 or other extreme values might be used to represent missing values. This parameter is only used when input is a dense matrix.
final.model	Boolean. Default is TRUE printing out the object of class xgb.Booster for the final model selected by replicate weights' CV with corresponding predicted values.
test.data	Optional. An object of xgb.DMatrix. It can be used to fit the final model with the best iteration for evaluation and to estimate predicted values only if final.model = TRUE.
cluster	A string of a cluster identifier. It could be NULL if the sampling design were plugged in the design argument.
strata	A string of a strata identifiers. It could be NULL if the sampling design were plugged in the design argument.

weights — A string of a sampling weights' identifier. It could be NULL if the sampling design were plugged in the design argument.

design — An object of class survey.design generated by survey::svydesign(). It could be NULL if information about cluster, strata, weights and data were given.

params — The list of parameters, or the list obtained by xgboost::xgb.params(). Should include objective to be a string of either binary:logistic for binary classification or reg:squarederror for linear regression. Default evaluation metrics assigned by params:objective are weighted mse (evalerror_reg()) for regression, weighted mean error (evalerror_bin()) for logistic reg (binary classification). The complete list of parameters is found on xgboost::xgb.train() or https://xgboost.readthedocs.io/en/latest/parameter.html.

nrounds — The maximum number of boosting iterations.

verbose — 0 for xgboost to stay silent. Default= 1 for printing information about performance. 2 for printing some additional information out. Note that setting verbose > 0 automatically engages the cb.print.evaluation(period=1) callback function.

print_every_n — Print each n-th iteration evaluation messages when verbose>0. Default is 1 printing all messages. This parameter is passed to the cb.print.evaluation callback.

maximize — If early_stopping_rounds is set, then this parameter must be set as well. Default is FALSE since the weighted loss function is used for minimization. This parameter is passed to the xgboost::cb.early.stop callback.

early_stopping_rounds — NULL does not trigger the early stopping function. Setting to an integer k will stop training with a validation set unless the performance improve for k rounds. Setting this parameter engages the cb.early.stop callback.

method — A character string indicating a method of replicate weights. Choose one of these: JKn, dCV, bootstrap, subbootstrap, BRR, split, extrapolation.

k — An integer. The number of folds for the dCV method. Default is k=10.

R — An integer. The number of times the sample is partitioned for dCV, split or extrapolation method. Default is R=1.

B — An integer. The number of bootstrap re-samples for bootstrap and subbootstrap methods. Default is B=200.

dCV.sw.test — A logical value indicating the method for estimating the error for dCV method. FALSE, (the default option) estimates the error for each test set and defines the cross-validated error based on the average strategy. TRUE estimates the cross-validated error based on the pooling strategy.

train.prob — A numeric between 0 and 1, indicating the proportion of clusters (for the method split) or strata (for the method extrapolation) to be set in the training sets. Default is train.prob = 0.7. Only applies to split and extrapolation methods.

method.split — A string of one the following replicate weights methods to be implemented under the split in the method argument: dCV, bootstrap or subbootstrap.

print.rw	A logical value. If TRUE, the data set with the replicate weights is saved in the output object. Default print.rw=FALSE.
save_period	When it is non-NULL, model is saved to disk after every save_period rounds, 0 means save at the end. The saving is handled by the cb.save.model callback.
save_name	The name or path for periodically saved model file.
wxgb_model	A previously built model to continue the training from. Could be either an object of class xgb.Booster, i.e., final.model from the object of class w.xgboost, its raw data, or the name of a file with a previously saved model.
callbacks	A list of callback functions to perform various task during boosting. See xgboost::callbacks. Some of the callbacks are automatically created depending on the parameters' values. Users can provide either existing or their own callback methods in order to customize the training process.

Value

The output object of the function wXGBoost() is an object of class w.xgboost:

- CV.iterations: A list containing information on tuning parameters, nrounds of three elements:
  - niter.range.per.R: A matrix of the range of iterations of replicate-weight CV for R replicates.
  - fold.best: A list of length R; each element fold.best[[r]] is a numeric vector of best iterations (one per fold/replicate-weight column actually produced for replicate r, minimizing the average error). The length of this vector depends on the selected method and is **not** fixed at k: it equals k itself only for method = "dCV"; for "JKn" it equals the number of PSUs; for "bootstrap"/"subbootstrap" it equals B; for "BRR" it is a power-of-4 count determined internally by survey::as.svrepdesign(); and for "split"/"extrapolation" it is 1. (Named fold.best, not k_fold.best, precisely because it is not always a "k-fold" count in this general sense – k itself only has meaning for dCV.)
  - best_iteration: An optimal nround that minimizes mean test error.
- CV.eval_log: A list containing information of two elements:
  - CV: A matrix of iteration-wise mean and sd for training and test sets, respectively.
  - Best_iteration: A vector of minimum CV.eval_log$CV to pick up the final model corresponding to CV.iteratons$best_iteration.
- final.model: A list containing information on the fitted model by Best_iteration: an object of class xgb.Booster. Note that the final model is fitted by the original data, not one of the generated training sets by replicate weights. When test.data is provided, the final model is fitted by evaluating model performance.
- predicted: A vector of predicted values with training set, data (test.data = NULL) or a list of respective predicted values with training set, data and test set, test.data from final.model= TRUE.
- data.rw: A data frame containing the original data set and the replicate weights added to define training and test sets. Only included in the output object if print.rw=TRUE.
- call:The call that executes this function producing the object of w.xgboost.

Note

final.model can leverage R-xgboost functionality for comprehensive data analysis for linear/logistic regression but will be extensively used for other models, i.e., Cox, Poisson, etc.

method = "bootstrap" and "subbootstrap" rely on replicate_weights's out-of-bag (OOB) style test weights (sw_r_1_test_* columns): for each of the B replicates, a unit is treated as "test" for that replicate if and only if its PSU was not drawn into that resample. The OOB test fraction per replicate is therefore random rather than a fixed, predetermined proportion (unlike dCV or split).

Breaking change (if upgrading from an earlier version): the result$CV.iterations field previously named k_fold.best is now named fold.best, and its contents changed from a pre-sized k x R matrix to a list of length R (with per-replicate vectors whose length depends on method, not fixed at k). This was a necessary bug fix: the old matrix, pre-sized assuming exactly k folds regardless of method, either silently corrupted results (for split/extrapolation, where the real fold count is 1, leaving most matrix rows as fake zeros) or crashed with "subscript out of bounds" (for JKn/BRR/bootstrap/ subbootstrap, where the real fold count is almost always greater than the default k=10). Code that previously read fit$CV.iterations$k_fold.best as a matrix must be updated to read fit$CV.iterations$fold.best as a list, e.g. fit$CV.iterations$fold.best[[r]] for replicate r's vector of best iterations.

On the argument k: k denotes the number of folds *only* for method = "dCV" (its only meaningful use, matching replicate_weights's own documentation of k). For every other method, k is ignored entirely (a message is printed if a non-default value is supplied), and the real number of replicate-weight columns used internally is determined by the method itself as described under fold.best above.

See Also

xgboost::xgb.train() for relevant arguments and return values in detail.

eval_loss() for additional customized function for weighted logloss.

Examples

```
 # Global options to avoid only one PSU in a stratum in a particular domain or subpopulation
   options(survey.adjust.domain.lonely=TRUE)
   options(survey.lonely.psu="adjust")

data(nhanes2013_sbc)

# Selecting optimal (hyper-) parameters for a model development:
param <- list(objective = "binary:logistic", max_depth = sample(4:8, 1),
             seed = sample.int(10000,1),
             eta = runif(1, .01, .3),gamma = runif(1, 0.0, 1),
             subsample = 0.5, # much smaller for large N
             colsample_bytree = runif(1, .5, .8),
             min_child_weight = sample(1:40, 1),
             max_delta_step = sample(1:10, 1))  # for very imbalanced case
 # Or equivalently,
param<- xgboost::xgb.params(objective = "binary:logistic", max_depth = sample(4:8, 1),
                      subsample = 0.5)

# Variable selection and best iteration (hyper-parameter) selection:
```

```
# Can add a xgb.DMatrix object of test data (`test.data`) to the function
# to obtain predicted values from the final model after extra evaluation with it.

xgb.cv <- wXGBoost(data = nhanes2013_sbc,
              y = nhanes2013_sbc$HBP, col.x = 2:61,
              cluster = "SDMVPSU", strata = "SDMVSTRA", weights = "WTSAF2YR",
              params = param, nrounds =10000,verbose=0,early_stopping_rounds=5,
              method = "dCV", k=10, R=20)


# Or equivalently:
des <- survey::svydesign(ids=~SDMVPSU, strata = ~SDMVSTRA, weights = ~WTSAF2YR,
                         nest = TRUE, data = nhanes2013_sbc)
xgb.cv <- wXGBoost(y = nhanes2013_sbc$HBP, col.x = 2:61, design = des,
              params = param, nrounds =10000,verbose=0,early_stopping_rounds=5,
              method = "dCV", k=10, R=20)
```

Index

∗ **data**
    nhanes2013_sbc, 11

as.svrepdesign, *23*
autoplot.par.w.elnet, 2
autoplot.par.w.elnet(), *4*
autoplot.w.elnet, 4

ci_wauc, 5

eval_loss, 8
eval_loss(), *36*
evalerror_bin, 9
evalerror_bin(), *34*
evalerror_reg, 9
evalerror_reg(), *34*

generate_wxgb_params, 10
generate_wxgb_params(), *13*
glmnet, *18*
glmnet::glmnet(), *27, 28*

nhanes2013_sbc, 11

optim_wxgb_para, 12
optim_wxgb_para(), *8–10*

par_wElnet, 15
par_wElnet(), *2–4*, *18–20*, *25*
plot.par.w.elnet, 18
plot.par.w.elnet(), *2*, *20*
plot.w.elnet, 20
plot.w.elnet(), *4*
print.w.elnet, 21

randomForest, *31*
randomForest::randomForest, *29*, *32*
randomForest::randomForest(), *31*
replicate_weights, 22, *36*
replicate_weights(), *14*

survey::as.svrepdesign(), *7*
svydesign, *17*, *23*, *26*
svyROC::wauc(), *7*
svyVarSel::wlasso(), *28*

wElnet, *4, 5*, *20*, *21*, 25
wElnet(), *15*, *17, 18*, *30, 31*
wlasso, *25*
wRandomforest, 28
wXGBoost, *13*, 32
wXGBoost(), *8–10*, *14*, *30, 31*

xgboost::xgb.cv(), *11*
xgboost::xgb.params(), *34*
xgboost::xgb.train(), *8–11*, *13, 14*, *34*, *36*
xgboost::xgboost(), *32*