Package 'surveyCV'

May 9, 2026

Type Package

Title Cross Validation Based on Survey Design

Version 0.2.0

Date 2022-03-14

Description Functions to generate K-fold cross validation (CV) folds
and CV test error estimates that take into account
how a survey dataset's sampling design was constructed
(SRS, clustering, stratification, and/or unequal sampling weights).
You can input linear and logistic regression models, along with data and a
type of survey design in order to get an output that can help you determine
which model best fits the data using K-fold cross validation.
Our paper on ``K-Fold Cross-Validation for Complex Sample Surveys"
by Wieczorek, Guerin, and McMahon (2022)
<doi:10.1002/sta4.454>
explains why differing how we take folds based on survey design is useful.

License GPL-2 | GPL-3

Encoding UTF-8

LazyData TRUE

Depends R (>= 4.0)

Imports survey (>= 4.1), magrittr (>= 2.0)

Suggests dplyr (>= 1.0), ggplot2 (>= 3.3), grid (>= 4.0), gridExtra
(>= 2.3), ISLR (>= 1.2), knitr (>= 1.29), rmarkdown (>= 2.2),
rpms (>= 0.5), splines (>= 4.0), testthat (>= 3.1)

VignetteBuilder knitr

URL https://github.com/ColbyStatSvyRsch/surveyCV/

BugReports https://github.com/ColbyStatSvyRsch/surveyCV/issues

RoxygenNote 7.1.2

NeedsCompilation no

Author Cole Guerin [aut],
Thomas McMahon [aut],

Jerzy Wieczorek [cre, aut] (ORCID:
<https://orcid.org/0000-0002-2859-6534>),
Hunter Ratliff [ctb]

Maintainer Jerzy Wieczorek <jawieczo@colby.edu>

Repository CRAN

Date/Publication 2022-03-15 08:50:02 UTC

Contents

`cv.svy` *CV for survey data*

Description

This is a cross validation function designed for survey samples taken using a SRS, stratified, clustered, or clustered-and-stratified sampling design. Returns survey CV estimates of the mean loss for each model (MSE for linear models, or binary cross-entropy for logistic models).

Usage

```
cv.svy(
  Data,
  formulae,
  nfolds = 5,
  strataID = NULL,
  clusterID = NULL,
  nest = FALSE,
  fpcID = NULL,
  method = c("linear", "logistic"),
  weightsID = NULL,
  useSvyForFolds = TRUE,
  useSvyForFits = TRUE,
  useSvyForLoss = TRUE,
  na.rm = FALSE
)
```

Arguments

Data	Dataframe of dataset to be used for CV
formulae	Vector of formulas (as strings) for the GLMs to be compared in cross validation
nfolds	Number of folds to be used during cross validation, defaults to 5
strataID	String of the variable name used to stratify during sampling, must be the same as in the dataset used
clusterID	String of the variable name used to cluster during sampling, must be the same as in the dataset used
nest	Specify nest = TRUE if clusters are nested within strata, defaults to FALSE
fpcID	String of the variable name used for finite population corrections, must be the same as in the dataset used, see svydesign for details
method	String, must be either "linear" or "logistic", determines type of model fit during cross validation, defaults to linear
weightsID	String of the variable name in the dataset that contains sampling weights
useSvyForFolds	Specify useSvyForFolds = TRUE (default) to take svydesign into account when making folds; should not be set FALSE except for running simulations to understand the properties of surveyCV
useSvyForFits	Specify useSvyForFits = TRUE (default) to take svydesign into account when fitting models on training sets; should not be set FALSE except for running simulations to understand the properties of surveyCV
useSvyForLoss	Specify useSvyForLoss = TRUE (default) to take svydesign into account when calculating loss over test sets; should not be set FALSE except for running simulations to understand the properties of surveyCV
na.rm	Whether to drop cases with missing values when taking ‘svymean‘ of test losses

Details

If you have already created a svydesign object or fitted a svyglm, you will probably prefer the convenience wrapper functions cv.svydesign or cv.svyglm.

For models other than linear or logistic regression, you can use folds.svy or folds.svydesign to generate CV fold IDs that respect any stratification or clustering in the survey design. You can then carry out K-fold CV as usual, taking care to also use the survey design features and survey weights when fitting models in each training set and also when evaluating models against each test set.

Value

Object of class svystat, which is a named vector of survey CV estimates of the mean loss (MSE for linear models, or binary cross-entropy for logistic models) for each model, with names ".Model_1", ".Model_2", etc. corresponding to the models provided in formulae; and with a var attribute giving the variances. See surveysummary for details.

See Also

surveysummary, svydesign

cv.svydesign for a wrapper to use with a svydesign object, or cv.svyglm for a wrapper to use with a svyglm object

Examples

```
# Compare CV MSEs and their SEs under 3 linear models
# for a stratified sample and a one-stage cluster sample,
# using data from the `survey` package
library(survey)
data("api", package = "survey")
# stratified sample
cv.svy(apistrat, c("api00~ell",
                   "api00~ell+meals",
                   "api00~ell+meals+mobility"),
       nfolds = 5, strataID = "stype", weightsID = "pw", fpcID = "fpc")
# one-stage cluster sample
cv.svy(apiclus1, c("api00~ell",
                   "api00~ell+meals",
                   "api00~ell+meals+mobility"),
       nfolds = 5, clusterID = "dnum", weightsID = "pw", fpcID = "fpc")

# Compare CV MSEs and their SEs under 3 linear models
# for a stratified cluster sample with clusters nested within strata
data(NSFG_data)
library(splines)
cv.svy(NSFG_data, c("income ~ ns(age, df = 2)",
                    "income ~ ns(age, df = 3)",
                    "income ~ ns(age, df = 4)"),
       nfolds = 4,
       strataID = "strata", clusterID = "SECU",
       nest = TRUE, weightsID = "wgt")

# Logistic regression example, using the same stratified cluster sample;
# instead of CV MSE, we calculate CV binary cross-entropy loss,
# where (as with MSE) lower values indicate better fitting models
# (NOTE: na.rm=TRUE is not usually ideal;
#  it's used below purely for convenience, to keep the example short,
#  but a thorough analysis would look for better ways to handle the missing data)
cv.svy(NSFG_data, c("KnowPreg ~ ns(age, df = 1)",
                    "KnowPreg ~ ns(age, df = 2)",
                    "KnowPreg ~ ns(age, df = 3)"),
       method = "logistic", nfolds = 4,
       strataID = "strata", clusterID = "SECU",
       nest = TRUE, weightsID = "wgt",
       na.rm = TRUE)
```

`cv.svydesign` *CV for* `svydesign` *objects*

Description

Wrapper function which takes a `svydesign` object and a vector of model formulas (as strings), and passes it into `cv.svy`. Returns survey CV estimates of the mean loss for each model (MSE for linear models, or binary cross-entropy for logistic models).

Usage

```
cv.svydesign(
  design_object,
  formulae,
  nfolds = 5,
  method = c("linear", "logistic"),
  na.rm = FALSE
)
```

Arguments

design_object	Name of a svydesign object created using the survey package. We do not yet support use of probs or pps.
formulae	Vector of formulas (as strings) for the GLMs to be compared in cross validation
nfolds	Number of folds to be used during cross validation, defaults to 5
method	String, must be either "linear" or "logistic", determines type of model fit during cross validation, defaults to linear
na.rm	Whether to drop cases with missing values when taking ‘svymean‘ of test losses

Details

If you have already fitted a svyglm, you may prefer the convenience wrapper function cv.svyglm.

For models other than linear or logistic regression, you can use folds.svy or folds.svydesign to generate CV fold IDs that respect any stratification or clustering in the survey design. You can then carry out K-fold CV as usual, taking care to also use the survey design features and survey weights when fitting models in each training set and also when evaluating models against each test set.

Value

Object of class svystat, which is a named vector of survey CV estimates of the mean loss (MSE for linear models, or binary cross-entropy for logistic models) for each model, with names ".Model_1", ".Model_2", etc. corresponding to the models provided in formulae; and with a var attribute giving the variances. See surveysummary for details.

See Also

surveysummary, svydesign

cv.svyglm for a wrapper to use with a svyglm object

Examples

```
# Compare CV MSEs and their SEs under 3 linear models
# for a stratified sample and a one-stage cluster sample,
# using data from the `survey` package
library(survey)
data("api", package = "survey")
# stratified sample
dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
```

```
                    fpc = ~fpc)
cv.svydesign(formulae = c("api00~ell",
                          "api00~ell+meals",
                          "api00~ell+meals+mobility"),
             design_object = dstrat, nfolds = 5)
# one-stage cluster sample
dclus1 <- svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
cv.svydesign(formulae = c("api00~ell",
                          "api00~ell+meals",
                          "api00~ell+meals+mobility"),
             design_object = dclus1, nfolds = 5)

# Compare CV MSEs and their SEs under 3 linear models
# for a stratified cluster sample with clusters nested within strata
data(NSFG_data)
library(splines)
NSFG.svydes <- svydesign(id = ~SECU, strata = ~strata, nest = TRUE,
                         weights = ~wgt, data = NSFG_data)
cv.svydesign(formulae = c("income ~ ns(age, df = 2)",
                          "income ~ ns(age, df = 3)",
                          "income ~ ns(age, df = 4)"),
             design_object = NSFG.svydes, nfolds = 4)

# Logistic regression example, using the same stratified cluster sample;
# instead of CV MSE, we calculate CV binary cross-entropy loss,
# where (as with MSE) lower values indicate better fitting models
# (NOTE: na.rm=TRUE is not usually ideal;
#  it's used below purely for convenience, to keep the example short,
#  but a thorough analysis would look for better ways to handle the missing data)
cv.svydesign(formulae = c("KnowPreg ~ ns(age, df = 1)",
                          "KnowPreg ~ ns(age, df = 2)",
                          "KnowPreg ~ ns(age, df = 3)"),
             design_object = NSFG.svydes, nfolds = 4,
             method = "logistic", na.rm = TRUE)
```

`cv.svyglm` *CV for* `svyglm` *objects*

Description

Wrapper function which takes a `svyglm` object (which itself contains a `svydesign` object) and passes it through `cv.svydesign` to `cv.svy`. Chooses linear or logistic regression based on the `svyglm` object's value of `family`. Returns survey CV estimates of the mean loss for each model (MSE for linear models, or binary cross-entropy for logistic models).

Usage

```
cv.svyglm(glm_object, nfolds = 5, na.rm = FALSE)
```

Arguments

glm_object	Name of a svyglm object created from the survey package
nfolds	Number of folds to be used during cross validation, defaults to 5
na.rm	Whether to drop cases with missing values when taking 'svymean' of test losses

Details

If you have created a svydesign object and want to compare several svyglm models, you may prefer the function cv.svydesign.

For models other than linear or logistic regression, you can use folds.svy or folds.svydesign to generate CV fold IDs that respect any stratification or clustering in the survey design. You can then carry out K-fold CV as usual, taking care to also use the survey design features and survey weights when fitting models in each training set and also when evaluating models against each test set.

Value

Object of class svystat, which is a named vector with the survey CV estimate of the mean loss (MSE for linear models, or binary cross-entropy for logistic models) for the model in the svyglm object provided to glm_object; and with a var attribute giving the variance. See surveysummary for details.

See Also

surveysummary, svydesign, svyglm

cv.svydesign to use with a svydesign object for comparing several svyglm models

Examples

```
# Calculate CV MSE and its SE under one `svyglm` linear model
# for a stratified sample and a one-stage cluster sample,
# using data from the `survey` package
library(survey)
data("api", package = "survey")
# stratified sample
dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
                    fpc = ~fpc)
glmstrat <- svyglm(api00 ~ ell+meals+mobility, design = dstrat)
cv.svyglm(glmstrat, nfolds = 5)
# one-stage cluster sample
dclus1 <- svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
glmclus1 <- svyglm(api00 ~ ell+meals+mobility, design = dclus1)
cv.svyglm(glmclus1, nfolds = 5)

# Calculate CV MSE and its SE under one `svyglm` linear model
# for a stratified cluster sample with clusters nested within strata
data(NSFG_data)
library(splines)
NSFG.svydes <- svydesign(id = ~SECU, strata = ~strata, nest = TRUE,
                         weights = ~wgt, data = NSFG_data)
```

```
NSFG.svyglm <- svyglm(income ~ ns(age, df = 3), design = NSFG.svydes)
cv.svyglm(glm_object = NSFG.svyglm, nfolds = 4)

# Logistic regression example, using the same stratified cluster sample;
# instead of CV MSE, we calculate CV binary cross-entropy loss,
# where (as with MSE) lower values indicate better fitting models
# (NOTE: na.rm=TRUE is not usually ideal;
#  it's used below purely for convenience, to keep the example short,
#  but a thorough analysis would look for better ways to handle the missing data)
NSFG.svyglm.logreg <- svyglm(KnowPreg ~ ns(age, df = 2),
                             design = NSFG.svydes, family = quasibinomial())
cv.svyglm(glm_object = NSFG.svyglm.logreg, nfolds = 4, na.rm = TRUE)
```

`folds.svy` *Creating CV folds based on the survey design*

Description

This function creates a fold ID for each row in the dataset, to be used for carrying out cross validation on survey samples taken using a SRS, stratified, clustered, or clustered-and-stratified sampling design. Returns a vector of fold IDs, which in most cases you will want to append to your dataset using `cbind` or similar (see Examples below). These fold IDs respect any stratification or clustering in the survey design. You can then carry out K-fold CV as usual, taking care to also use the survey design features and survey weights when fitting models in each training set and also when evaluating models against each test set.

Usage

```
folds.svy(Data, nfolds, strataID = NULL, clusterID = NULL)
```

Arguments

`Data`	Dataframe of dataset
`nfolds`	Number of folds to be used during cross validation
`strataID`	String of the variable name used to stratify during sampling, must be the same as in the dataset used
`clusterID`	String of the variable name used to cluster during sampling, must be the same as in the dataset used

Details

If you have already created a `svydesign` object, you will probably prefer the convenience wrapper function `folds.svydesign`.

For the special cases of linear or logistic GLMs, use instead `cv.svy`, `cv.svydesign`, or `cv.svyglm` which will automate the whole CV process for you.

Value

Integer vector of fold IDs with length `nrow(Data)`. Most likely you will want to append the returned vector to your dataset, for instance with `cbind` (see Examples below).

See Also

`folds.svydesign` for a wrapper to use with a `svydesign` object

`cv.svy`, `cv.svydesign`, or `cv.svyglm` to carry out the whole CV process (not just forming folds but also training and testing your models) for linear or logistic regression models

Examples

```
# Set up CV folds for a stratified sample and a one-stage cluster sample,
# using data from the `survey` package
library(survey)
data("api", package = "survey")
# stratified sample
apistrat <- cbind(apistrat,
                  .foldID = folds.svy(apistrat, nfolds = 5, strataID = "stype"))
# Each fold will have observations from every stratum
with(apistrat, table(stype, .foldID))
# Fold sizes should be roughly equal
table(apistrat$.foldID)
#
# one-stage cluster sample
apiclus1 <- cbind(apiclus1,
                  .foldID = folds.svy(apiclus1, nfolds = 5, clusterID = "dnum"))
# For any given cluster, all its observations will be in the same fold;
# and each fold should contain roughly the same number of clusters
with(apiclus1, table(dnum, .foldID))
# But if cluster sizes are unequal,
# the number of individuals per fold will also vary
table(apiclus1$.foldID)
# See the end of `intro` vignette for an example of using such folds
# as part of a custom loop over CV folds
# to tune parameters in a design-consistent random forest model
```

`folds.svydesign` *Creating CV folds based on the* `svydesign` *object*

Description

Wrapper function which takes a `svydesign` object and desired number of CV folds, and passes it into `folds.svy`. Returns a vector of fold IDs, which in most cases you will want to append to your `svydesign` object using `update.svydesign` (see Examples below). These fold IDs respect any stratification or clustering in the survey design. You can then carry out K-fold CV as usual, taking care to also use the survey design features and survey weights when fitting models in each training set and also when evaluating models against each test set.

Usage

```
folds.svydesign(design_object, nfolds)
```

Arguments

design_object	Name of a svydesign object created using the survey package. The arguments id and strata (if used) must be specified as formulas, e.g. svydesign(ids = ~MyPSUs, ...).
nfolds	Number of folds to be used during cross validation

Details

For the special cases of linear or logistic GLMs, use instead cv.svydesign or cv.svyglm which will automate the whole CV process for you.

Value

Integer vector of fold IDs with length nrow(Data). Most likely you will want to append the returned vector to the svydesign object, for instance with update.svydesign (see Examples below).

See Also

folds.svy

cv.svy, cv.svydesign, or cv.svyglm to carry out the whole CV process (not just forming folds but also training and testing your models) for linear or logistic regression models

Examples

```
# Set up CV folds for a stratified sample and a one-stage cluster sample,
# using data from the `survey` package
library(survey)
data("api", package = "survey")
# stratified sample
dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
                    fpc = ~fpc)
dstrat <- update(dstrat, .foldID = folds.svydesign(dstrat, nfolds = 5))
# Each fold will have observations from every stratum
with(dstrat$variables, table(stype, .foldID))
# Fold sizes should be roughly equal
table(dstrat$variables$.foldID)
#
# one-stage cluster sample
dclus1 <- svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
dclus1 <- update(dclus1, .foldID = folds.svydesign(dclus1, nfolds = 5))
# For any given cluster, all its observations will be in the same fold;
# and each fold should contain roughly the same number of clusters
with(dclus1$variables, table(dnum, .foldID))
# But if cluster sizes are unequal,
# the number of individuals per fold will also vary
table(dclus1$variables$.foldID)
```

```
# See the end of `intro` vignette for an example of using such folds
# as part of a custom loop over CV folds
# to tune parameters in a design-consistent random forest model
```

NSFG_data *Subset of the 2015-2017 National Survey of Family Growth (NSFG): one birth per respondent.*

Description

We downloaded this data from the NSFG website and cleaned it following an approach posted to RPubs by Hunter Ratliff.

Usage

```
NSFG_data
```

Format

A data frame with 2801 rows and 17 variables:

CASEID Respondent ID number (per respondent, not per pregnancy)

LBW (originally LBW1) Low birthweight (TRUE/FALSE) for the 1st baby from this pregnancy

PreMe (recode of WKSGEST) Whether gestational age was premature (below 37 weeks) or full term

gotPNcare (recode of BGNPRENA) Whether or not respondent got prenatal care in first trimester (before 13 weeks)

KnowPreg (recode of KNEWPREG) Whether or not respondent learned she was pregnant by 6 weeks

age (originally AGECON) Age at time of conception

income (originally POVERTY) Income as percent of poverty level, so that 100 = income is at the poverty line; topcoded at 500

YrEdu (originally EDUCAT) Education (number of years of schooling)

race (originally HISPRACE) Race & Hispanic origin of respondent

BMI Body Mass Index

PregNum (originally PREGNUM) Respondent's total number of pregnancies

eduCat (originally HIEDUC) Highest completed year of school or highest degree received

GA (originally WKSGEST) Gestational length of completed pregnancy (in weeks)

Wanted (recode of NEWWANTR) Whether or not pregnancy came at right time according to respondent (rather than too soon, too late, or unwanted)

wgt (originally WGT2015_2017) Final weight for the 2015-2017 NSFG (at the respondent level, not pregnancy level)

SECU Randomized version of cluster ID, or "sampling error computational unit" – these are nested within strata

strata (originally SEST) Randomized version of stratum ID

Details

Note that these data were filtered down to include only:

- live births, - with gestational ages below 45 weeks, - born to mothers who were aged 20-40 years old at time of conception;

...then filtered further down to only the *first* such birth per respondent.

Also note that SECUs = Sampling Error Computation Units are effectively pseudo-PSUs, nested within (pseudo-)strata. See page 35 of the NSFG 2011-2013 sample design documentation for details.

Source

https://www.cdc.gov/nchs/nsfg/nsfg_2015_2017_puf.htm

https://rpubs.com/HunterRatliff1/NSFG_Wrangle

https://www.cdc.gov/nchs/data/nsfg/nsfg_2011_2013_sampledesign.pdf

---

NSFG_data_everypreg *Subset of the 2015-2017 National Survey of Family Growth (NSFG): all live births per respondent.*

Description

Same as 'NSFG_data' but using *every* birth, not just the *first* birth, out of the initial subset there (live births with gestational age < 45 weeks for mothers aged 20 to 40 at time of conception).

Usage

```
NSFG_data_everypreg
```

Format

A data frame with 5089 rows and 17 variables

---

surveyCV *surveyCV: Cross Validation Based on Survey Design*

Description

Functions to generate K-fold cross validation (CV) folds and CV test error estimates that take into account how a survey dataset's sampling design was constructed (SRS, clustering, stratification, and/or unequal sampling weights). You can input linear and logistic regression models, along with data and a type of survey design in order to get an output that can help you determine which model best fits the data using K-fold cross validation. Our paper on "K-Fold Cross-Validation for Complex Sample Surveys" by Wieczorek, Guerin, and McMahon (2022) <doi: 10.1002/sta4.454> explains why differing how we take folds based on survey design is useful.

Details

The code for this package seeks to create an alternative for the `boot::cv.glm` function, so that results correctly account for survey designs during K-fold cross validation.

Index