Package ‘cv’

July 22, 2025

Type Package

Title Cross-Validating Regression Models

Version 2.0.4

Date 2025-06-16

Description Cross-validation methods of regression models that exploit features of various modeling functions to improve speed. Some of the methods implemented in the package are novel, as described in the package vignettes; for general introductions to cross-validation, see, for example, Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2021, ISBN 978-1-0716-1417-4, Secs. 5.1, 5.3), ``An Introduction to Statistical Learning with Applications in R, Second Edition'', and Trevor Hastie, Robert Tibshirani, and Jerome Friedman (2009, ISBN 978-0-387-84857-0, Sec. 7.10), ``The Elements of Statistical Learning, Second Edition''.

Depends R (>= 3.5.0), doParallel

Imports car, foreach, glmmTMB, graphics, grDevices, gtools, insight, lattice, lme4, MASS, methods, nlme, parallel, stats, utils

Suggests boot, carData, dplyr, effects, ISLR2, knitr, latticeExtra, leaps, Metrics, microbenchmark, nnet, rmarkdown, spelling, testthat

LazyData TRUE

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

URL `https://gmonette.github.io/cv/`, `https://CRAN.R-project.org/package=cv`

BugReports `https://github.com/gmonette/cv/issues`

NeedsCompilation no

Author John Fox [aut] (ORCID: <`https://orcid.org/0000-0002-1196-8012`>), Georges Monette [aut, cre]

Maintainer Georges Monette <`georges+cv@yorku.ca`>

Repository CRAN

Date/Publication 2025-06-16 18:40:02 UTC

Contents

`cv` *Cross-Validate Regression Models*

Description

`cv()` is a parallelized generic k-fold (including n-fold, i.e., leave-one-out) cross-validation function, with a default method, specific methods for linear and generalized-linear models that can be much more computationally efficient, and a method for robust linear models. There are also `cv()` methods for mixed-effects models, for model-selection procedures, and for several models fit to the same data, which are documented separately.

Usage

```
cv(model, data, criterion, k, reps = 1L, seed, ...)

## Default S3 method:
cv(
  model,
  data = insight::get_data(model),
  criterion = mse,
  k = 10L,
  reps = 1L,
  seed = NULL,
  criterion.name = deparse(substitute(criterion)),
  details = k <= 10L,
  confint = n >= 400L,
  level = 0.95,
  ncores = 1L,
  type = "response",
  start = FALSE,
```

```
  model.function,
  ...
)

## S3 method for class 'lm'
cv(
  model,
  data = insight::get_data(model),
  criterion = mse,
  k = 10L,
  reps = 1L,
  seed = NULL,
  details = k <= 10L,
  confint = n >= 400L,
  level = 0.95,
  method = c("auto", "hatvalues", "Woodbury", "naive"),
  ncores = 1L,
  ...
)

## S3 method for class 'glm'
cv(
  model,
  data = insight::get_data(model),
  criterion = mse,
  k = 10L,
  reps = 1L,
  seed = NULL,
  details = k <= 10L,
  confint = n >= 400L,
  level = 0.95,
  method = c("exact", "hatvalues", "Woodbury"),
  ncores = 1L,
  start = FALSE,
  ...
)

## S3 method for class 'rlm'
cv(model, data, criterion, k, reps = 1L, seed, ...)

## S3 method for class 'cv'
print(x, digits = getOption("digits"), ...)

## S3 method for class 'cv'
summary(object, digits = getOption("digits"), ...)

## S3 method for class 'cvList'
print(x, ...)
```

```
## S3 method for class 'cvList'
summary(object, ...)

## S3 method for class 'cv'
plot(x, y, what = c("CV criterion", "coefficients"), ...)

## S3 method for class 'cvList'
plot(
  x,
  y,
  what = c("adjusted CV criterion", "CV criterion"),
  confint = TRUE,
  ...
)

cvInfo(object, what, ...)

## S3 method for class 'cv'
cvInfo(
  object,
 what = c("CV criterion", "adjusted CV criterion", "full CV criterion", "confint", "SE",
    "k", "seed", "method", "criterion name"),
  ...
)

## S3 method for class 'cvModList'
cvInfo(
  object,
 what = c("CV criterion", "adjusted CV criterion", "full CV criterion", "confint", "SE",
    "k", "seed", "method", "criterion name"),
  ...
)

## S3 method for class 'cvList'
cvInfo(
  object,
 what = c("CV criterion", "adjusted CV criterion", "full CV criterion", "confint", "SE",
    "k", "seed", "method", "criterion name"),
  ...
)

## S3 method for class 'cv'
as.data.frame(
  x,
  row.names = NULL,
  optional = TRUE,
  rows = c("cv", "folds"),
```

```
  columns = c("criteria", "coefficients"),
  ...
)

## S3 method for class 'cvList'
as.data.frame(x, row.names = NULL, optional = TRUE, ...)

## S3 method for class 'cvDataFrame'
print(x, digits = getOption("digits") - 2L, ...)

## S3 method for class 'cvDataFrame'
summary(
  object,
  formula,
  subset = NULL,
  fun = mean,
  include = c("cv", "folds", "all"),
  ...
)
```

Arguments

`model`	a regression model object (see Details).
`data`	data frame to which the model was fit (not usually necessary).
`criterion`	cross-validation criterion ("cost" or lack-of-fit) function of form `f(y, yhat)` where `y` is the observed values of the response and `yhat` the predicted values; the default is `mse` (the mean-squared error).
`k`	perform k-fold cross-validation (default is `10`); `k` may be a number or `"loo"` or `"n"` for n-fold (leave-one-out) cross-validation.
`reps`	number of times to replicate k-fold CV (default is `1`).
`seed`	for R's random number generator; optional, if not supplied a random seed will be selected and saved; not needed for n-fold cross-validation.
`...`	to match generic; passed to `predict()` for the default `cv()` method; passed to the `Tapply()` function in the **car** package for `summary.cvDataFrame()`; passed to default `plot()` method for `plot.cvList()` or `plot.cv()`.
`criterion.name`	a character string giving the name of the CV criterion function in the returned `"cv"` object (not usually needed).
`details`	if `TRUE` (the default if the number of folds `k <= 10`), save detailed information about the value of the CV criterion for the cases in each fold and the regression coefficients with that fold deleted.
`confint`	if `TRUE` (the default if the number of cases is 400 or greater), compute a confidence interval for the bias-corrected CV criterion, if the criterion is the average of casewise components; for `plot.cvList()`, whether to plot confidence intervals around the biased-adjusted CV criterion, defaulting to `TRUE` and applicable only if confidence intervals are included in the `"cv"` object.
`level`	confidence level (default `0.95`).

`ncores`	number of cores to use for parallel computations (default is `1`, i.e., computations aren't done in parallel).
`type`	for the default method, value to be passed to the `type` argument of `predict()`; the default is `type="response"`, which is appropriate, e.g., for a `"glm"` model and may be recognized or ignored by `predict()` methods for other model classes.
`start`	if `TRUE` (the default is `FALSE`), the `start` argument to `update()` is set to the vector of regression coefficients for the model fit to the full data, possibly making the CV updates faster, e.g., for a GLM.
`model.function`	a regression function, typically for a new `cv()` method that that calls `cv.default()` via `NextMethod()`, residing in a package that's not a declared dependency of the **cv** package, e.g., `nnet::multinom`. It's usually not necessary to specify `model.function` to make `cv.default()` work.
`method`	computational method to apply to a linear (i.e., `"lm"`) model or to a generalized linear (i.e., `"glm"`) model. See Details for an explanation of the available options.
`x`	a `"cv"`, `"cvList"`, or `"cvDataFrame"` object to be plotted or summarized.
`digits`	significant digits for printing, default taken from the `"digits"` option.
`object`	an object to summarize or a `"cv"`, `"cvModlist"`, or `"cvList"` object from which to extract information via `cvInfo()`.
`y`	to match the `plot()` generic function, ignored.
`what`	for `plot()` methods, what to plot: for the `"cv"` method, either `"CV criterion"` (the default), or `"coefficients"`; for the `"cvList"` method, either `"adjusted CV criterion"` (the default if present in the `"cv"` object) or `"CV object"`. For `cvInfo()`, the information to extract from a `"cv"`, `"cvModList"`, or `"cvList"` object, one of: `"CV criterion"`, `"adjusted CV criterion"`, `"full CV criterion"` (the CV criterion applied to the model fit to the full data set), `"SE"` (the standard error of the adjusted CV criterion), `"confint"` (confidence interval for the adjusted CV criterion), `"k"`, (the number of folds), `"seed"` (the seed employed for R's random-number generator), `"method"` (the computational method employed, e.g., for a `"lm"` model object), or `"criterion name"` (the CV criterion employed); not all of these elements may be present, in which case `cvInfo()` would return `NULL`. Partial matching is supported, so, e.g., `cvInfo(cv-object, "adjusted")` is equivalent to `cvInfo(cv-object, "adjusted CV criterion")`
`row.names`	optional row names for the result, defaults to `NULL`.
`optional`	to match the `as.data.frame()` generic function; if `FALSE` (the default is `TRUE`), then the names of the columns of the returned data frame, including the names of coefficients, are coerced to syntactically correct names.
`rows`	the rows of the resulting data frame to retain: setting `rows="cv"` retains rows pertaining to the overall CV result (marked as `"fold 0"`); setting `rows="folds"` retains rows pertaining to individual folds 1 through k; the default is `rows = c("cv", "folds")`, which retains all rows.

`columns`	the columns of the resulting data frame to retain: setting `columns="critera"` retains columns pertaining to CV criteria; setting `columns="coefficients"` retains columns pertaining to model coefficients (broadly construed); the default is `columns = c("criteria", "coefficients")`, which retains both; and the columns `"model"`, `"rep"`, and `"fold"`, if present, are always retained.
`formula`	of the form `some.criterion ~ classifying.variable(s)` (see examples).
`subset`	a subsetting expression; the default (`NULL`) is not to subset the `"cvDataFrame"` object.
`fun`	summary function to apply, defaulting to `mean`.
`include`	which rows of the `"cvDataFrame"` to include in the summary. One of `"cv"` (the default), rows representing the overall CV results; `"folds"`, rows for individual folds; `"all"`, all rows (generally not sensible).

Details

The default `cv()` method uses `update()` to refit the model to each fold, and should work if there are appropriate `update()` and `predict()` methods, and if the default method for `GetResponse()` works or if a `GetResponse()` method is supplied. The model must, however, work correctly with `update()`, and in particular not have variables in the model formula that aren't in the data to which the model was fit: see the last example.

The `"lm"` and `"glm"` methods can use much faster computational algorithms, as selected by the `method` argument. The linear-model method accommodates weighted linear models.

For both classes of models, for the leave-one-out (n-fold) case, fitted values for the folds can be computed from the hat-values via `method="hatvalues"` without refitting the model; for GLMs, this method is approximate, for LMs it is exact.

Again for both classes of models, when more than one case is omitted in each fold, fitted values may be obtained without refitting the model by exploiting the Woodbury matrix identity via `method="Woodbury"`. As for hatvalues, this method is exact for LMs and approximate for GLMs.

The default for linear models is `method="auto"`, which is equivalent to `method="hatvalues"` for n-fold cross-validation and `method="Woodbury"` otherwise; `method="naive"` refits the model via `update()` and is generally much slower. The default for generalized linear models is `method="exact"`, which employs `update()`. This default is conservative, and it is usually safe to use `method="hatvalues"` for n-fold CV or `method="Woodbury"` for k-fold CV.

There is also a method for robust linear models fit by `rlm()` in the **MASS** package (to avoid inheriting the `"lm"` method for which the default `"auto"` computational method would be inappropriate).

For additional details, see the "Cross-validating regression models" vignette (`vignette("cv", package="cv")`).

`cv()` is designed to be extensible to other classes of regression models; see the "Extending the cv package" vignette (`vignette("cv-extend", package="cv")`).

Value

The `cv()` methods return an object of class `"cv"`, with the CV criterion (`"CV crit"`), the bias-adjusted CV criterion (`"adj CV crit"`), the criterion for the model applied to the full data (`"full crit"`), the confidence interval and level for the bias-adjusted CV criterion (`"confint"`), the number of folds (`"k"`), and the seed for R's random-number generator (`"seed"`). If `details=TRUE`, then the returned object will also include a `"details"` component, which is a list of two elements: `"criterion"`, containing the CV criterion computed for the cases in each fold; and `"coefficients"`, regression coefficients computed for the model with each fold deleted. Some methods may return a subset of these components and may add additional information. If `reps` > `1`, then an object of class `"cvList"` is returned, which is literally a list of `"cv"` objects.

Methods (by class)

- `cv(default)`: `"default"` method.
- `cv(lm)`: `"lm"` method.
- `cv(glm)`: `"glm"` method.
- `cv(rlm)`: `"rlm"` method (to avoid inheriting the `"lm"` method).

Methods (by generic)

- `print(cv)`: `print()` method for `"cv"` objects.
- `summary(cv)`: `summary()` method for `"cv"` objects.
- `plot(cv)`: `plot()` method for `"cv"` objects.
- `as.data.frame(cv)`: `as.data.frame()` method for `"cv"` objects.

Functions

- `print(cvList)`: `print()` method for `"cvList"` objects.
- `summary(cvList)`: `summary()` method for `"cvList"` objects.
- `plot(cvList)`: `plot()` method for `"cvList"` objects.
- `cvInfo()`: extract information from a `"cv"` object.
- `as.data.frame(cvList)`: `as.data.frame()` method for `"cvList"` objects.
- `print(cvDataFrame)`: `print()` method for `"cvDataFrame"` objects.
- `summary(cvDataFrame)`: `summary()` method for `"cvDataFrame"` objects.

See Also

`cv.merMod`, `cv.function`, `cv.modList`.

Examples

```
if (requireNamespace("ISLR2", quietly=TRUE)){
withAutoprint({
data("Auto", package="ISLR2")
m.auto <- lm(mpg ~ horsepower, data=Auto)
cv(m.auto,  k="loo")
summary(cv(m.auto,  k="loo"))
summary(cv.auto <- cv(m.auto, seed=1234))
compareFolds(cv.auto)
plot(cv.auto)
plot(cv.auto, what="coefficients")
summary(cv.auto.reps <- cv(m.auto, seed=1234, reps=3))
```

```
cvInfo(cv.auto.reps, what="adjusted CV criterion")
plot(cv.auto.reps)
plot(cv(m.auto, seed=1234, reps=10, confint=TRUE))
D.auto.reps <- as.data.frame(cv.auto.reps)
head(D.auto.reps)
summary(D.auto.reps, mse ~ rep + fold, include="folds")
summary(D.auto.reps, mse ~ rep + fold, include = "folds",
        subset = fold <= 5) # first 5 folds
summary(D.auto.reps, mse ~ rep, include="folds")
summary(D.auto.reps, mse ~ rep, fun=sd, include="folds")
})
} else {
cat("\n install 'ISLR2' package to run these examples\n")
}

if (requireNamespace("carData", quietly=TRUE)){
withAutoprint({
data("Mroz", package="carData")
m.mroz <- glm(lfp ~ ., data=Mroz, family=binomial)
summary(cv.mroz <- cv(m.mroz, criterion=BayesRule, seed=123))
cvInfo(cv.mroz)
cvInfo(cv.mroz, "adjusted")
cvInfo(cv.mroz, "confint")

data("Duncan", package="carData")
m.lm <- lm(prestige ~ income + education, data=Duncan)
m.rlm <- MASS::rlm(prestige ~ income + education,
                   data=Duncan)
summary(cv(m.lm, k="loo", method="Woodbury"))
summary(cv(m.rlm, k="loo"))
})
} else {
cat("\n install 'carData' package to run these examples\n")
}

# the following (due to Joshua Philipp Entrop)
# produces an error:
## Not run:
data("Auto", package="ISLR2")
Auto$mpg_20 <- as.numeric(Auto$mpg < 20)
mlist <- lapply(
                1:3,
                \(p) glm(mpg_20  ~ poly(horsepower, p), data = Auto)
)
cv(
   models(mlist),
   data = Auto,
   seed = 2120)

## End(Not run)
```

`cv.function` *Cross-Validate a Model-Selection Procedure*

Description

The `cv()` `"function"` method is a general function to cross-validate a model-selection procedure, such as the following: `selectStepAIC()` is a procedure that applies the `stepAIC()` model-selection function in the **MASS** package; `selectTrans()` is a procedure for selecting predictor and response transformations in regression, which uses the `powerTransform()` function in the **car** package; `selectTransAndStepAIC()` combines predictor and response transformations with predictor selection; and `selectModelList()` uses cross-validation to select a model from a list of models created by `models()` and employs (meta) cross-validation to assess the predictive accuracy of this procedure.

Usage

```
## S3 method for class '`function`'
cv(
  model,
  data,
  criterion = mse,
  k = 10L,
  reps = 1L,
  seed = NULL,
  working.model = NULL,
  y.expression = NULL,
  confint = n >= 400L,
  level = 0.95,
  details = k <= 10L,
  save.model = FALSE,
  ncores = 1L,
  ...
)

selectStepAIC(
  data,
  indices,
  model,
  criterion = mse,
  AIC = TRUE,
  details = TRUE,
  save.model = FALSE,
  ...
)

selectTrans(
  data,
```

```
  indices,
  details = TRUE,
  save.model = FALSE,
  model,
  criterion = mse,
  predictors,
  response,
  family = c("bcPower", "bcnPower", "yjPower", "basicPower"),
  family.y = c("bcPower", "bcnPower", "yjPower", "basicPower"),
  rounded = TRUE,
  ...
)

selectTransStepAIC(
  data,
  indices,
  details = TRUE,
  save.model = FALSE,
  model,
  criterion = mse,
  predictors,
  response,
  family = c("bcPower", "bcnPower", "yjPower", "basicPower"),
  family.y = c("bcPower", "bcnPower", "yjPower", "basicPower"),
  rounded = TRUE,
  AIC = TRUE,
  ...
)

selectModelList(
  data,
  indices,
  model,
  criterion = mse,
  k = 10L,
  k.meta = k,
  details = k <= 10L,
  save.model = FALSE,
  seed = FALSE,
  quietly = TRUE,
  ...
)

compareFolds(object, digits = 3, ...)

## S3 method for class 'cvSelect'
coef(object, average, NAs = 0, ...)
```

```
## S3 method for class 'cvSelect'
cvInfo(
  object,
  what = c("CV criterion", "adjusted CV criterion", "full CV criterion", "confint", "SE",
    "k", "seed", "method", "criterion name", "selected model"),
  ...
)
```

Arguments

`model`	a regression model object fit to data, or for the `cv()` `"function"` method, a model-selection procedure function (see Details).
`data`	full data frame for model selection.
`criterion`	a CV criterion ("cost" or lack-of-fit) function.
`k`	perform k-fold cross-validation (default is 10); `k` may be a number or `"loo"` or `"n"` for n-fold (leave-one-out) cross-validation.
`reps`	number of times to replicate k-fold CV (default is `1`)
`seed`	for R's random number generator; not used for n-fold cross-validation. If not explicitly set, a seed is randomly generated and saved to make the results reproducible. In some cases, for internal use only, `seed` is set to `FALSE` to suppress automatically setting the seed.
`working.model`	a regression model object fit to data, typically to begin a model-selection process; for use with `selectModelList()`, a list of competing models created by `models()`.
`y.expression`	normally the response variable is found from the `model` or `working.model` argument; but if, for a particular selection procedure, the `model` or `working.model` argument is absent, or if the response can't be inferred from the model, the response can be specified by an expression, such as `expression(log(income))`, to be evaluated within the data set provided by the `data` argument.
`confint`	if `TRUE` (the default if the number of cases is 400 or greater), compute a confidence interval for the bias-corrected CV criterion, if the criterion is the average of casewise components.
`level`	confidence level (default `0.95`).
`details`	if `TRUE`, save detailed information about the value of the CV criterion for the cases in each fold and the regression coefficients (and possibly other information) with that fold deleted; default is `TRUE` if `k` is 10 or smaller, `FALSE` otherwise.
`save.model`	save the model that's selected using the *full* data set (default, `FALSE`).
`ncores`	number of cores to use for parallel computations (default is `1`, i.e., computations aren't done in parallel)
`...`	for `cvSelect()` and the `cv()` `"function"` method, arguments to be passed to `procedure()`; for `selectStepAIC()` and `selectTransStepAIC()`, arguments to be passed to `stepAIC()`.
`indices`	indices of cases in data defining the current fold.

`AIC`	if `TRUE` (the default) use the AIC as the model-selection criterion; if `FALSE`, use the BIC. The `k` argument to `stepAIC()` is set accordingly (note that this is distinct from the number of folds `k`).
`predictors`	character vector of names of the predictors in the model to transform; if missing, no predictors will be transformed.
`response`	name of the response variable; if missing, the response won't be transformed.
`family`	transformation family for the predictors, one of `"bcPower", "bcnPower", "yjPower", "basicPower"`, with `"bcPower"` as the default. These are the names of transformation functions in the **car** package; see `bcPower()`.
`family.y`	transformation family for the response, with `"bcPower"` as the default.
`rounded`	if `TRUE` (the default) use nicely rounded versions of the estimated transformation parameters (see `bcPower()`).
`k.meta`	the number of folds for meta CV; defaults to the value of `k`; may be specified as `"loo"` or `"n"` as well as an integer.
`quietly`	if `TRUE` (the default), simple messages (for example about the value to which the random-number generator seed is set), but not warnings or errors, are suppressed.
`object`	an object of class `"cvSelect"`.
`digits`	significant digits for printing coefficients (default `3`).
`average`	if supplied, a function, such as `mean` or `median`, to use us in averaging estimates across folds; if missing, the estimates for each fold are returned.
`NAs`	values to substitute for `NA`s in calculating averaged estimates; the default, `0`, is appropriate, e.g., for regression coefficients; the value `1` might be appropriate for power-transformation estimates.
`what`	the information to extract from a `"cvSelect"` object, one of: `"CV criterion"`, `"adjusted CV criterion"`, `"full CV criterion"` (the CV criterion applied to the model fit to the full data set), `"SE"` (the standard error of the adjusted CV criterion), `"confint"` (confidence interval for the adjusted CV criterion), `"k"`, (the number of folds), `"seed"` (the seed employed for R's random-number generator), `"method"` (the computational method employed, e.g., for a `"lm"` model object), `"criterion name"` (the CV criterion employed), or `"selected model"` (the model object for the model that was selected); not all of these elements may be present, in which case `cvInfo()` would return `NULL`.

Details

The model-selection function supplied as the `procedure` (for `cvSelect()`) or `model` (for `cv()`) argument should accept the following arguments:

`data` set to the `data` argument to `cvSelect()` or `cv()`.

`indices` the indices of the rows of `data` defining the current fold; if missing, the model-selection procedure is applied to the full `data`.

other arguments to be passed via `...` from `cvSelect()` or `cv()`.

`procedure()` or `model()` should return a list with the following named elements: `fit.i`, the vector of predicted values for the cases in the current fold computed from the model omitting these cases; `crit.all.i`, the CV criterion computed for all of the cases using the model omitting the current fold; and (optionally) `coefficients`, parameter estimates from the model computed omitting the current fold.

When the `indices` argument is missing, `procedure()` returns the cross-validation criterion for all of the cases based on the model fit to all of the cases.

For examples of model-selection functions for the `procedure` argument, see the code for `selectStepAIC()`, `selectTrans()`, and `selectTransAndStepAIC()`.

For additional information, see the "Cross-validating model selection" vignette (`vignette("cv-select", package="cv")`) and the "Extending the cv package" vignette (`vignette("cv-extend", package="cv")`).

Value

An object of class `"cvSelect"`, inheriting from class `"cv"`, with the CV criterion (`"CV crit"`), the bias-adjusted CV criterion (`"adj CV crit"`), the criterion for the model applied to the full data (`"full crit"`), the confidence interval and level for the bias-adjusted CV criterion (`"confint"`), the number of folds (`"k"`), the seed for R's random-number generator (`"seed"`), and (optionally) a list of coefficients (or, in the case of `selectTrans()`, estimated transformation parameters, and in the case of `selectTransAndStepAIC()`, both regression coefficients and transformation parameters) for the selected models for each fold (`"coefficients"`). If `reps` > `1`, then an object of class `c("cvSelectList", "cvList")` is returned, which is literally a list of `c("cvSelect", "cv")` objects.

Functions

- `` cv(`function`) ``: `cv()` method for applying a model model-selection (or specification) procedure.
- `selectStepAIC()`: select a regression model using the `stepAIC()` function in the **MASS** package.
- `selectTrans()`: select transformations of the predictors and response using `powerTransform()` in the **car** package.
- `selectTransStepAIC()`: select transformations of the predictors and response, and then select predictors.
- `selectModelList()`: select a model using (meta) CV.
- `compareFolds()`: print the coefficients from the selected models for the several folds.
- `coef(cvSelect)`: extract the coefficients from the selected models for the several folds and possibly average them.

See Also

`stepAIC`, `bcPower`, `powerTransform`, `cv`.

Examples

```
if (requireNamespace("ISLR2", quietly=TRUE)){
withAutoprint({
data("Auto", package="ISLR2")
m.auto <- lm(mpg ~ . - name - origin, data=Auto)
cv(selectStepAIC, Auto, seed=123, working.model=m.auto)
cv(selectStepAIC, Auto, seed=123, working.model=m.auto,
         AIC=FALSE, k=5, reps=3) # via BIC
})
} else {
cat("\n install the 'ISLR2' package to run these examples\n")
}
if (requireNamespace("carData", quietly=TRUE)){
withAutoprint({
data("Prestige", package="carData")
m.pres <- lm(prestige ~ income + education + women,
             data=Prestige)
cvt <- cv(selectTrans, data=Prestige, working.model=m.pres, seed=123,
          predictors=c("income", "education", "women"),
          response="prestige", family="yjPower")
cvt
compareFolds(cvt)
coef(cvt, average=median, NAs=1) # NAs not really needed here
cv(m.pres, seed=123)
})
} else {
cat("install the 'carData' package to run these examples\n")
}
if (requireNamespace("ISLR2", quietly=TRUE)){
withAutoprint({
Auto$year <- as.factor(Auto$year)
Auto$origin <- factor(Auto$origin,
                      labels=c("America", "Europe", "Japan"))
rownames(Auto) <- make.names(Auto$name, unique=TRUE)
Auto$name <- NULL
m.auto <- lm(mpg ~ . , data=Auto)
cvs <- cv(selectTransStepAIC, data=Auto, seed=76692, working.model=m.auto,
          criterion=medAbsErr,
          predictors=c("cylinders", "displacement", "horsepower",
                       "weight", "acceleration"),
          response="mpg", AIC=FALSE)
cvs
compareFolds(cvs)
})
}
data("Duncan", package="carData")
m1 <- lm(prestige ~ income + education, data=Duncan)
m2 <- lm(prestige ~ income + education + type, data=Duncan)
m3 <- lm(prestige ~ (income + education)*type, data=Duncan)
summary(cv.sel <- cv(selectModelList, data=Duncan, seed=5963,
                     working.model=models(m1, m2, m3),
                     save.model=TRUE)) # meta CV
```

```
cvInfo(cv.sel, "selected model")
```

`cv.merMod` *Cross-Validate Mixed-Effects Model*

Description

`cv()` methods for mixed-effect models of class `"merMod"`, fit by the `lmer()` and `glmer()` functions in the **lme4** package; for models of class `"lme"` fit by the `lme()` function in the **nlme** package; and for models of class `"glmmTMB"` fit by the `glmmTMB()` function in the **glmmTMB** package.

Usage

```
## S3 method for class 'merMod'
cv(
  model,
  data = insight::get_data(model),
  criterion = mse,
  k = NULL,
  reps = 1L,
  seed,
  details = NULL,
  ncores = 1L,
  clusterVariables,
  ...
)

## S3 method for class 'lme'
cv(
  model,
  data = insight::get_data(model),
  criterion = mse,
  k = NULL,
  reps = 1L,
  seed,
  details = NULL,
  ncores = 1L,
  clusterVariables,
  ...
)

## S3 method for class 'glmmTMB'
cv(
  model,
  data = insight::get_data(model),
  criterion = mse,
```

```
  k = NULL,
  reps = 1L,
  seed,
  details = NULL,
  ncores = 1L,
  clusterVariables,
  ...
)
```

Arguments

`model`	a mixed-effects model object for which a `cv()` method is available.
`data`	data frame to which the model was fit (not usually necessary).
`criterion`	cross-validation ("cost" or lack-of-fit) criterion function of form `f(y, yhat)` where `y` is the observed values of the response and `yhat` the predicted values; the default is `mse` (the mean-squared error).
`k`	perform k-fold cross-validation; `k` may be a number or `"loo"` or `"n"` for n-fold (leave-one-out) cross-validation; the default is `10` if cross-validating individual cases and `"loo"` if cross-validating clusters.
`reps`	number of times to replicate k-fold CV (default is `1`), or greater), compute a confidence interval for the bias-corrected CV criterion, if the criterion is the average of casewise components.
`seed`	for R's random number generator; optional, if not supplied a random seed will be selected and saved; not needed for n-fold cross-validation.
`details`	if `TRUE` (the default if the number of folds `k <= 10`), save detailed information about the value of the CV criterion for the cases in each fold and the regression coefficients with that fold deleted.
`ncores`	number of cores to use for parallel computations (default is `1`, i.e., computations aren't done in parallel).
`clusterVariables`	a character vector of names of the variables defining clusters for a mixed model with nested or crossed random effects; if missing, cross-validation is performed for individual cases rather than for clusters.
`...`	for `cv()` methods, to match generic, and for `cvMixed()`, arguments to be passed to `update()`.

Details

For mixed-effects models, cross-validation can be done by "clusters" or by individual observations. If the former, predictions are based only on fixed effects; if the latter, predictions include the random effects (i.e., are the best linear unbiased predictors or "BLUPS").

The model supplied must work properly with `update()`, and in particular the formula for the model should not include variables that are not in the data set to which the model was fit. See the last (faulty) example in the help for `cv()`.

Value

The methods `cv.merMod()`, `cv.lme()`, and `cv.glmmTMB()`, return objects of class `"cv"`, or, if `reps` > 1, of class `"cvList"` (see `cv()`).

Functions

- `cv(merMod)`: `cv()` method for `lmer()` and `glmer()` models from the **lme4** package.
- `cv(lme)`: `cv()` method for `lme()` models from the **nlme** package.
- `cv(glmmTMB)`: `cv()` method for `glmmTMB()` models from the **glmmTMB** package.

See Also

`cv`, `lmer`, `glmer`, `lme`, `glmmTMB`

Examples

```
library("lme4")
# from ?lmer:
(fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy))
summary(cv(fm1, clusterVariables="Subject")) # LOO CV of clusters
summary(cv(fm1, seed=447)) # 10-fold CV of cases
summary(cv(fm1, clusterVariables="Subject", k=5,
   seed=834, reps=3)) # 5-fold CV of clusters, repeated 3 times

library("nlme")
# from ?lme
(fm2 <- lme(distance ~ age + Sex, data = Orthodont,
            random = ~ 1))
summary(cv(fm2)) # LOO CV of cases
summary(cv(fm2, clusterVariables="Subject",
        k=5, seed=321)) # 5-fold CV of clusters

library("glmmTMB")
# from ?glmmTMB
(m1 <- glmmTMB(count ~ mined + (1|site),
               zi=~mined,
               family=poisson, data=Salamanders))
summary(cv(m1, seed=97816, k=5,
          clusterVariables="site")) # 5-fold CV of clusters
summary(cv(m1, seed=34506, k=5)) # 5-fold CV of cases
```

`cv.modList` *Cross-Validate Several Models Fit to the Same Data*

Description

A cv() method for an object of class "modlist", created by the models() function. This cv() method simplifies the process of cross-validating several models on the same set of CV folds and may also be used for meta CV, where CV is used to select one from among several models. models() performs some "sanity" checks, warning if the models are of different classes, and reporting an error if they are fit to apparently different data sets or different response variables.

Usage

```
## S3 method for class 'modList'
cv(
  model,
  data,
  criterion = mse,
  k,
  reps = 1L,
  seed,
  quietly = TRUE,
  meta = FALSE,
  ...
)

models(...)

## S3 method for class 'cvModList'
print(x, ...)

## S3 method for class 'cvModList'
summary(object, ...)

## S3 method for class 'cvModList'
plot(
  x,
  y,
  spread = c("range", "sd"),
  confint = TRUE,
  xlab = "",
  ylab,
  main,
  axis.args = list(labels = names(x), las = 3L),
  col = palette()[2L],
  lwd = 2L,
  grid = TRUE,
  ...
)

## S3 method for class 'cvModList'
as.data.frame(x, row.names = NULL, optional = TRUE, ...)
```

Arguments

`model`	a list of regression model objects, created by `models()`.
`data`	(required) the data set to which the models were fit.
`criterion`	the CV criterion ("cost" or lack-of-fit) function, defaults to `mse`.
`k`	the number of CV folds; may be omitted, in which case the value will depend on the default for the `cv()` method invoked for the individual models.
`reps`	number of replications of CV for each model (default is 1).
`seed`	(optional) seed for R's pseudo-random-number generator, to be used to create the same set of CV folds for all of the models; if omitted, a seed will be randomly generated and saved. Not used for leave-one-out CV.
`quietly`	if `TRUE` (the default), simple messages (for example about the value to which the random-number generator seed is set), but not warnings or errors, are suppressed.
`meta`	if `TRUE` (the default is `FALSE`), cross-validation is performed recursively to select a "best" model deleting each fold in turn by calculating the CV estimate of the criterion for the remaining folds; this is equivalent to employing the `selectModelList()` model-selection procedure.
`...`	for `cv.modList()`, additional arguments to be passed to the `cv()` method applied to each model. For `models()`, two or more competing models fit to the the same data; the several models may be named. It is also possible to specify a single argument, which should then be list of models (which has the effect of turning a list of models into a `"modList"` object). For the `print()` method, arguments to be passed to the `print()` method for the individual model cross-validations. For the `plot()` method, arguments to be passed to the base `plot()` function.
`x`	an object of class `"cvModList"` to be printed or plotted.
`object`	an object to summarize.
`y`	the name of the element in each `"cv"` object to be plotted; defaults to `"adj CV crit"`, if it exists, or to `"CV crit"`.
`spread`	if `"range"`, the default, show the range of CV criteria for each model along with their average; if `"sd"`, show the average plus or minus 1 standard deviation.
`confint`	if `TRUE` (the default) and if confidence intervals are in any of the `"cv"` objects, then plot the confidence intervals around the CV criteria.
`xlab`	label for the x-axis (defaults to blank).
`ylab`	label for the y-axis (if missing, a label is constructed).
`main`	main title for the graph (if missing, a label is constructed).
`axis.args`	a list of arguments for the `axis()` function, used to draw the horizontal axis. In addition to the axis arguments given explicitly, `side=1` (the horizontal axis) and `at=seq(along=x)` (i.e., 1 to the number of models) are used and can't be modified.

`col`	color for the line and points, defaults to the second element of the color palette; see `palette()`.
`lwd`	line width for the line (defaults to 2).
`grid`	if `TRUE` (the default), include grid lines on the graph.
`row.names`	optional row names for the result, defaults to `NULL`.
`optional`	to match the `as.data.frame()` generic function; if `FALSE` (the default is `TRUE`), then the names of the columns of the returned data frame, including the names of coefficients, are coerced to syntactically correct names.

Value

`models()` returns a `"modList"` object, the `cv()` method for which returns a `"cvModList"` object, or, when `meta=TRUE`, an object of class `c("cvSelect", "cv")`.

Functions

- `cv(modList)`: `cv()` method for `"modList"` objects.
- `models()`: create a list of models.
- `print(cvModList)`: `print()` method for `"cvModList"` objects.
- `summary(cvModList)`: `summary()` method for `"cvModList"` objects.
- `plot(cvModList)`: `plot()` method for `"cvModList"` objects.
- `as.data.frame(cvModList)`: `as.data.frame()` method for `"cvModList"` objects.

See Also

`cv`, `cv.merMod`, `selectModelList`.

Examples

```
if (requireNamespace("carData", quietly=TRUE)){
withAutoprint({
data("Duncan", package="carData")
m1 <- lm(prestige ~ income + education, data=Duncan)
m2 <- lm(prestige ~ income + education + type, data=Duncan)
m3 <- lm(prestige ~ (income + education)*type, data=Duncan)
(cv.models <- cv(models(m1=m1, m2=m2, m3=m3),
                 data=Duncan, seed=7949, reps=5))
D.cv.models <- as.data.frame(cv.models)
head(D.cv.models)
summary(D.cv.models, criterion ~ model + rep, include="folds")
plot(cv.models)
(cv.models.ci <- cv(models(m1=m1, m2=m2, m3=m3),
                    data=Duncan, seed=5963, confint=TRUE, level=0.50))
                 # nb: n too small for accurate CIs
plot(cv.models.ci)
(cv.models.meta <- cv(models(m1=m1, m2=m2, m3=m3),
                      data=Duncan, seed=5963,
                      meta=TRUE, save.model=TRUE))
```

```
cvInfo(cv.models.meta, "selected model")
})
} else {
cat("install the 'carData' package to run these examples\n")
}
```

`cvCompute` *Utility Functions for the cv Package*

Description

These functions are primarily useful for writing methods for the `cv()` generic function. They are used internally in the package and can also be used for extensions (see the vignette "Extending the cv package, `vignette("cv-extend", package="cv")`).

Usage

```
cvCompute(
  model,
  data = insight::get_data(model),
  criterion = mse,
  criterion.name,
  k = 10L,
  reps = 1L,
  seed,
  details = k <= 10L,
  confint,
  level = 0.95,
  method = NULL,
  ncores = 1L,
  type = "response",
  start = FALSE,
  f,
  fPara = f,
  locals = list(),
  model.function = NULL,
  model.function.name = NULL,
  ...
)

cvMixed(
  model,
  package,
  data = insight::get_data(model),
  criterion = mse,
  criterion.name,
  k,
```

```
  reps = 1L,
  confint,
  level = 0.95,
  seed,
  details,
  ncores = 1L,
  clusterVariables,
  predict.clusters.args = list(object = model, newdata = data),
  predict.cases.args = list(object = model, newdata = data),
  fixed.effects,
  ...
)

cvSelect(
  procedure,
  data,
  criterion = mse,
  criterion.name,
  model,
  y.expression,
  k = 10L,
  confint = n >= 400,
  level = 0.95,
  reps = 1L,
  save.coef,
  details = k <= 10L,
  save.model = FALSE,
  seed,
  ncores = 1L,
  ...
)

folds(n, k)

fold(folds, i_, ...)

## S3 method for class 'folds'
fold(folds, i_, ...)

## S3 method for class 'folds'
print(x, ...)

GetResponse(model, ...)

## Default S3 method:
GetResponse(model, ...)

## S3 method for class 'merMod'
```

```
GetResponse(model, ...)

## S3 method for class 'lme'
GetResponse(model, ...)

## S3 method for class 'glmmTMB'
GetResponse(model, ...)

## S3 method for class 'modList'
GetResponse(model, ...)

checkFormula(model, data.names)
```

Arguments

`model`	a regression model object.
`data`	data frame to which the model was fit (not usually necessary, except for `cvSelect()`).
`criterion`	cross-validation criterion ("cost" or lack-of-fit) function of form `f(y, yhat)` where `y` is the observed values of the response and `yhat` the predicted values; the default is `mse` (the mean-squared error).
`criterion.name`	a character string giving the name of the CV criterion function in the returned `"cv"` object).
`k`	perform k-fold cross-validation (default is `10`); `k` may be a number or `"loo"` or `"n"` for n-fold (leave-one-out) cross-validation; for `folds()`, `k` must be a number.
`reps`	number of times to replicate k-fold CV (default is `1`).
`seed`	for R's random number generator; optional, if not supplied a random seed will be selected and saved; not needed for n-fold cross-validation.
`details`	if `TRUE` (the default if the number of folds `k <= 10`), save detailed information about the value of the CV criterion for the cases in each fold and the regression coefficients with that fold deleted.
`confint`	if `TRUE` (the default if the number of cases is 400 or greater), compute a confidence interval for the bias-corrected CV criterion, if the criterion is the average of casewise components.
`level`	confidence level (default `0.95`).
`method`	computational method to apply; use by some `cv()` methods.
`ncores`	number of cores to use for parallel computations (default is `1`, i.e., computations aren't done in parallel).
`type`	used by some `cv()` methods, such as the default method, where `type` is passed to the `type` argument of `predict()`; the default is `type="response"`, which is appropriate, e.g., for a `"glm"` model and may be recognized or ignored by `predict()` methods for other model classes.
`start`	used by some `cv()` methods; if `TRUE` (the default is `FALSE`), the `start` argument, set to the vector of regression coefficients for the model fit to the full data, is passed to `update()`, possibly making the CV updates faster, e.g. for a GLM.

`f`	function to be called by `cvCompute()` for each fold.
`fPara`	function to be called by `cvCompute()` for each fold using parallel computation.
`locals`	a named list of objects that are required in the local environment of `cvCompute()` for `f()` or `fPara()`.
`model.function`	a regression function, typically for a new `cv()` method, residing in a package that's not a declared dependency of the **cv** package, e.g., `nnet::multinom`.
`model.function.name`	the quoted name of the regression function, e.g., `"multinom"`.
`...`	to match generic; passed to `predict()` for the default method, and to `fPara()` (for parallel computations) in `cvCompute()`.
`package`	the name of the package in which mixed-modeling function (or functions) employed resides; used to get the namespace of the package.
`clusterVariables`	a character vector of names of the variables defining clusters for a mixed model with nested or crossed random effects; if missing, cross-validation is performed for individual cases rather than for clusters
`predict.clusters.args`	a list of arguments to be used to predict the whole data set from a mixed model when performing CV on clusters; the first two elements should be `model` and `newdata`; see the "Extending the cv package" vignette (`vignette("cv-extend", package="cv")`).
`predict.cases.args`	a list of arguments to be used to predict the whole data set from a mixed model when performing CV on cases; the first two elements should be `model` and `newdata`; see the "Extending the cv package" vignette (`vignette("cv-extend", package="cv")`).
`fixed.effects`	a function to be used to compute fixed-effect coefficients for cluster-based CV when `details = TRUE`.
`procedure`	a model-selection procedure function (see Details).
`y.expression`	normally the response variable is found from the `model` argument; but if, for a particular selection procedure, the `model` argument is absent, or if the response can't be inferred from the model, the response can be specified by an expression, such as `expression(log(income))`, to be evaluated within the data set provided by the `data` argument.
`save.coef`	save the coefficients from the selected models? Deprecated in favor of the `details` argument; if specified, `details` is set is set to the value of `save.coef`.
`save.model`	save the model that's selected using the *full* data set.
`n`	number of cases, for constructed folds.
`folds`	an object of class `"folds"`.
`i_`	a fold number for an object of class `"folds"`.
`x`	a `"cv"`, `"cvList"`, or `"folds"` object to be printed
`data.names`	names of variables in the data set to which the model was fit; if missing, an attempt will be made to extract the data from the model.

Value

The utility functions return various kinds of objects:

- `cvCompute()` returns an object of class `"cv"`, with the CV criterion (`"CV crit"`), the bias-adjusted CV criterion (`"adj CV crit"`), the criterion for the model applied to the full data (`"full crit"`), the confidence interval and level for the bias-adjusted CV criterion (`"confint"`), the number of folds (`"k"`), and the seed for R's random-number generator (`"seed"`). If `details=TRUE`, then the returned object will also include a `"details"` component, which is a list of two elements: `"criterion"`, containing the CV criterion computed for the cases in each fold; and `"coefficients"`, regression coefficients computed for the model with each fold deleted. Some `cv()` methods calling `cvCompute()` may return a subset of these components and may add additional information. If `reps` > `1`, then an object of class `"cvList"` is returned, which is literally a list of `"cv"` objects.
- `cvMixed()` also returns an object of class `"cv"` or `"cvList"`.
- `cvSelect` returns an object of class `"cvSelect"` inheriting from `"cv"`, or an object of class `"cvSelectList"` inheriting from `"cvList"`.
- `folds()` returns an object of class folds, for which there are `fold()` and `print()` methods.
- `GetResponse()` returns the (numeric) response variable from the model.

  The supplied `default` method returns the `model$y` component of the model object, or, if `model` is an S4 object, the result returned by the `get_response()` function in the **insight** package. If this result is `NULL`, the result of `model.response(model.frame(model))` is returned, checking in any case whether the result is a numeric vector.

  There are also `"lme"`, `"merMod"` and `"glmmTMB"` methods that convert factor responses to numeric 0/1 responses, as would be appropriate for a generalized linear mixed model with a binary response.
- `checkFormula()` returns `TRUE` if all variables in the model formula are also in the data to which the model is fit; `FALSE` is this is not the case (and q warning is printed); or `NA` if the function couldn't extract a model formula.

Functions

- `cvCompute()`: used internally by `cv()` methods (not for direct use); exported to support new `cv()` methods.
- `cvMixed()`: used internally by `cv()` methods for mixed-effect models (not for direct use); exported to support new `cv()` methods.
- `cvSelect()`: used internally by `cv()` methods for cross-validating a model-selection procedure; may also be called directly for this purpose, but use via `cv()` is preferred. `cvSelect()` is exported primarily to support new model-selection procedures.
- `folds()`: used internally by `cv()` methods (not for direct use).
- `fold()`: to extract a fold from a `"folds"` object.
- `fold(folds)`: `fold()` method for `"folds"` objects.
- `print(folds)`: `print()` method for `"folds"` objects.
- `GetResponse()`: function to return the response variable from a regression model.
- `GetResponse(default)`: default method.

- `GetResponse(merMod)`: `"merMod"` method.
- `GetResponse(lme)`: `"lme"` method.
- `GetResponse(glmmTMB)`: `"glmmTMB"` method.
- `GetResponse(modList)`: `"modList"` method.
- `checkFormula()`: check a model formula to determine whether it include variables not in the data to which the model was fit; prints a warning if this is not the case.

See Also

`cv`, `cv.merMod`, `cv.function`.

Examples

```
fit <- lm(mpg ~ gear, mtcars)
GetResponse(fit)

set.seed(123)
(ffs <- folds(n=22, k=5))
fold(ffs, 2)
```

`mse` *Cost Functions for Fitted Regression Models*

Description

Compute cost functions (cross-validation criteria) for fitted regression models.

Usage

```
mse(y, yhat)

rmse(y, yhat)

medAbsErr(y, yhat)

BayesRule(y, yhat)

BayesRule2(y, yhat)
```

Arguments

`y`	response
`yhat`	fitted value

Details

Cost functions (cross-validation criteria) are meant to measure lack-of-fit. Several cost functions are provided:

1. `mse()` returns the mean-squared error of prediction for a numeric response variable `y` and predictions `yhat`; and `rmse()` returns the root-mean-squared error and is just the square-root of `mse()`.
2. `medAbsErr()` returns the median absolute error of prediction for a numeric response `y` and predictions `yhat`.
3. `BayesRule()` and `BayesRule2()` report the proportion of incorrect predictions for a dichotomous response variable `y`, assumed coded (or coercible to) `0` and `1`. The `yhat` values are predicted probabilities and are rounded to 0 or 1. The distinction between `BayesRule()` and `BayesRule2()` is that the former checks that the `y` values are all either `0` or `1` and that the `yhat` values are all between 0 and 1, while the latter doesn't and is therefore faster.

Value

In general, cost functions should return a single numeric value measuring lack-of-fit. `mse()` returns the mean-squared error; `rmse()` returns the root-mean-squared error; `medAbsErr()` returns the median absolute error; and `BayesRule()` and `BayesRule2()` return the proportion of misclassified cases.

Functions

- `mse()`: Mean-square error.
- `rmse()`: Root-mean-square error.
- `medAbsErr()`: Median absolute error.
- `BayesRule()`: Bayes Rule for a binary response.
- `BayesRule2()`: Bayes rule for a binary response (without bounds checking).

See Also

`cv`, `cv.merMod`, `cv.function`.

Examples

```
if (requireNamespace("carData", quietly=TRUE)){
withAutoprint({
data("Duncan", package="carData")
m.lm <- lm(prestige ~ income + education, data=Duncan)
mse(Duncan$prestige, fitted(m.lm))

data("Mroz", package="carData")
m.glm <- glm(lfp ~ ., data=Mroz, family=binomial)
BayesRule(Mroz$lfp == "yes", fitted(m.glm))
})
} else {
cat("\n install 'carData' package to run these examples\n")
}
```

`Pigs` *Body Weights of 48 Pigs in 9 Successive Weeks*

Description

This data set appears in Table 3.1 of Diggle, Liang, and Zeger (1994).

Usage

```
data("Pigs", package = "cv")
```

Format

A data frame with 432 rows and 3 columns.

id Pig id number, 1–48.

week Week number, 1–9.

weight Weight in kg.

Source

P. J. Diggle, K.-Y. Liang, and S. L. Zeger, *Analysis of Longitudinal Data* (Oxford, 1994).

Examples

```
library("lme4")
m.p <- lmer(weight ~ week + (1 | id) + (1 | week),
            data=Pigs, REML=FALSE,
            control=lmerControl(optimizer="bobyqa"))
summary(m.p)
cv(m.p, clusterVariables=c("id", "week"), k=10, seed=8469)
```

Index