Package 'mosaicModel'

September 22, 2017

Type Package

Title An Interface to Statistical Modeling Independent of Model Architecture

Version 0.3.0

Author Kaplan Daniel [aut, cre],
Pruim Randall [aut, cre]

Maintainer Daniel Kaplan <kaplan@macalester.edu>

Description Provides functions for evaluating, displaying, and interpreting statistical models. The goal is to abstract the operations on models from the particular architecture of the model. For instance, calculating effect sizes rather than looking at coefficients. The package includes interfaces to both regression and classification architectures, including lm(), glm(), rlm() in 'MASS', random forests and recursive partitioning, k-nearest neighbors, linear and quadratic discriminant analysis, and models produced by the 'caret' package's train(). It's straightforward to add in other other model architectures.

License MIT + file LICENSE

Encoding UTF-8

LazyData TRUE

LazyLoad TRUE

RoxygenNote 6.0.1

Depends R (>= 3.1), mosaicCore, splines, dplyr

Imports caret, ggplot2, ggformula, lazyeval, knitr, MASS, testthat, tibble, tidyr, tidyverse

Suggests mosaic, mosaicData, randomForest, rpart

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2017-09-22 16:21:41 UTC

R topics documented:

AARP *Prices for life insurance*

Description

The AARP (original named the "American Association of Retired People" but now just AARP) offers life insurance to it's members. The data come from a full-page advertisement (circa 2012) in the "AARP Bulletin", which has the second largest circulation in the world of any magazine, with upward of 40 million subscribers. (Only the "AARP Magazine" has a larger circulation.)

Usage

```
data(AARP)
```

Format

A data frame with 36 observations on the following variables.

- `Age` The age of the person covered by the insurance policy.
- `Sex` The sex of the person covered by the insurance policy.
- `Coverage` The "death benefit" in 1000 USD.
- `Cost` Monthly cost in USD.

Details

Life insurance provides a "death benefit", money paid out to the insured person's survivors upon death of the insured. There is a cost for the insurance. Among other factors, the cost depends on both age and sex. (For this type of insurance, called "term insurance", the cost changes as the insured person ages.)

Source

The "AARP Bulletin". A copy of the ad is available at this link.

Examples

```
mod_1 <- lm(Cost ~ Age + Coverage, data = AARP)
mod_effect(mod_1, ~ Coverage)
```

`Alder` *Nitrogen fixing by alder plants*

Description

These data were collected by biologist Mike Anderson in a study of nitrogen fixation by bacteria growing on the root nodules of alder bushes.

Usage

```
data(Alder)
```

Format

A data frame `Alder` with 196 rows and 24 variables:

- LAND. landscape, floodplain vs. upland.
- SAMPPER. sampling period: early, mid, late
- SPECIES. host species, Alnus tenuifolia (AT) vs. A. crispa (AC)
- STAGE. successional stage, early vs. late in floodplain and upland landscapes. This is not equivalent across landscapes.
- JULDAY. Julian day
- PERNODN. nodule percent nitrogen by mass
- RF. bacterial genotype
- SNF. nitrogen fixation rate of nodule tissue, umol N2/gram of nodule dry weight/hr
- SLA. specific leaf weight, grams of leaf weight/square-meter, dry
- ONECM. soil temperature at 1 cm depth
- FIVECM. soil temperature at 5 cm depth
- PERH2O. soil moisture, percent H2O by mass
- DEL. del15N of leaf tissue
- DELNOD. del15N of nodule tissue
- NPERAREA. leaf nitrogen content per unit leaf area
- NDiff. nitrogen content difference between leaf and nodule of the same plant
- delDiff. del15N difference between leaf and nodule of the same plant
- SITE. Site designations: 1A,B,C for replicate early succession floodplain sites, 4A,B,C for late succession floodplain, UP1A,B,C for early succession upland and UP3A,B,C for late succession upland
- HABSPEC. habitat+species, concatenated LAND, STAGE, SPECIES
- SITESPEC. concatenated SITE, SPECIES
- REP. replicate site within a given level of HABSPEC
- PLNO. plant number, unique for individuals of each species (AT1-180, AC1-270)

Details

Two questions that Anderson wanted to answer are: (1) Can any variation in nitrogen fixation (variable `SNF`) be attributed to genotype (variable `RF`)? (2) What are the major sources of variation in `SNF` and `PERLEAFN`? Variables of biological interest are seasonality (`SAMPPER` or `JULDAY`), soil temperature and moisture, and habitat differences (`STAGE` for host species AT and`STAGE` and `LAND` for host species AC).

Three replicate sites were sampled for each landscape/stage combination in three sampling periods across the growing season. Site sampling was arranged in a Latin Square design in order to systematize any effects of seasonality on N2-fixation rates.

Source

Michael Anderson

References

Anderson MD, Ruess RW, Myrold DD, Taylor DL. “Host species and habitat affect modulation by specific Frankia genotypes in interior Alaska” Oecologia (2009) 160:619-630.

Examples

```
mod <- lm(logSNF ~ RF + SITESPEC, data = Alder)
```

`Birth_weight` *Birth weights and maternal data*

Description

Birth weight, date, and gestational period collected as part of the Child Health and Development Studies in 1961 and 1962. Information about the baby’s parents — age, education, height, weight, and whether the mother smoked is also recorded. The data were present by Nolan and Speed to address the question of whether there is a link between maternal smoking and the baby’s health.

Usage

```
data(Birth_weight)
```

Format

A data frame with 886 observations on the following variables.

- `baby_wt` Birth weight of baby, in ounces.
- `mother_wt` in pounds
- `gestation` Length of the pregnancy, in days.
- `smoker` Whether the mother smoked during the pregnancy.
- `income`Family yearly income in 2500USD increments 0 = under 2500, 1=2500-4999, ..., 8= 12,500-14,999, 9=15000+

Source

D. Nolan and T.P. Speed (2009) "Stat Labs: Mathematical Statistics Through Applications"

Examples

```
mod_1 <- lm(baby_wt ~ gestation + mother_wt, data = Birth_weight)
mod_effect(mod_1, ~ gestation)
```

ci.proportion *Function builder for confidence intervals on proportions*

Description

Similar to proportion, but

Usage

```
ci.proportion(nm = NULL, level = 0.95)
```

Arguments

nm	The level for which to find the proportion
level	The confidence interval (Default: 0.95)

Examples

```
## Not run:
df_stats(mtcars, ~ cyl, cyl_prop = ci.proportion(6, level = 0.90))

## End(Not run)
```

College_grades *Grades at a small college*

Description

These are the actual grades for 400+ individual students in the courses they took at a small, liberal-arts college in the midwest US. All the students graduated in 2006. Each row corresponds to a single student in a single course. The data have been de-identified by translating the student ID, the instructor ID, and the name of the department. Typically a graduating student has taken about 32 courses. As another form of de-identification, only half of the courses each student, selected randomly, are included. Only courses with 10 or more students enrolled were included.

Usage

```
data(College_grades)
```

Format

A data frame with 6146 Grades for 443 students.

- `grade` The letter grade for the student in this course: A is the highest.
- `sessionID` An identifier for the course taken. Courses offered multiple times in one semester or across semesters have individual IDs.
- `sid` The student ID
- `dept` The department in which the course was offered. 100 is entry-level, 200 sophomore-level, 300 junior-level, 400 senior-level.
- `enroll` Student enrollment in the course. This includes students who are not part of this sample.
- `iid` Instructor ID
- `gradepoint` A translation of the letter grade into a numerical scale. 4 is high. Some letter grades are not counted in a student's gradepoint average. These have `NA` for the gradepoint.

Source

The data were helpfully provided by the registrar of the college with the proviso that the de-identification steps outlined above be performed.

Examples

```
## Not run:
GPA <- lm(gradepoint ~ sid - 1, data = College_grades)

## End(Not run)
```

`collinearity` *Calculate measures of collinearity*

Description

Calculate measures of collinearity

Usage

```
collinearity(formula, data, format = c("SeIF", "degrees", "radians", "VIF"))
```

Arguments

`formula` a formula giving, on the right-hand side, the explanatory variables to be considered. Iteractions, etc. may also be specified.

`data` a data frame from which to draw the variables in the formula

`format` choice of `"SeIF"` for inflation of standard errors, `"degrees"` or `"radians"` for collinearity described as an angle or `"VIF"` for the variance inflation factor (which is the square of SeIF).

Examples

```
collinearity( ~ cyl * disp * hp, data = mtcars)
collinearity( ~ cyl * disp * hp, data = mtcars, format = "degrees")
```

---

construct_fitting_call *Construct a call for refitting a model from the model itself*

Description

This will typically *not* be used by the end-user.

Usage

```
construct_fitting_call(model, data_name = "training", ...)
```

Arguments

`model`	the model in question
`data_name`	character string specifying the name of the data frame used for the refitting. This object *must* be defined in the environment in which the call is being made.
`...`	(not used)

Details

This provides a way to refit a model on either resampled or sub-sampled data. Not all model architectures support this. If not, then you can't use `mod_ensemble` or `mod_cv`, or use the `bootstrap=` argument in any of the other functions.

---

coverage *Interval statistics for use with df_stats()*

Description

Function builders for calculating intervals. These must *always* be evaluated with the *result* being handed as a argument to `df_stats()`.

Usage

```
coverage(level = 0.95)
```

Arguments

`level`	Number in 0 to 1 specifying the confidence level for the interval. (Default: 0.95)

Examples

```
cover <- coverage(0.95)
df_stats(hp ~ cyl, data = mtcars, c95 = cover)
```

Crime *Data from the US FBI Uniform Crime Report, 1960*

Description

A report of the number of offenses reported to police per million population, and many other social and demographic variables. Each case corresponds to a state in the US.

Usage

```
data(Crime)
```

Format

A data frame with 47 cases, each of which is a US state, with observations on the following variables.

- `R` Crime rate: number of offenses reported to police per million population.
- `Age` Number of males aged 14-24 per 1000 population
- `N` State population (in 100,000s)
- `W` State-wise median value of transferable goods and assets or family income in tens of dollars.
- `X` Number of families per 1000 earning below half the median income.
- `ExDiff` Change in per capita expenditure on police by state and local government from 1950 to 1960
- `Ex0` 1960 per capita expenditures on police.

Source

FBI Uniform Crime Report via DASL: Data and Story Library

Examples

```
mod_1 <- lm(R ~ W, data = Crime)
mod_2 <- lm(R ~ X, data = Crime)
mod_3 <- lm(R ~ W + X, data = Crime)
mod_effect(mod_1, ~ W)
mod_effect(mod_3, ~ W)
mod_effect(mod_2, ~ X)
mod_effect(mod_3, ~ X)
```

`data_from_mod` *Extract training data from model*

Description

This typically will *not* be used by an end-user.

Usage

```
data_from_mod(model, ...)
```

Arguments

`model` the model from which to extract the training data

`...` additional arguments (not used)

Details

not all model architectures keep track of the training data If a model architecture isn't recognized, you'll have to add a method for that class. See vignette.

`df_counts` *Formula interface to counts*

Description

Counts the number of cases in a data frame broken down by the variables in the formula.

Usage

```
df_counts(formula, data, wide = FALSE, margins = FALSE)
```

Arguments

`formula` the formula describing the relationship

`data` a data frame (or you can pipe this in)

`wide` reformat the output as a cross-tabulation. This makes sense only when there are just two variables

`margins` show the marginal counts. Makes the most sense if `wide = TRUE`.

See Also

`df_props`

df_props *Joint and conditional proportions*

Description

Uses a formula interface to specify how the proportions are to be calculated.

Usage

```
df_props(formula, data, as.percent = FALSE, ..., wide = FALSE,
  margins = FALSE, format = c("proportion", "percent", "count"))
```

Arguments

formula	the formula describing the relationship
data	a data frame (or you can pipe this in)
as.percent	show proportions in percent (e.g. multiply by 100)
...	statistics functions to be applied to the data, e.g. mean, sd, confidence(0.95)
wide	reformat the output as a cross-tabulation. This makes sense only when there are just two variables
margins	show the marginal probabilities. Makes the most sense if `wide = TRUE`.
format	Use just for internal purposes.

Details

Using `|` in the formula specifies a conditional proportion

- ~ A : proportion of cases in each level of A
- ~ A + B: joint proportion: each level of A crossed with B
- ~ A | B: conditional proportion: for each level of B, what fraction are in each level of A
- A ~ B: another way of specifying the conditional proportion

Examples

```
df_props(mtcars, ~ cyl + gear)
df_props(mtcars, ~ cyl | gear)
df_props(mtcars, ~ cyl + gear, wide = TRUE)
df_props(mtcars, ~ cyl + gear, margins = TRUE)
df_props(mtcars, ~ cyl | gear, margins = TRUE)
```

df_typical *Find typical levels of explanatory variables in a model/dataset.*

Description

This function tries to choose sensible values of the explanatory variables from the data used to build a model or any other specified data. (or from data specified with the `data =` argument.)

Usage

```
df_typical(data = NULL, nlevels = 3, at = list(), model = NULL, ...)
```

Arguments

`data`	optional data frame from which to extract levels for explanatory variables
`nlevels`	how many levels to construct for input variables. For quantitative variables, this is a suggestion. Set to `Inf` to get all levels for categorical variables and 100 levels for quantitative variables.
`at`	named list giving specific values at which to hold the variables. Use this to override the automatic generation of levels for any or all explanatory variables.
`model`	the model to display graphically
`...`	a more concise mechanism to passing desired values for variables

Details

For categorical variables, the most populated levels are used. For quantitative variables, a sequence of `pretty()` values is generated.

For categorical variables, will return the nlevels most popular levels, unless the levels are specified explicitly in an argument.

Value

A dataframe containing all combinations of the selected values for the explanatory variables. If there are p explanatory variables, there will be about `nlevels^p` cases.

Examples

```
## Not run:
df_typical(mosaicData::Galton, nlevels = 2, father = 70, mother = 68, nkids = 3)
df_typical(mosaicData::Galton, nlevels = 2)
mod1 <- lm(wage ~ age * sex + sector, data = mosaicData::CPS85)
df_typical(model = mod1, nlevels = 3)

## End(Not run)
```

explanatory_vars *Get the names of the explanatory or response variables in a model*

Description

This will typically *not* be used by the end_user. These functions let you interrogate any model architecture that's covered by mosaicModel about the response and explanatory variables. These are used internally in functions such as `mod_plot`.

Usage

```
explanatory_vars(model, ...)
```

Arguments

`model`	the model in question
`...`	(not used)

formula_from_mod *Extract the model formula used in specifying the model*

Description

This typically will *not* be used by an end-user.

Usage

```
formula_from_mod(model, ...)
```

Arguments

`model`	the model
`...`	(not used)

Details

Not all model architectures support this. If a model architecture isn't recognized, you'll have to add a method for that class. See vignette.

HDD_Minneapolis *Heating degree days in Minneapolis, Minnesota, USA*

Description

A "heating degree day" is a measure of weather coldness. It's defined to be the difference between the outdoor ambient temperature and 65 degrees F, but has a value of zero when the ambient temperature is above 65 degrees. This difference is averaged over time and multiplied by the number of days in the time period covered. The heating degree day is often used as a measure of the demand for domestic heating in a locale.

Usage

```
data(HDD_Minneapolis)
```

Format

A data frame `HDD_Minneapolis` with 1412 rows and 4 variables:

- year the year
- month the month
- hdd the number of heating degree days for that period.
- loc the location at which the temperature was measured. In the early years, this was downtown Minneapolis. Later, the site was moved to the Minneapolis/Saint-Paul International Airport.

Details

These data report monthly heating degree days. For teaching purposes, the data give an extreme example of how a relationship (hdd vs year) can be revealed by including a covariate (month). Although interest focusses on the change in temperature over the century the data cover, there is such regular seasonal variation that no systematic trend over the years is evident unless month is taken into account.

Examples

```
mod_1 <- lm(hdd ~ year, data = HDD_Minneapolis)
mod_2 <- lm(hdd ~ year + month, data = HDD_Minneapolis)
```

`Houses_for_sale` *Houses for sale*

Description

A random sample of 1,728 homes taken from public records from the Saratoga County (http://www.saratogacountyny.gov/dep property-tax-service-agency/). Collected by Candice Corvetti (Williams College '07) for her senior thesis.

Usage

```
data(Houses_for_sale)
```

Format

A dataframe with 1728 cases, each of which is a house for sale.

Details

These data are part of a case study developed by Prof. Dick de Veaux at Williams. They are available from the American Statistical Association's Stat 101 collection of case studies and included in this package for convenience.

References

Dick De Veaux (2015) "How much is a fireplace worth?" Stats 101: A resource for teaching introductory statistics, American Statistical Association

Examples

```
mod_1 <- lm(price ~ fireplaces, data = Houses_for_sale)
mod_2 <- lm(price ~ fireplaces + living_area, data = Houses_for_sale)
mod_effect(mod_1, ~ fireplaces)
mod_effect(mod_2, ~ fireplaces)
mod_plot(mod_2, ~ living_area + fireplaces)
```

`mod_cv` *Compare models with k-fold cross validation*

Description

Compare models with k-fold cross validation

Usage

```
mod_cv(..., k = 10, ntrials = 5, error_type = c("default", "mse", "sse",
  "mad", "LL", "mLL", "dev", "class_error"))
```

Arguments

... one or more models on which to perform the cross validation

k the k in k-fold. cross-validation will use k-1/k of the data for training.

ntrials how many random partitions to make. Each partition will be one case in the output of the function

error_type The kind of output to produce from each cross-validation. See mod_error for details.

Details

The purpose of cross-validation is to provide "new" data on which to test a model's performance. In k-fold cross-validation, the data set used to train the model is broken into new training and testing data. This is accomplished simply by using most of the data for training while reserving the remaining data for evaluating the model: testing. Rather than training a single model, k models are trained, each with its own particular testing set. The testing sets in the k models are arranged to cover the whole of the data set. On each of the k testing sets, a performance output is calculated. Which output is most appropriate depends on the kind of model: regression model or classifier. The most basic measure is the mean square error: the difference between the actual response variable in the testing data and the output of the model when presented with inputs from the testing data. This is appropriate in many regression models.

mod_effect *Calculate effect sizes in a model*

Description

Like a derivative or finite-difference

Usage

```
mod_effect(model, formula, step = NULL, bootstrap = 0, to = step,
  nlevels = 1, data = NULL, at = NULL, class_level = NULL, ...)
```

Arguments

model the model from which the effect size is to be calculated

formula a formula whose right-hand side is the variable with respect to which the effect size is to be calculated.

step the numerical stepsize for the change var, or a comparison category for a categorical change var. This will be either a character string or a number, depending on the type of variable specified in the formula.

bootstrap The number of bootstrap replications to construct. If greater than 1, calculate a standard error using that number of replications.

to a synonym for step. (In English, "to" is more appropriate for a categorical input, "step" for a quantitative. But you can use either.)

`nlevels`	integer specifying the number of levels to use for "typical" inputs. (Default: up to 3)
`data`	Specifies exactly the cases at which you want to calculate the effect size.
`at`	similar to `...` but expects a list or dataframe of the values you want to set. Like `...`, all combinations of the values specified will be used as inputs.
`class_level`	Name of the categorical level for which the probability is to be used. Applies only to classifiers. (Default: Use the first level.) Unlike `...` or `at`, no new combinations will be created.
`...`	additional arguments for evaluation levels of explanatory variables.

Details

When you want to force or restrict the effect size calculation to specific values for explanatory variables, list those variables and levels as a vector in ... For example, `educ = c(10, 12, 16)` will cause the effect size to be calculated at each of those three levels of education. Any variables whose levels are not specified in ... will have values selected automatically.

Value

a data frame giving the effect size and the values of the explanatory variables at which the effect size was calculated. There will also be a column `to_` showing the value jumped to for the variable with respect to which the effect size is calculated. When `bootstrap` is greater than 1, there will be a standard error reported on the effect size; see the variable ending in `_se`.

Examples

```
mod1 <- lm(wage ~ age * sex * educ + sector, data = mosaicData::CPS85)
mod_effect(mod1, ~ sex)
mod_effect(mod1, ~ sector)
mod_effect(mod1, ~ age, sex = "M", educ = c(10, 12, 16), age = c(30, 40))
mod_effect(mod1, ~ age, sex = "F", age = 34, step = 1)
mod_effect(mod1, ~ sex, age = 35, sex = "M", to = "F" )
# For classifiers, the change in *probability* of a level is reported.
mod2 <- rpart::rpart(sector ~ age + sex + educ + wage, data = mosaicData::CPS85)
mod_effect(mod2, ~ educ)
mod_effect(mod2, ~ educ, class_level = "manag")
```

`mod_ensemble` *Create bootstrapped ensembles of a model*

Description

Create bootstrapped ensembles of a model

Usage

```
mod_ensemble(model, nreps = 2, data = NULL)
```

Arguments

`model` a model whose data will be used for resampling

`nreps` how many resampling trials should be created

`data` a data table to use for resampling. This is not needed for many common model types, such as `lm`, `glm`, etc. See details.

Details

The approach to bootstrapping implemented by this function is to create a set of bootstrap trials all in one go. Then, other functions such as `mod_effect()` and `mod_eval()` will be used to extract the information from each of the bootstrap replicates. Many model types in R carry the data used to train the model as part of the model object produced. For these types of models, e.g. `lm` and `glm`, there is no need to provide a value for the `data` argument. But there are some types of models for which the training data cannot be extracted from the model object. In such situations, you use `data =` to provide the data set to use for resampling.

`mod_error` *Mean square prediction error*

Description

Compares model predictions to the actual value of the response variable. To do this, testing data must be provided with *both* the input variables and the corresponding response variable. The measure calculated for a quantitative response variable is the mean square prediction error (MSPE). For categorical response variables, an analog of MSPE can be calculated (see details) but by default, a mean log-likelihood (mean per case) is computed instead.

Usage

```
mod_error(model, testdata, error_type = c("default", "mse", "sse", "mad",
  "LL", "mLL", "dev", "class_error"))
```

Arguments

`model` The model whose prediction error is to be estimated.

`testdata` A data frame giving both model inputs and the actual value of the response variable. If no testing data is provided, the training data will be used and a warning issued.

`error_type` The measure of error you are interested in. By default, this is mean-square error for regression models and log-likelihood for classifiers. The choices are:

- `"mse"` – mean square error
- `"sse"` – sum of square errors
- `"mad"` – mean absolute deviation
- `"LL"` – log-likelihood
- `"mLL"` – mean log-likehood (per case in the testing data)

- "dev" – deviance. (Plus a constant, which is often zero. The constant is fixed for a given testing data set, regardless of the model. So differences between deviances of two models are correct.)
- "class_error" – classification error rate.

Details

When the response variable is categorical, the model (called a 'classifier' in such situations) must be capable of computing *probabilities* for each output rather than just a bare category. This is true for many commonly encountered classifier model architectures.

The analog of the mean squared error for classifiers is the mean of (1-p)^2, where p is the probability assigned by the model to the actual output. This is a rough approximation to the log-likelihood. By default, the log-likelihood will be calculated, but for pedagogical reasons you may prefer (1-p)^2, in which case set `error_type = "mse"`. Classifiers can assign a probability of zero to the actual output, in which case the log-likelihood is `-Inf`. The `"mse"` error type avoids this.

Examples

```
mod <- lm(mpg ~ hp + wt, data = mtcars)
mod_error(mod) # In-sample prediction error.
## Not run:
classifier <- rpart::rpart(Species ~ ., data = iris)
mod_error(classifier)
mod_error(classifier, error_type = "LL")
# More typically
inds <- sample(1:nrow(iris), size = 100)
Training <- iris[inds, ]
Testing  <- iris[ - inds, ]
classifier <- rpart::rpart(Species ~ ., data = Training)
# This may well assign zero probability to events that appeared in the
# Testing data
mod_error(classifier, testdata = Testing)
mod_error(classifier, testdata = Testing, error_type = "mse")

## End(Not run)
```

mod_eval *Evaluate a model for specified inputs*

Description

Find the model outputs for specified inputs. This is equivalent to the generic `predict()` function, except it will choose sensible values by default. This simplifies getting a quick look at model values.

Usage

```
mod_eval(model = NULL, data = NULL, append = TRUE, interval = c("none",
  "prediction", "confidence"), nlevels = 2, bootstrap = 0, ...,
  on_training = FALSE)
```

Arguments

model the model to display graphically

data optional set of cases from which to extract levels for explanatory variables

append flag whether to include the inputs to the model along with the calculated model value in the output. Default: TRUE.

interval the type of interval to use: "none", "confidence", "prediction". But not all types are available for all model architectures.

nlevels how many levels to construct for input variables. (default: 3) For quantitative variables, this is a suggestion; an attempt is made to have the levels equally spaced. If you're dissatisfied with the result, use the ... to specify exactly what levels you want for any variable.

bootstrap if > 1, the number of bootstrap trials to run to construct a standard error on the model output for each value of the inputs. This is an alternative to interval; you can't use both.

... arguments about or values at which to evaluate the model or the kind of output to be passed along to predict(). Unlike data = the variables given in at = or ... will be crossed, so that the evaluation will occur at all combinations of the various levels.

on_training flag whether to use the training data for evaluation. Only needed when there are random terms, e.g. from rand(), shuffle(), ... See details.

Details

There are four distinct ways to specify the values at which the model is to be evaluated. (1) Look for some "typical values" in the data to create a handful of inputs. This is useful for getting a quick look at what the output of the model looks like. This is the default behavior. (2) Using data = to a dataframe containing the explanatory variables will evaluate the model at all of the cases contained in that dataframe. (3) Setting input variables explicitly by using arguments of the form var_name = values, e.g. sex = "F". If not all input variables are specified in this way, the ones that are not will have values set per (1). All combinations of the various variables will be created. See the nlevels argument. (4) Evaluating the model on the training data. There are two ways to do this. The first is to set the data argument to the same data frame used to train the model. The second is to use the on_training = TRUE argument. These are equivalent unless there is some random component among the explanatory terms, as with mosaic::rand(), mosaic::shuffle() and so on.

Value

A dataframe containing both the explanatory variable inputs and the resulting model output (in the model_value field). This differs from the output of predict(), which for many model classes/architectures may be a vector or matrix.

A data frame containing both the inputs to the model and the corresponding outputs.

Examples

```
## Not run:
mod1 <- lm(wage ~ age * sex + sector, data = mosaicData::CPS85)
```

```
mod_eval(mod1)
mod2 <- glm(married == "Married" ~ age + sex * sector,
            data = mosaicData::CPS85, family = "binomial")
mod_eval(mod2, nlevels = 2)
mod_eval(mod2, nlevels = 2, sex = "F")

## End(Not run)
```

mod_eval_fun *Internal functions for evaluating models*

Description

These functions are the interface to the various model types for mod_eval(), and through that to all the other mod_ functions that need to evaluate models, e.g. mod_effect(), mod_cv(), and so on.

Usage

```
mod_eval_fun(model, data = NULL, interval = "none", ...)
```

Arguments

model	A model object of the classes permitted
data	Usually, a data table specifying the inputs to the model. But if not specified, the training data will be used.
interval	One of "none", "confidence", or "prediction". Not all model types support "prediction" or even "confidence".
...	additional arguments

Details

All of the eval_ functions are ex These functions return a numerical vector (for regression types) or a matrix of probabilities (for classifiers)

mod_fun *Transforms a model into a function of inputs -> output*

Description

Implicit in many statistical models is a function that takes the explanatory variables as inputs and returns the corresponding model value at those inputs. mod_fun creates an R function that works this way. The function returned by ‘mod_fun“ has arguments named for each of the explanatory variables. In calling that returned function, you can specify as many or as few of these as you like.

Usage

```
mod_fun(mod, nlevels = 1)
```

Arguments

mod	the model to be rendered in a functional form
nlevels	the number of levels for which to find "typical levels" for those arguments not specified in the call to the returned function

Details

When you evaluate the function, you can set the values of all, any, or none of the arguments. Any arguments that you do not set will automatically be set to "typical values" as in `mod_eval`.

There's nothing essential about the behavior of 'mod_eval" that explicitly names the arguments to the model function with the names of the explanatory variables. This has been done purely for pedagogical reasons, as a reminder of what those variables are and to make it possible to spot mistaken inputs to models.

Value

a function whose arguments are the explanatory variable used in the model

Examples

```
my_mod <- lm(mpg ~ hp * cyl, data = mtcars)
f <- mod_fun(my_mod)
names(formals(f)) # the arguments will be the explanatory variables
f(hp = 1:2)
f(hp = 1:2, cyl = 3:4)
f() # typical values for inputs
```

---

mod_plot *Plot out model values*

Description

Plot out model values

Usage

```
mod_plot(model = NULL, formula = NULL, data = NULL, bootstrap = 0,
  nlevels = 3, at = list(), class_level = NULL, interval = c("none",
  "confidence", "prediction"), post_transform = NULL, size = 1,
  alpha = 0.8, ...)
```

Arguments

`model` the model to display graphically. Can also be an ensemble produced with `mod_ensemble()`

`formula` setting the y ~ x + color variables

`data` optional data set from which to extract levels for explanatory variables

`bootstrap` when > 1, this will generate bootstrap replications of the model and plot all of them. Use as an alternative to `interval` for confidence intervals.

`nlevels` how many levels to display for those variables shown at discrete levels

`at` named list giving specific values at which to hold the variables. You can accomplish this without forming a list by using `...`. See examples.

`class_level` character string. If a probability for a classifier is being shown, which levels of the response variable to use in the plot. (Default: the first one.)

`interval` show confidence or prediction intervals: values "none", "confidence", "prediction"

`post_transform` a scalar transformation and new name for the response variable, e.g. `post_transform = c(price = exp)` to undo a log transformation of price.

`size` numerical value for line width (default: 1)

`alpha` numerical value in 0 to 1 for transparency (default: 0.8)

`...` specific values for explantory variables

Examples

```
## Not run:
mod1 <- lm(wage ~ age * sex + sector, data = mosaicData::CPS85)
mod_plot(mod1)
mod_plot(mod1, n = Inf, interval = "confidence")
mod_plot(mod1, ~ sector + sex + age) # not necessarily a good ordering
mod_plot(mod1, ~ age + sex + sector, nlevels = 8)
mod2 <- lm(log(wage) ~ age + sex + sector, data = mosaicData::CPS85)
mod_plot(mod2, post_transform = c(wage = exp),
     interval = "confidence") # undo the log in the display
mod3 <- glm(married == "Married" ~ age + sex * sector,
            data = mosaicData::CPS85, family = "binomial")
mod_plot(mod3)
E3 <- mod_ensemble(mod3, 10)
mod_plot(E3)
mod4 <- rpart::rpart(sector ~ age + sex + married, data = mosaicData::CPS85)
mod_plot(mod4)
mod_plot(mod4, class_level = "manag")
mod5 <- randomForest::randomForest(
         sector ~ age + sex + married, data = mosaicData::CPS85)
mod_plot(mod5)
mod_plot(mod5, class_level = "manag")

## End(Not run)
```

mosaicModel — mosaicModel *package*

Description

Functions for teaching about modeling.

Details

The package offers a handful of high-level functions for evaluating, displaying, and interpreting models that work in a consistent way across model architectures, e.g. lm, glm, rpart, randomForest, knn3, caret-train, and so on.

- `mod_eval()` – evaluate a model, that is, turn inputs into model values. For many model architectures, you can also get prediction or confidence intervals on the outputs.
- `mod_plot()` – produce a graphical display of the "shape" of a model. There can be as many as 4 input variables shown, along with the output.
- `mod_effect()` – calculate effect sizes, that is, how a change in an input variable changes the output
- `mod_error()` – find the mean square prediction error (or the log likelihood)
- `mod_ensemble()` – create an ensemble of bootstrap replications of the model, that is, models fit to resampled data from the original model.
- `mod_cv()` – carry out cross validation on one or more models.
- `mod_fun()` – extract a function from a model that implements the inputs-to-output relationship. `mosaicModel` stays out of the business of training models. You do that using functions, e.g.

- the familiar `lm` or `glm` provided by the `stats` package
- `train` from the `caret` package for machine learning
- `rpart`, `randomForest`, `rlm`, and other functions provided by other packages

Mussels — *Metabolism of zebra mussels*

Description

Zebra mussels are a small, fast reproducing species of freshwater mussel native to the lakes of southeast Russia. They have accidentally been introduced in other areas, competing with native species and creating problems for people as they cover the undersides of docks and boats, clog water intakes and other underwater structures. Zebra mussels even attach themselves to other mussels, sometimes starving those mussels.

Usage

```
data(Mussels)
```

Format

A data frame `Mussels` with 30 rows and 11 variables.

- `GroupID` ID for the cluster of mussels growing on a substrate.
- `dry.mass` The mass of the mussels (as a group) after dehydration.
- `count` How many mussels were in the cluster.
- `attachment` The substrate to which the mussels were attached.
- `lipid` Percentage of dry mass that is lipid.
- `protein` Percentage of dry mass that is protein.
- `carbo` Percentage of dry mass that is carbohydrate.
- `ash` Percentage of dry mass that is ash.
- `ammonia` Nitrogen excretion measured as ammonia in mg per hour for the group.
- `Kcal` Total calorific value of the tissue in kilo-calories per gram.
- `O2` Oxygen uptake in mg per hour for the group.

Details

Ecologists Shirley Baker and Daniel Hornbach examined whether zebra mussels gain an advantage by attaching to other mussels rather than to rocks.(baker-hornbach-2008) The ecologists collected samples of small rocks and Amblema plicata mussels, each of which had a collection of zebra mussels attached. The samples were transported to a laboratory where the group of mussels from each individual rock or Amblema were removed and placed in an aquarium equipped to measure oxygen uptake and ammonia excretion. After these physiological measurements were made, the biochemical composition of the mussel tissue was determined: the percentage of protein, lipid, carbohydrate, and ash. Baker and Hornbach found that zebra mussels attached to Amblema had greater physiological activity than those attached to rocks as measured by oxygen uptake and ammonia excretion. But this appears to be a sign of extra effort for the Amblema-attached zebra mussels, since they had lower carbohydrate and lipid levels. In other words, attaching to Amblema appears to be disadvantageous to the zebra mussels compared to attaching to a rock.

Examples

```
Mussels$ind.mass <- with(Mussels, dry.mass/count)
mod_1 <- lm(O2/count ~ attachment, data = Mussels)
mod_2 <- lm(ammonia/count ~ attachment, data = Mussels)
mod_3 <- lm(O2/count ~ ind.mass + attachment, data = Mussels)
mod_4 <- lm(ammonia/count ~ ind.mass + attachment, data = Mussels)
```

NCI60_snippet *Gene expression in cancer cells.*

Description

The data come from a National Cancer Institute study of gene expression in cell lines drawn from various sorts of cancer. Each row corresponds to a different cell line. The type of cancer is identified by the first two or three letters of the row names, e.g. CO is colon, ME is melanoma, RE is kidney.

Usage

```
data(NCI60_snippet)
```

Format

A data frame `NCI60_snippet` with 60 rows and 6000 variables.

Details

For each row, there are 6000 measurements of the gene expression as identified by activity on a microarray probe. The variable names are the names of the probes.

Examples

```
data(NCI60_snippet)
```

Oil_history *Historical production of crude oil, worldwide 1880-2014*

Description

Annual production of crude oil, in millions of barrels (mbbl).

Usage

```
data(Oil_history)
```

Format

A data frame with 47 cases, each of which is a US state, with observations on the following variables.

- `year` the year for which production is reported
- `mbbl` oil production in millions of barrels

Source

Assembled from older information from RH Romer (1976) "Energy: An Introduction to Physics" and more recent data from `data.oecd.org`.

Examples

```
model <- lm(log(mbbl) ~ year, data = Oil_history)
mod_plot(model)
```

`proportion` *Function builder for proportions.*

Description

Evaluate this and hand the result to `df_stats()`

Usage

```
proportion(nm = NULL)
```

Arguments

`nm`	The level for which to find the proportion

Examples

```
## Not run:
df_stats(mtcars, ~ cyl, proportion(6))

## End(Not run)
```

`reference_values` *Compute sensible values from a data set for use as a baseline*

Description

Compute sensible values from a data set for use as a baseline

Usage

```
reference_values(data, n = 1, at = list())
```

Arguments

`data`	a data frame
`n`	number of values for specified variables: could be a single number or a list assigning a number of levels for individual variables
`at`	optional values at which to set values: a list whose names are the variables whose values are to be set.

Details

Variables not listed in `at` will be assigned levels using these principles: Categorical variables: the most populated levels. Quantitative variables: central quantiles, e.g. median for n=1,

`Runners` *Performance of runners in a ten-mile race as they age*

Description

These data are assembled from the records of the "Cherry Blossom Ten Miler" held in Washington, DC each April. The records span the years 1999 to 2008.

Usage

```
data(Runners)
```

Format

A data frame `Runners` with 24,334 rows and 8 variables.

- `age` The runners age at the time of the race.
- `net` The time (in min.) elapsed from when the runner crossed the start line until the finish.
- `gun` The time (in min.) from when the starter's gun was fired to when the runner finished the race. With so many participants, some runners do not reach the start line until several minutes after the gun.
- `sex` The runner's sex.
- `previous` How many times the runner participated before this year's race.
- `nruns` How many of the 10 years' runs the runner eventually participated in.
- `start_position` A made-up categorical description of where the runner was situated in the line up for the race..

Details

There are about 10,000 participants each year, of whom roughly half have run the race previously. During the ten years of these records, some 25 runners ran in ech of the years, 73 ran in nine of the years, and so on. The data allow you to track the performace of runners as they age. This simplified version of the data does not include personal identifying data other than sex, age, and number of previous runs in any given year. (Runs before 1999 are not considered.)

Examples

```
mod_1 = lm(net ~ age + sex, data = Runners)
```

School_data *Simulated data bearing on school vouchers*

Description

In the US, there have been controversial proposals to provide vouchers to students in failing public schools. The vouchers would allow the students to attend private schools. There are arguments pro and con that are often rooted in political philosophy (free choice!) and politics. The presumption behind the *pro* arguments is that attending private schools would create better outcomes for students.

Usage

```
data(School_data)
```

Format

A data frame with 500 cases, each of which is a simulated student, with observations on the following variables.

- `test_score` a simulated test score for the student
- `school` whether the student attended public or private school
- `lottery` whether the student was entered into a lottery for a private-school voucher
- `group` the racial/ethnic group of the student
- `acad_motivation` the overall level of involvement and concern of the student's parents for the student's academic performance
- `relig_motivation` the overall level of interest motivated by religion. This is potentially an issue because a large majority of urban private schools are Catholic.

Details

A reasonable way to test this presumption is to compare test scores for students in public and private schools. One famous analysis (Howell and Peterson, 2003, "The Education Gap: Vouchers and Urban Schools") found that voucher schools are most helpful for African-American students, and not so much for white or Hispanic students.

The `School_data` data frame comes from a simulation designed by the package author to replicate the overall results but supporting a very different policy recommendation. WARNING: This is just a simulation, reflecting one hypothesis about how the world might work. Don't be tempted to draw conclusions about the actual factors involved in school performance from such simulated data.

Examples

```
lm(test_score ~ school, data = School_data)
# the simulation mechanism itself:
nstudents <- 500
acad_motivation <- rnorm(nstudents)
group <- sample(c("black", "hispanic", "white"), replace = TRUE, size = nstudents)
relig_motivation <- ifelse( group == "black", -1, ifelse(group == "white", 0, 1))
relig_motivation <- rnorm(nstudents, mean = relig_motivation)
lottery <- (acad_motivation + relig_motivation) > 0
school <- ifelse( (runif(nstudents) + .8* lottery ) > 1, "private", "public")
test_score <- rnorm(nstudents, mean = 100 - 5 * (school == "private") + 20 * acad_motivation)
School_data <- data.frame(test_score, acad_motivation, group, relig_motivation, lottery, school)
```

Tadpoles *Swimming speed of tadpoles.*

Description

Tim Watkins examined the swimming speed of tadpoles as a function of the water temperature, and the water temperature at which the tadpoles had been raised. Since size is a major determinant of speed, the tadpole's length was measured as well.

Usage

```
data(Tadpoles)
```

Format

A data frame `Trucking_jobs` with 129 rows and 11 variables.

- `group` Whether the tadpole was raised in cold water ("c") or warm ("w").
- `rtemp` The temperature (C) of the water in which the swimming speed was measured. (The swimming channel is called a "race".)
- `length` The tadpole's length, in mm.
- `vmax` The maximum swimming speed attained in one trial, in mm/sec.

Details

It was hypothesized that tadpoles would swim faster in the temperature of water close to that in which they had been raised.

Examples

```
mod_1 = lm(vmax ~ poly(rtemp, 2) * group + length, data = Tadpoles)
```

Trucking_jobs *Earnings of workers at a trucking company.*

Description

A dataset from a mid-western US trucking company on annual earnings of its employees in 2007. Datasets like this are used in audits by the Federal Government to look for signs of discrimination.

Usage

```
data(Trucking_jobs)
```

Format

A data frame `Trucking_jobs` with 129 rows and 11 variables.

- `sex` The employee's sex: M or F
- `earnings` Annual earnings, in dollars. Hourly wages have been converted to a full-time basis.
- `age` The employee's age, in years.
- `title` The employee's job title.
- `hiredyears` How long the employee has been working for the company.

Examples

```
mod_1 = lm(earnings ~ age + hiredyears + sex, data = Trucking_jobs)
```

Used_Fords *Prices of used Ford automobiles in 2009*

Description

These data were compiled by Macalester College students for a class project. Then-undergraduates Elise delMas, Emiliano Urbina, and Candace Groth collected the data from `cars.com`

Usage

```
data(Used_Fords)
```

Format

A data frame `Used_Fords` with 635 rows and 7 variables.

- `Price` The asking price for the car in USD.
- `Year` The model year of the car.
- `Mileage` The reported odometer reading.
- `Location` Which of the several regions the car was marketed in.
- `Color` The car's body color.
- `Age` The age of the car at the time the data were collected in 2009. This is directly related to `Year`

Examples

```
mod_1 <- lm(Price ~ Mileage, data = Used_Fords)
mod_2 <- lm(Price ~ Mileage + Age, data = Used_Fords)
```

Index