Package ‘cvms’

September 29, 2019

Title Cross-Validation for Model Selection

Version 0.3.1

Description Cross-validate one or multiple regression and classification models and get relevant evaluation metrics in a tidy format. Validate the best model on a test set and compare it to a baseline evaluation. Alternatively, evaluate predictions from an external model. Currently supports regression and classification (binary and multiclass). Described in chp. 5 of Jeyaraman, B. P., Olsen, L. R., & Wambugu M. (2019, ISBN: 9781838550134).

License MIT + file LICENSE

URL `https://github.com/ludvigolsen/cvms`

BugReports `https://github.com/ludvigolsen/cvms/issues`

Depends R (>= 3.5)

Imports data.table (>= 1.12),
dplyr,
plyr,
tidyr (>= 0.8.3),
ggplot2,
purrr,
tibble (>= 2.1.1),
caret (>= 6.0-84),
pROC (>= 1.14.0),
stats,
lme4 (>= 1.1-21),
MuMIn (>= 1.43.6),
broom (>= 0.5.2),
stringr,
mltools (>= 0.3.5),
rlang,
utils,
lifecycle

Suggests knitr,
groupdata2 (>= 1.1.2),
e1071 (>= 1.7-2),
rmarkdown,
testthat (>= 2.2.1),
AUC,

furrr,
ModelMetrics (>= 1.2.2),
covr (>= 3.3.1),
nnet (>= 7.3-12),
randomForest (>= 4.6-14)

VignetteBuilder knitr

RdMacros lifecycle

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

R topics documented:

`baseline` *Create baseline evaluations*

Description

Maturing

Create a baseline evaluation of a test set.

When `family` is `gaussian`: fits baseline models (`y ~ 1`) on `n` random subsets of `train_data` and evalutes each model on `test_data`. Also evaluates a model fitted on all rows in `train_data`.

When `family` is `binomial`: evaluates `n` sets of random predictions against the dependent variable, along with a set of all `0` predictions and a set of all `1` predictions.

When `family` is `multinomial`: creates one-vs-all (binomial) baseline evaluations for `n` sets of random predictions against the dependent variable, along with sets of "all class x,y,z,..." predictions.

Usage

```
baseline(test_data, dependent_col, train_data = NULL, n = 100,
  family = "binomial", positive = 2, cutoff = 0.5,
  random_generator_fn = runif, random_effects = NULL,
  min_training_rows = 5, min_training_rows_left_out = 3,
  REML = FALSE, parallel = FALSE)
```

Arguments

`test_data`	Data Frame.
`dependent_col`	Name of dependent variable in the supplied test and training sets.
`train_data`	Data Frame. Only used when `family` == `"gaussian"`.
`n`	Number of random samplings to perform. For `gaussian`: The number of random samplings of train_data to fit baseline models on. For `binomial` and `multinomial`: The number of sets of random predictions to evaluate.
`family`	Name of family. (Character) Currently supports `"gaussian"`, `"binomial"` and `"multinomial"`.
`positive`	Level from dependent variable to predict. Either as character or level index (1 or 2 - alphabetically). E.g. if we have the levels `"cat"` and `"dog"` and we want `"dog"` to be the positive class, we can either provide `"dog"` or `2`, as alphabetically, `"dog"` comes after `"cat"`. Used when calculating confusion matrix metrics and creating ROC curves. N.B. Only affects evaluation metrics, not the returned predictions. **N.B. Binomial only**. (Character or Integer)
`cutoff`	Threshold for predicted classes. (Numeric) N.B. **Binomial only**
`random_generator_fn`	Function for generating random numbers when `type` is `"multinomial"`. The softmax function is applied to the generated numbers to transform them to probabilities. The first argument must be the number of random numbers to generate, as no other arguments are supplied. To test the effect of using different functions, see `multiclass_probability_tibble`. N.B. **Multinomial only**
`random_effects`	Random effects structure for Gaussian baseline model. (Character) E.g. with `"(1|ID)"`, the model becomes `"y ~ 1 + (1|ID)"`. N.B. **Gaussian only**
`min_training_rows`	Minimum number of rows in the random subsets of `train_data`. **Gaussian only**. (Integer)
`min_training_rows_left_out`	Minimum number of rows left out of the random subsets of `train_data`. I.e. a subset will maximally have the size: `max_rows_in_subset` = `nrow(train_data) -min_training_rows_left_out`. N.B. **Gaussian only**. (Integer)

`REML`	Whether to use Restricted Maximum Likelihood. (Logical) N.B. **Gaussian only**. (Integer)
`parallel`	Whether to run the `n` evaluations in parallel. (Logical) Remember to register a parallel backend first. E.g. with doParallel::registerDoParallel.

Details

Packages used:

Models:

Gaussian: `stats::lm`

Results: **Gaussian**:

r2m : `MuMIn::r.squaredGLMM`

r2c : `MuMIn::r.squaredGLMM`

AIC : `stats::AIC`

AICc : `MuMIn::AICc`

BIC : `stats::BIC`

Binomial and **Multinomial**:

Confusion matrix and related metrics: `caret::confusionMatrix`

ROC and related metrics: `pROC::roc`

MCC: `mltools::mcc`

Value

List containing:

1. a tibble with summarized results (called `summarized_metrics`)
2. a tibble with random evaluations (`random_evaluations`)
3. a tibble with the summarized class level results (`summarized_class_level_results`) **(Multinomial only)**

———————————————————————-

Gaussian Results:

———————————————————————-

The **Summarized Results** tibble contains:

Average **RMSE**, **MAE**, **r2m**, **r2c**, **AIC**, **AICc**, and **BIC**.

The **Measure** column indicates the statistical descriptor used on the evaluations. The row where `Measure == All_rows` is the evaluation when the baseline model is trained on all rows in `train_data`.

The **Training Rows** column contains the aggregated number of rows used from `train_data`, when fitting the baseline models.

....

The **Random Evaluations** tibble contains:

The **non-aggregated metrics**.

A nested tibble with the **predictions** and targets.

A nested tibble with the **coefficients** of the baseline models.

Number of **training rows** used when fitting the baseline model on the training set.

Specified **family**.

Name of **dependent** variable.

Name of **fixed** effect (bias term only).

Random effects structure (if specified).

———————————————————————-

Binomial Results:

———————————————————————-

Based on the generated test set predictions, a confusion matrix and ROC curve are used to get the following:

ROC:

AUC, **Lower CI**, and **Upper CI**

Confusion Matrix:

Balanced Accuracy, **F1**, **Sensitivity**, **Specificity**, **Positive Prediction Value**, **Negative Prediction Value**, **Kappa**, **Detection Rate**, **Detection Prevalence**, **Prevalence**, and **MCC** (Matthews correlation coefficient).

....

The **Summarized Results** tibble contains:

The **Measure** column indicates the statistical descriptor used on the evaluations. The row where `Measure == All_0` is the evaluation when all predictions are 0. The row where `Measure == All_1` is the evaluation when all predictions are 1.

The **aggregated metrics**.

....

The **Random Evaluations** tibble contains:

The **non-aggregated metrics**.

A nested tibble with the **predictions** and targets.

A nested tibble with the sensativities and specificities from the **ROC** curve.

A nested tibble with the **confusion matrix**. The `Pos_` columns tells you whether a row is a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN), depending on which level is the "positive" class. I.e. the level you wish to predict.

Specified **family**.

Name of **dependent** variable.

———————————————————————-

Multinomial Results:

———————————————————————-

Based on the generated test set predictions, one-vs-all (binomial) evaluations are performed and aggregated to get the same metrics as in the `binomial` results, with the addition of **Overall Accuracy** in the summarized results.

....

The **Summarized Results** tibble contains:

Summary of the random evaluations.

How: First, the one-vs-all binomial evaluations are aggregated by repetition (ignoring `NA`s), and then, these aggregations are summarized. Besides the metrics from the binomial evaluations (see *Binomial Results* above), it also includes the **Overall Accuracy** metric.

The **Measure** column indicates the statistical descriptor used on the evaluations. The **Mean**, **Median**, **SD**, and **IQR** describe the repetition evaluations (similar to the *Random Evaluations* tibble, but ignoring NAs when aggregating, as the NAs and INFs are counted instead), while the **Max**, **Min**, **NAs**, and **INFs** are extracted from the *Summarized Class Level Results* tibble, to get the overall values. The NAs and INFs are only counted in the one-vs-all evaluations.

The rows where Measure == All_<<class name>> are the evaluations when all the observations are predicted to be in that class.

....

The **Summarized Class Level Results** tibble contains:

The (nested) summarized results for each class, with the same metrics and descriptors as the *Summarized Results* tibble. Use tidyr::unnest on the tibble to inspect the results.

How: The one-vs-all evaluations are summarized by class.

The rows where Measure == All_0 are the evaluations when none of the observations are predicted to be in that class, while the rows where Measure == All_1 are the evaluations when all of the observations are predicted to be in that class.

....

The **Random Evaluation** tibble contains:

The repetition results with the same metrics as the *Summarized Results* tibble.

How: The one-vs-all evaluations are aggregated by repetition. NA's are not ignored, meaning that any NA from a one-vs-all evaluation will lead to an NA result for that repetition.

Also includes:

A nested tibble with the one-vs-all binomial evaluations (**Class Level Results**), including nested **ROC** curves and **Confusion Matrices**, and the **Support** column, which is a count of how many observations from the class is in the test set.

A nested tibble with the **predictions** and targets.

A nested tibble with the multiclass **confusion matrix**.

Specified **family**.

Name of **dependent** variable.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

Examples

```
# Attach packages
library(cvms)
library(groupdata2) # partition()
library(dplyr) # %>% arrange()
library(tibble)

# Data is part of cvms
data <- participant.scores

# Set seed for reproducibility
set.seed(1)

# Partition data
partitions <- partition(data, p = 0.7, list_out = TRUE)
train_set <- partitions[[1]]
test_set <- partitions[[2]]

# Create baseline evaluations
# Note: usually n=100 is a good setting

# Gaussian
baseline(test_data = test_set, train_data = train_set,
         dependent_col = "score", random_effects = "(1|session)",
         n = 2, family = "gaussian")

# Binomial
baseline(test_data = test_set, dependent_col = "diagnosis",
         n = 2, family = "binomial")

# Multinomial

# Create some data with multiple classes
multiclass_data <- tibble(
    "target" = rep(paste0("class_", 1:5), each = 10)) %>%
    dplyr::sample_n(35)

baseline(test_data = multiclass_data,
         dependent_col = "target",
         n = 4, family = "multinomial")

# Parallelize evaluations

# Attach doParallel and register four cores
# Uncomment:
# library(doParallel)
# registerDoParallel(4)

# Binomial
baseline(test_data = test_set, dependent_col = "diagnosis",
         n = 4, family = "binomial", parallel = TRUE)

# Gaussian
baseline(test_data = test_set, train_data = train_set,
         dependent_col = "score", random_effects = "(1|session)",
         n = 4, family = "gaussian", parallel = TRUE)

# Multinomial
(mb <- baseline(test_data = multiclass_data,
               dependent_col = "target",
               n = 4, family = "multinomial",
               parallel = TRUE))

# Inspect the summarized class level results
# for class_2
mb$summarized_class_level_results %>%
 dplyr::filter(Class == "class_2") %>%
 tidyr::unnest(Results)

# Multinomial with custom random generator function
# that creates very "certain" predictions
# (once softmax is applied)

rcertain <- function(n){
    (runif(n, min = 1, max = 100)^1.4)/100
}
```

```
baseline(test_data = multiclass_data,
         dependent_col = "target",
         n = 4, family = "multinomial",
         parallel = TRUE,
         random_generator_fn = rcertain)
```

---

combine_predictors *Generate model formulas by combining predictors*

Description

Maturing

Create model formulas with every combination of your fixed effects, along with the dependent variable and random effects. 259,358 formulas have been precomputed with two- and three-way interactions for up to 8 fixed effects, with up to 5 included effects per formula. Uses the + and * operators, so lower order interactions are automatically included.

Usage

```
combine_predictors(dependent, fixed_effects, random_effects = NULL,
  max_fixed_effects = 5, max_interaction_size = 3,
  max_effect_frequency = NULL)
```

Arguments

dependent	Name of dependent variable. (Character)
fixed_effects	List of fixed effects. (Character) Max. limit of 8 effects **when interactions are included**! A fixed effect name cannot contain: white spaces, "*" or "+". Effects in sublists will be interchanged. This can be useful, when we have multiple versions of a predictor (e.g. x1 and log(x1)) that we do not wish to have in the same formula. Example of interchangeable effects: `list( list( "x1","log_x1" ),"x2","x3" )`
random_effects	The random effects structure. (Character) Is appended to the model formulas.
max_fixed_effects	Maximum number of fixed effects in a model formula. (Integer) Max. limit of 5 **when interactions are included**!
max_interaction_size	Maximum number of effects in an interaction. (Integer) Max. limit of 3. Use this to limit the n-way interactions allowed. 0 or 1 excludes interactions all together. A model formula can contain multiple interactions.
max_effect_frequency	Maximum number of times an effect is included in a formula string.

Value

List of model formulas.

E.g.:

`c("y ~ x1 + (1|z)","y ~ x2 + (1|z)","y ~ x1 + x2 + (1|z)","y ~ x1 * x2 + (1|z)").`

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

Examples

```
# Attach packages
library(cvms)

# Create effect names
dependent <- "y"
fixed_effects <- c("a","b","c")
random_effects <- "(1|e)"


# Create model formulas
combine_predictors(dependent, fixed_effects,
                   random_effects)


# Create effect names with interchangeable effects in sublists
fixed_effects <- list("a",list("b","log_b"),"c")


# Create model formulas
combine_predictors(dependent, fixed_effects,
                   random_effects)
```

`compatible.formula.terms` *Compatible formula terms*

Description

162,660 pairs of compatible terms for building model formulas with up to 15 fixed effects.

Format

A data frame with 162,660 rows and 5 variables:

left term, fixed effect or interaction, with fixed effects separated by "`*`"

right term, fixed effect or interaction, with fixed effects separated by "`*`"

max_interaction_size maximum interaction size in the two terms, up to `3`

num_effects number of unique fixed effects in the two terms, up to `5`

min_num_fixed_effects minimum number of fixed effects required to use a formula with the two terms, i.e. the index in the alphabet of the last of the alphabetically ordered effects (letters) in the two terms, so 4 if `left == "A"` and `right == "D"`

Details

A term is either a fixed effect or an interaction between fixed effects (up to three-way), where the effects are separated by the "*" operator.

Two terms are compatible if they are not redundant, meaning that both add a fixed effect to the formula. E.g. as the interaction `"x1 * x2 * x3"` expands to `"x1 + x2 + x3 + x1 * x2 + x1 * x3 + x2 * x3 + x1 * x2 * x3"`, the higher order interaction makes these "sub terms" redundant. Note: All terms are compatible with `NA`.

Effects are represented by the first fifteen capital letters.

Used to generate the model formulas for `combine_predictors`.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

---

cross_validate — *Cross-validate regression models for model selection*

Description

Stable

Cross-validate one or multiple gaussian or binomial models at once. Perform repeated cross-validation. Returns results in a tibble for easy comparison, reporting and further analysis.

See `cross_validate_fn()` for use with custom model functions.

Usage

```
cross_validate(data, models, fold_cols = ".folds", family = "gaussian",
  link = NULL, control = NULL, REML = FALSE, cutoff = 0.5,
  positive = 2, metrics = list(), rm_nc = FALSE, parallel = FALSE,
  model_verbose = FALSE)
```

Arguments

`data`	Data frame. Must include grouping factor for identifying folds - as made with `groupdata2::fold()`.
`models`	Model formulas as strings. (Character) E.g. `c("y~x","y~z")`. Can contain random effects. E.g. `c("y~x+(1|r)","y~z+(1|r)")`.
`fold_cols`	Name(s) of grouping factor(s) for identifying folds. (Character) Include names of multiple grouping factors for repeated cross-validation.
`family`	Name of family. (Character) Currently supports `"gaussian"` and `"binomial"`.
`link`	Link function. (Character) E.g. `link = "log"` with `family = "gaussian"` will use `family = gaussian(link = "log")`. See `stats::family` for available link functions.

	Default link functions: Gaussian: `'identity'`. Binomial: `'logit'`.
`control`	Construct control structures for mixed model fitting (i.e. `lmer` and `glmer`). See `lme4::lmerControl` and `lme4::glmerControl`. N.B. Ignored if fitting `lm` or `glm` models.
`REML`	Restricted Maximum Likelihood. (Logical)
`cutoff`	Threshold for predicted classes. (Numeric) N.B. **Binomial models only**
`positive`	Level from dependent variable to predict. Either as character or level index (`1` or `2` - alphabetically). E.g. if we have the levels `"cat"` and `"dog"` and we want `"dog"` to be the positive class, we can either provide `"dog"` or `2`, as alphabetically, `"dog"` comes after `"cat"`. Used when calculating confusion matrix metrics and creating ROC curves. N.B. Only affects evaluation metrics, not the model training or returned predictions. N.B. **Binomial models only**.
`metrics`	List for enabling/disabling metrics. E.g. `list("RMSE" = FALSE)` would remove RMSE from the results, and `list("Accuracy" = TRUE)` would add the regular accuracy metric to the classification results. Default values (TRUE/FALSE) will be used for the remaining metrics available. Also accepts the string `"all"`. N.B. Currently, disabled metrics are still computed.
`rm_nc`	Remove non-converged models from output. (Logical)
`parallel`	Whether to cross-validate the list of models in parallel. (Logical) Remember to register a parallel backend first. E.g. with `doParallel::registerDoParallel`.
`model_verbose`	Message name of used model function on each iteration. (Logical)

Details

Packages used:

Models:

Gaussian: stats::lm, `lme4::lmer`

Binomial: `stats::glm`, `lme4::glmer`

Results:

Gaussian:
r2m : `MuMIn::r.squaredGLMM`
r2c : `MuMIn::r.squaredGLMM`
AIC : `stats::AIC`
AICc : `MuMIn::AICc`
BIC : `stats::BIC`

Binomial:
Confusion matrix: `caret::confusionMatrix`
ROC: `pROC::roc`
MCC: `mltools::mcc`

Value

Tbl (tibble) with results for each model.

Shared across families: A nested tibble with **coefficients** of the models from all iterations.

Number of *total* **folds**.

Number of **fold columns**.

Count of **convergence warnings**. Consider discarding models that did not converge on all iterations. Note: you might still see results, but these should be taken with a grain of salt!

Count of **other warnings**. These are warnings without keywords such as "convergence".

Count of **Singular Fit messages**. See `?lme4::isSingular` for more information.

Nested tibble with the **warnings and messages** caught for each model.

Specified **family**.

Specified **link** function.

Name of **dependent** variable.

Names of **fixed** effects.

Names of **random** effects, if any.

———————————————————————-

Gaussian Results:

———————————————————————-

Average **RMSE**, **MAE**, **r2m**, **r2c**, **AIC**, **AICc**, and **BIC** of all the iterations*, **omitting potential NAs** *from non-converged iterations*. Note that the Information Criteria metrics (AIC, AICc, and BIC) are also averages.

A nested tibble with the **predictions** and targets.

A nested tibble with the non-averaged **results** from all iterations.

* In *repeated cross-validation*, the metrics are first averaged for each fold column (repetition) and then averaged again.

———————————————————————-

Binomial Results:

———————————————————————-

Based on the **collected** predictions from the test folds*, a confusion matrix and a ROC curve are created to get the following:

ROC:

AUC, **Lower CI**, and **Upper CI**

Confusion Matrix:

Balanced Accuracy, **F1**, **Sensitivity**, **Specificity**, **Positive Prediction Value**, **Negative Prediction Value**, **Kappa**, **Detection Rate**, **Detection Prevalence**, **Prevalence**, and **MCC** (Matthews correlation coefficient).

Other available metrics (disabled by default, see `metrics`): **Accuracy**.

Also includes:

A nested tibble with **predictions**, predicted classes (depends on `cutoff`), and the targets. Note, that the **predictions are not necessarily of the specified** `positive` **class**, but of the model's positive class (second level of dependent variable, alphabetically).

A nested tibble with the sensativities and specificities from the **ROC** curve(s).

A nested tibble with the **confusion matrix**/matrices. The `Pos_` columns tells you whether a row is a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN), depending on which level is the "positive" class. I.e. the level you wish to predict.

A nested tibble with the **results** from all fold columns, when using *repeated cross-validation*.
* In *repeated cross-validation*, an evaluation is made per fold column (repetition) and averaged.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

Benjamin Hugh Zachariae

See Also

Other validation functions: `cross_validate_fn`, `validate`

Examples

```
# Attach packages
library(cvms)
library(groupdata2) # fold()
library(dplyr) # %>% arrange()

# Data is part of cvms
data <- participant.scores

# Set seed for reproducibility
set.seed(7)

# Fold data
data <- fold(data, k = 4,
             cat_col = 'diagnosis',
             id_col = 'participant') %>%
        arrange(.folds)

# Cross-validate a single model


# Gaussian
cross_validate(data,
               models = "score~diagnosis",
               family = 'gaussian',
               REML = FALSE)

# Binomial
cross_validate(data,
               models = "diagnosis~score",
               family='binomial')

# Cross-validate multiple models

models <- c("score~diagnosis+(1|session)",
            "score~age+(1|session)")

cross_validate(data,
               models = models,
               family = 'gaussian',
               REML = FALSE)

# Use non-default link functions
```

```
cross_validate(data,
               models = "score~diagnosis",
               family = 'gaussian',
               link = 'log',
               REML = FALSE)

# Use parallelization


# Attach doParallel and register four cores
# Uncomment:
# library(doParallel)
# registerDoParallel(4)

# Create list of 20 model formulas
models <- rep(c("score~diagnosis+(1|session)",
                "score~age+(1|session)"), 10)

# Cross-validate a list of 20 model formulas in parallel
system.time({cross_validate(data,
                            models = models,
                            family = 'gaussian',
                            parallel = TRUE)})

# Cross-validate a list of 20 model formulas sequentially
system.time({cross_validate(data,
                            models = models,
                            family = 'gaussian',
                            parallel = FALSE)})
```

---

cross_validate_fn   *Cross-validate custom model functions for model selection*

Description

Experimental

Cross-validate your model function with one or multiple model formulas at once. Perform repeated cross-validation. Returns results in a tibble for easy comparison, reporting and further analysis.

Compared to `cross_validate()`, this function allows you supply a custom model function and (if needed) a custom predict function.

Supports regression and classification (binary and multiclass). See `type`.

Note that some metrics may not be computable for all types of model objects.

Usage

```
cross_validate_fn(data, model_fn, formulas, fold_cols = ".folds",
  type = "gaussian", cutoff = 0.5, positive = 2,
  predict_type = NULL, predict_fn = NULL, metrics = list(),
  rm_nc = FALSE, parallel = FALSE)
```

Arguments

`data`	Data frame. Must include grouping factor for identifying folds - as made with `groupdata2::fold()`.
`model_fn`	Model function that returns a fitted model object. Will usually wrap an existing model function like `e1071::svm` or `nnet::multinom`. Must have the following function arguments: `function(train_data,formula)`
`formulas`	Model formulas as strings. (Character) Will be converted to `formula` objects before being passed to `model_fn`. E.g. `c("y~x","y~z")`. Can contain random effects. E.g. `c("y~x+(1|r)","y~z+(1|r)")`.
`fold_cols`	Name(s) of grouping factor(s) for identifying folds. (Character) Include names of multiple grouping factors for repeated cross-validation.
`type`	Type of evaluation to perform: `"gaussian"` for regression (like linear regression). `"binomial"` for binary classification. `"multinomial"` for multiclass classification.
`cutoff`	Threshold for predicted classes. (Numeric) N.B. **Binomial models only**
`positive`	Level from dependent variable to predict. Either as character or level index (1 or 2 - alphabetically). E.g. if we have the levels `"cat"` and `"dog"` and we want `"dog"` to be the positive class, we can either provide `"dog"` or `2`, as alphabetically, `"dog"` comes after `"cat"`. Used when calculating confusion matrix metrics and creating ROC curves. N.B. Only affects evaluation metrics, not the model training or returned predictions. N.B. **Binomial models only**.
`predict_type`	The `type` argument for `predict()`. When the defaults fail, provide it such that the `predict()` output is as follows:

Binomial: Vector or one-column matrix / data frame with probabilities (0-1). E.g.:
`c(0.3,0.5,0.1,0.5)`

Gaussian: Vector or one-column matrix / data frame with the predicted value. E.g.:
`c(3.7,0.9,1.2,7.3)`

Multinomial: Data frame with one column per class containing probabilities of the class. Column names should be identical to how the class names are written in the target column. E.g.:

class_1	**class_2**	**class_3**
0.269	0.528	0.203
0.368	0.322	0.310
0.375	0.371	0.254
...	...	...

N.B. `predict_fn` and `predict_type` are mutually exclusive. Specify only one of them.

`predict_fn` Function for predicting the targets in the test folds using the fitted model object. Will usually wrap `predict()`, but doesn't have to. Must return predictions in the format described in `predict_type` above.

Must have the following function arguments:

`function(test_data,model,formula = NULL)`

N.B. `predict_fn` and `predict_type` are mutually exclusive. Specify only one of them.

`metrics` List for enabling/disabling metrics.

E.g. `list("RMSE" = FALSE)` would remove RMSE from the results, and `list("Accuracy" = TRUE)` would add the regular accuracy metric to the classification results. Default values (TRUE/FALSE) will be used for the remaining metrics available.

Also accepts the string `"all"`.

N.B. Currently, disabled metrics are still computed.

`rm_nc` Remove non-converged models from output. (Logical)

`parallel` Whether to cross-validate the list of models in parallel. (Logical)

Remember to register a parallel backend first. E.g. with `doParallel::registerDoParallel`.

Details

Packages used:

Results:

Gaussian:
r2m : `MuMIn::r.squaredGLMM`
r2c : `MuMIn::r.squaredGLMM`
AIC : `stats::AIC`
AICc : `MuMIn::AICc`
BIC : `stats::BIC`

Binomial:
Confusion matrix: `caret::confusionMatrix`
ROC: `pROC::roc`
MCC: `mltools::mcc`

Value

Tbl (tibble) with results for each model.

Shared across families: A nested tibble with **coefficients** of the models from all iterations. The coefficients are extracted from the model object with `broom::tidy()` or `coef()` (with some restrictions on the output). If these attempts fail, a default coefficients tibble filled with `NA`s is returned.

Number of *total* **folds**.

Number of **fold columns**.

Count of **convergence warnings**, using a limited set of keywords (e.g. "convergence"). If a convergence warning does not contain one of these keywords, it will be counted with **other warnings**. Consider discarding models that did not converge on all iterations. Note: you might still see results, but these should be taken with a grain of salt!

Nested tibble with the **warnings and messages** caught for each model.

Specified **family**.

Name of **dependent** variable.

Names of **fixed** effects.

Names of **random** effects, if any.

———————————————————————-

Gaussian Results:

———————————————————————-

Average **RMSE**, **MAE**, **r2m**, **r2c**, **AIC**, **AICc**, and **BIC** of all the iterations*, **omitting potential NAs** *from non-converged iterations*. Some metrics will return `NA` if they can't be extracted from the fitted model objects.

N.B. The Information Criteria metrics (AIC, AICc, and BIC) are also averages.

A nested tibble with the **predictions** and targets.

A nested tibble with the non-averaged **results** from all iterations.

* In *repeated cross-validation*, the metrics are first averaged for each fold column (repetition) and then averaged again.

———————————————————————-

Binomial Results:

———————————————————————-

Based on the **collected** predictions from the test folds*, a confusion matrix and a ROC curve are created to get the following:

ROC:

AUC, **Lower CI**, and **Upper CI**

Confusion Matrix:

Balanced Accuracy, **F1**, **Sensitivity**, **Specificity**, **Positive Prediction Value**, **Negative Prediction Value**, **Kappa**, **Detection Rate**, **Detection Prevalence**, **Prevalence**, and **MCC** (Matthews correlation coefficient).

Other available metrics (disabled by default, see `metrics`): **Accuracy**.

Also includes:

A nested tibble with the **predictions**, predicted classes (depends on `cutoff`), and targets. Note, that the **predictions are not necessarily of the specified** `positive` **class**, but of the model's positive class (second level of dependent variable, alphabetically).

A nested tibble with the sensativities and specificities from the **ROC** curves.

A nested tibble with the **confusion matrix**/matrices. The `Pos_` columns tells you whether a row is a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN), depending on which level is the "positive" class. I.e. the level you wish to predict.

A nested tibble with the **results** from all fold columns, when using *repeated cross-validation*.

* In *repeated cross-validation*, an evaluation is made per fold column (repetition) and averaged.

———————————————————————-

Multinomial Results:

———————————————————————-

For each class, a *one-vs-all* binomial evaluation is performed. This creates a **class level results** tibble containing the same metrics as the binomial results described above, along with the **Support** metric, which is simply a count of the class in the target column. These metrics are used to calculate the macro metrics in the output tibble. The nested class level results tibble is also included in the output tibble, and would usually be reported along with the macro and overall metrics.

The output tibble contains the macro and overall metrics. The metrics that share their name with the metrics in the nested class level results tibble are averages of those metrics (note: does not remove `NA`s before averaging). In addition to these, it also includes the **Overall Accuracy** metric.

Other available metrics (disabled by default, see `metrics`): **Accuracy**, **Weighted Balanced Accuracy**, **Weighted Accuracy**, **Weighted F1**, **Weighted Sensitivity**, **Weighted Sensitivity**, **Weighted Specificity**, **Weighted Pos Pred Value**, **Weighted Neg Pred Value**, **Weighted AUC**, **Weighted Lower CI**, **Weighted Upper CI**, **Weighted Kappa**, **Weighted MCC**, **Weighted Detection Rate**, **Weighted Detection Prevalence**, and **Weighted Prevalence**.

Note that the "Weighted" metrics are weighted averages, weighted by the `Support`.

Also includes:

A nested tibble with the **predictions**, predicted classes, and targets.

A nested tibble with the multiclass **Confusion Matrix**.

Class Level Results

The nested class level results tibble also includes:

A nested tibble with the sensativities and specificities from the **ROC** curve.

A nested tibble with the **confusion matrix** from the one-vs-all evaluation. The `Pos_` columns tells you whether a row is a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN), depending on which level is the "positive" class. In our case, `1` is the current class and `0` represents all the other classes together.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

Benjamin Hugh Zachariae

See Also

Other validation functions: `cross_validate`, `validate`

Examples

```
# Attach packages
library(cvms)
library(groupdata2) # fold()
library(dplyr) # %>% arrange() mutate()

# Data is part of cvms
data <- participant.scores

# Set seed for reproducibility
set.seed(7)

# Fold data
data <- fold(data, k = 4,
             cat_col = 'diagnosis',
             id_col = 'participant') %>%
    mutate(diagnosis = as.factor(diagnosis)) %>%
    arrange(.folds)

# Cross-validate multiple formulas
```

```
formulas_gaussian <- c("score ~ diagnosis",
                       "score ~ age")
formulas_binomial <- c("diagnosis ~ score",
                       "diagnosis ~ age")


# Gaussian

# Create model function with args 'train_data' and 'formula'
# that returns a fitted model object
lm_model_fn <- function(train_data, formula){
    lm(formula = formula, data = train_data)
}

# Cross-validate the model function
cross_validate_fn(data,
                  model_fn = lm_model_fn,
                  formulas = formulas_gaussian,
                  type = 'gaussian',
                  fold_cols = ".folds")

# Binomial

# Create model function with args 'train_data' and 'formula'
# that returns a fitted model object
glm_model_fn <- function(train_data, formula){
    glm(formula = formula, data = train_data, family = "binomial")
}

# Cross-validate the model function
cross_validate_fn(data,
                  model_fn = glm_model_fn,
                  formulas = formulas_binomial,
                  type = 'binomial',
                  fold_cols = ".folds")

# Support Vector Machine (svm)

# Create model function with args 'train_data' and 'formula'
# that returns a fitted model object
svm_model_fn <- function(train_data, formula){
    e1071::svm(formula = formula,
               data = train_data,
               kernel = "linear",
               type = "C-classification")
}

# Cross-validate the model function
cross_validate_fn(data,
                  model_fn = svm_model_fn,
                  formulas = formulas_binomial,
                  type = 'binomial',
                  fold_cols = ".folds")

# Naive Bayes

# Create model function with args 'train_data' and 'formula'
# that returns a fitted model object
nb_model_fn <- function(train_data, formula){
    e1071::naiveBayes(formula = formula,
                      data = train_data)
}

# Create predict function with args 'test_data', 'model', and 'formula'
# that returns predictions in right format (here, a one-column matrix)
# Note the type = "raw" and that we pick the probabilities for class 1 with [,2]
nb_predict_fn <- function(test_data, model, formula = NULL){
  stats::predict(object = model, newdata = test_data,
                 type = "raw", allow.new.levels = TRUE)[,2]
}

# Cross-validate the model function
cross_validate_fn(data,
                  model_fn = nb_model_fn,
                  formulas = formulas_binomial,
                  type = 'binomial',
                  predict_fn = nb_predict_fn,
                  fold_cols = ".folds")


# Use parallelization


# Attach doParallel and register four cores
# Uncomment:
# library(doParallel)
# registerDoParallel(4)

# Create list of 20 model formulas
formulas <- rep(c("score~diagnosis",
                "score~age"), 10)

# Cross-validate a list of 20 model formulas in parallel
system.time({cross_validate_fn(data,
                               model_fn = lm_model_fn,
                               formulas = formulas,
                               type = 'gaussian',
                               fold_cols = ".folds",
                               parallel = TRUE)})

# Cross-validate a list of 20 model formulas sequentially
system.time({cross_validate_fn(data,
                               model_fn = lm_model_fn,
                               formulas = formulas,
                               type = 'gaussian',
                               fold_cols = ".folds",
                               parallel = FALSE)})
```

`cvms` *cvms: A package for cross-validating regression and classification models*

Description

Perform (repeated) cross-validation on a list of model formulas. Validate the best model on a validation set. Perform baseline evaluations on your test set. Generate model formulas by combining your fixed effects. Evaluate predictions from an external model.

Details

Returns results in a tibble for easy comparison, reporting and further analysis.

The cvms package provides 5 main functions: `cross_validate`, `cross_validate_fn`, `validate`, `baseline`, and `evaluate`.

And a couple of helper functions: `combine_predictors`, `select_metrics`, `reconstruct_formulas`, `cv_plot`.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

`cv_plot` *Wrapper for plotting common plots using ggplot2*

Description

Experimental

Creates various plots based on the output of cvms::cross_validate()

Usage

```
cv_plot(x, type)
```

Arguments

`x`	Object returned by cvms::cross_validate() (tbl)
`type`	Type of plot. **Gaussian:** 'RMSE' - boxplot 'r2' - boxplot 'IC' - boxplot 'coefficients' - boxplot **Binomial:** "ROC" - ROC curve

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

Examples

```
# Attach packages
library(cvms)
library(groupdata2) # fold()

# Load data (included in cvms)
data <- participant.scores

# Fold data
data <- fold(data, k = 4,
             cat_col = 'diagnosis',
             id_col = 'participant')

# Cross-validate a gaussian model
CVgauss <- cross_validate(data,
                          "score~diagnosis",
                          family='gaussian')

# Plot results for gaussian model
cv_plot(CVgauss, type = 'RMSE')
cv_plot(CVgauss, type = 'r2')
cv_plot(CVgauss, type = 'IC')
cv_plot(CVgauss, type = 'coefficients')

# Cross-validate a binomial model
CVbinom <- cross_validate(data,
                          "diagnosis~score",
                          family='binomial')

# Plot results for binomial model
cv_plot(CVbinom, type = 'ROC')
```

`evaluate` *Evaluate your model's performance*

Description

Maturing

Evaluate your model's predictions on a set of evaluation metrics.

Create ID-aggregated evaluations by multiple methods.

Currently supports regression and classification (binary and multiclass). See `type`.

Usage

```
evaluate(data, target_col, prediction_cols, type = "gaussian",
  id_col = NULL, id_method = "mean", models = NULL,
  apply_softmax = TRUE, cutoff = 0.5, positive = 2,
  metrics = list(), include_predictions = TRUE, parallel = FALSE)
```

Arguments

`data` Data frame with predictions, targets and (optionally) an ID column. Can be grouped with `group_by`.

Multinomial: When `type` is `"multinomial"`, the predictions should be passed as one column per class with the probability of that class. The columns should have the name of their class, as they are named in the target column. E.g.:

class_1	class_2	class_3	target
0.269	0.528	0.203	class_2
0.368	0.322	0.310	class_3
0.375	0.371	0.254	class_2
...	...	...	...

Binomial: When `type` is `"binomial"`, the predictions should be passed as one column with the probability of class being the second class alphabetically (1 if classes are 0 and 1). E.g.:

prediction	target
0.769	1
0.368	1
0.375	0
...	...

Gaussian: When `type` is `"gaussian"`, the predictions should be passed as one column with the predicted values. E.g.:

prediction	target
28.9	30.2
33.2	27.1
23.4	21.3
...	...

`target_col` Name of the column with the true classes/values in `data`.

When `type` is `"multinomial"`, this column should contain the class names, not their indices.

`prediction_cols`

Name(s) of column(s) with the predictions.

When evaluating a classification task, the column(s) should contain the predicted probabilities.

`type` Type of evaluation to perform:

`"gaussian"` for regression (like linear regression).

`"binomial"` for binary classification.

`"multinomial"` for multiclass classification.

`id_col` Name of ID column to aggregate predictions by.

N.B. Current methods assume that the target class/value is constant within the IDs.

N.B. When aggregating by ID, some metrics (such as those from model objects) are excluded.

`id_method` Method to use when aggregating predictions by ID. Either `"mean"` or `"majority"`.

When `type` is `gaussian`, only the `"mean"` method is available.

mean: The average prediction (value or probability) is calculated per ID and evaluated. This method assumes that the target class/value is constant within the IDs.

majority: The most predicted class per ID is found and evaluated. In case of a tie, the winning classes share the probability (e.g. `P = 0.5` each when two majority classes). This method assumes that the target class/value is constant within the IDs.

`models` Unnamed list of fitted model(s) for calculating R^2 metrics and information criterion metrics. May only work for some types of models.

When only passing one model, remember to pass it in a list (e.g. `list(m)`).

N.B. When `data` is grouped, provide one model per group in the same order as the groups.

N.B. When aggregating by ID (i.e. when `id_col` is not `NULL`), it's not currently possible to pass model objects, as these would not be aggregated by the IDs.

N.B. Currently, **Gaussian only**.

`apply_softmax` Whether to apply the softmax function to the prediction columns when `type` is `"multinomial"`.

N.B. **Multinomial models only**.

`cutoff` Threshold for predicted classes. (Numeric)

N.B. **Binomial models only**.

`positive` Level from dependent variable to predict. Either as character or level index (1 or 2 - alphabetically).

E.g. if we have the levels `"cat"` and `"dog"` and we want `"dog"` to be the positive class, we can either provide `"dog"` or `2`, as alphabetically, `"dog"` comes after `"cat"`.

Used when calculating confusion matrix metrics and creating ROC curves.

N.B. Only affects the evaluation metrics.

N.B. **Binomial models only**.

`metrics` List for enabling/disabling metrics.

E.g. `list("RMSE" = FALSE)` would remove RMSE from the results, and `list("Accuracy" = TRUE)` would add the regular accuracy metric to the classification results. Default values (TRUE/FALSE) will be used for the remaining metrics available.

Also accepts the string `"all"`.

N.B. Currently, disabled metrics are still computed.

`include_predictions`
Whether to include the predictions in the output as a nested tibble. (Logical)

`parallel` Whether to run evaluations in parallel, when `data` is grouped with `group_by`.

Details

Packages used:

Gaussian:

r2m : `MuMIn::r.squaredGLMM`

r2c : `MuMIn::r.squaredGLMM`

AIC : `stats::AIC`

AICc : `MuMIn::AICc`

BIC : `stats::BIC`

Binomial and **Multinomial**:

Confusion matrix and related metrics: `caret::confusionMatrix`

ROC and related metrics: `pROC::roc`

MCC: `mltools::mcc`

Value

Gaussian Results:

Tibble containing the following metrics by default:

Average **RMSE**, **MAE**, **r2m**, **r2c**, **AIC**, **AICc**, and **BIC**.

N.B. Some of the metrics will only be returned if model objects were passed, and will be `NA` if they could not be extracted from the passed model objects.

Also includes:

A nested tibble with the **Predictions** and targets.

A nested tibble with the model **Coefficients**. The coefficients are extracted from the model object with `broom::tidy()` or `coef()` (with some restrictions on the output). If these attempts fail, a default coefficients tibble filled with `NA`s is returned.

Binomial Results:

Tibble with the following evaluation metrics, based on a confusion matrix and a ROC curve fitted to the predictions:

ROC:

AUC, **Lower CI**, and **Upper CI**

Confusion Matrix:

Balanced Accuracy, **F1**, **Sensitivity**, **Specificity**, **Positive Prediction Value**, **Negative Prediction Value**, **Kappa**, **Detection Rate**, **Detection Prevalence**, **Prevalence**, and **MCC** (Matthews correlation coefficient).

Other available metrics (disabled by default, see `metrics`): **Accuracy**.

Also includes:

A nested tibble with the **predictions** and targets.

A nested tibble with the sensativities and specificities from the **ROC** curve.

A nested tibble with the **confusion matrix**. The `Pos_` columns tells you whether a row is a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN), depending on which level is the "`positive`" class. I.e. the level you wish to predict.

Multinomial Results:

For each class, a *one-vs-all* binomial evaluation is performed. This creates a **Class Level Results** tibble containing the same metrics as the binomial results described above, along with the **Support** metric, which is simply a count of the class in the target column. These metrics are used to calculate the macro metrics in the output tibble. The nested class level results tibble is also included in the output tibble, and would usually be reported along with the macro and overall metrics.

The output tibble contains the macro and overall metrics. The metrics that share their name with the metrics in the nested class level results tibble are averages of those metrics (note: does not remove NAs before averaging). In addition to these, it also includes the **Overall Accuracy** metric.

Other available metrics (disabled by default, see metrics): **Accuracy**, **Weighted Balanced Accuracy**, **Weighted Accuracy**, **Weighted F1**, **Weighted Sensitivity**, **Weighted Sensitivity**, **Weighted Specificity**, **Weighted Pos Pred Value**, **Weighted Neg Pred Value**, **Weighted AUC**, **Weighted Lower CI**, **Weighted Upper CI**, **Weighted Kappa**, **Weighted MCC**, **Weighted Detection Rate**, **Weighted Detection Prevalence**, and **Weighted Prevalence**.

Note that the "Weighted" metrics are weighted averages, weighted by the Support.

Also includes:

A nested tibble with the **Predictions** and targets.

A nested tibble with the multiclass **Confusion Matrix**.

Class Level Results

Besides the binomial evaluation metrics and the Support metric, the nested class level results tibble also contains:

A nested tibble with the sensativities and specificities from the **ROC** curve.

A nested tibble with the **Confusion Matrix** from the one-vs-all evaluation. The Pos_ columns tells you whether a row is a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN), depending on which level is the "positive" class. In our case, 1 is the current class and 0 represents all the other classes together.

Author(s)

Ludvig Renbo Olsen, <r-pkgs@ludvigolsen.dk>

Examples

```
# Attach packages
library(cvms)
library(dplyr)

# Load data
data <- participant.scores

# Fit models
gaussian_model <- lm(age ~ diagnosis, data = data)
binomial_model <- glm(diagnosis ~ score, data = data)

# Add predictions
data[["gaussian_predictions"]] <- predict(gaussian_model, data,
                                          type = "response",
                                          allow.new.levels = TRUE)
data[["binomial_predictions"]] <- predict(binomial_model, data,
                                          allow.new.levels = TRUE)

# Gaussian evaluation
evaluate(data = data, target_col = "age",
         prediction_cols = "gaussian_predictions",
         models = list(gaussian_model),
         type = "gaussian")

# Binomial evaluation
evaluate(data = data, target_col = "diagnosis",
         prediction_cols = "binomial_predictions",
         type = "binomial")

# Multinomial

# Create a tibble with predicted probabilities
data_mc <- multiclass_probability_tibble(
    num_classes = 3, num_observations = 30,
    apply_softmax = TRUE, FUN = runif,
    class_name = "class_")

# Add targets
class_names <- paste0("class_", c(1,2,3))
data_mc[["target"]] <- sample(x = class_names,
                              size = 30, replace = TRUE)

# Multinomial evaluation
evaluate(data = data_mc, target_col = "target",
         prediction_cols = class_names,
         type = "multinomial")

# ID evaluation

# Gaussian ID evaluation
# Note that 'age' is the same for all observations
# of a participant
evaluate(data = data, target_col = "age",
         prediction_cols = "gaussian_predictions",
         id_col = "participant",
         type = "gaussian")

# Binomial ID evaluation
evaluate(data = data, target_col = "diagnosis",
         prediction_cols = "binomial_predictions",
         id_col = "participant",
         id_method = "mean", # alternatively: "majority"
         type = "binomial")

# Multinomial ID evaluation

# Add IDs and new targets (must be constant within IDs)
data_mc[["target"]] <- NULL
data_mc[["id"]] <- rep(1:6, each = 5)
id_classes <- tibble::tibble(
    "id" = 1:6,
    target = sample(x = class_names, size = 6, replace = TRUE)
)
data_mc <- data_mc %>%
    dplyr::left_join(id_classes, by = "id")

# Perform ID evaluation
evaluate(data = data_mc, target_col = "target",
         prediction_cols = class_names,
         id_col = "id",
         id_method = "mean", # alternatively: "majority"
         type = "multinomial")

# Training and evaluating a multinomial model with nnet

# Create a data frame with some predictors and a target column
class_names <- paste0("class_", 1:4)
data_for_nnet <- multiclass_probability_tibble(
    num_classes = 3, # Here, number of predictors
    num_observations = 30,
    apply_softmax = FALSE,
    FUN = rnorm,
    class_name = "predictor_") %>%
    dplyr::mutate(class = sample(
        class_names,
        size = 30,
        replace = TRUE))

# Train multinomial model using the nnet package
mn_model <- nnet::multinom(
    "class ~ predictor_1 + predictor_2 + predictor_3",
    data = data_for_nnet)

# Predict the targets in the dataset
# (we would usually use a test set instead)
predictions <- predict(mn_model, data_for_nnet,
                       type = "probs") %>%
    dplyr::as_tibble()

# Add the targets
predictions[["target"]] <- data_for_nnet[["class"]]

# Evaluate predictions
evaluate(data = predictions, target_col = "target",
         prediction_cols = class_names,
         type = "multinomial")
```

`multiclass_probability_tibble`
Generate a multiclass probability tibble

Description

Maturing

Generate a tibble with random numbers containing one column per specified class. When the softmax function is applied, the numbers become probabilities that sum to 1 rowwise.

Usage

```
multiclass_probability_tibble(num_classes, num_observations,
  apply_softmax = TRUE, FUN = runif, class_name = "class_")
```

Arguments

`num_classes`	The number of classes. Also the number of columns in the tibble.
`num_observations`	The number of observations. Also the number of rows in the tibble.
`apply_softmax`	Whether to apply the softmax function rowwise. This will transform the numbers to probabilities that sum to 1 rowwise.
`FUN`	Function for generating random numbers. The first argument must be the number of random numbers to generate, as no other arguments are supplied.
`class_name`	The prefix for the column names. The column index is appended.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

Examples

```
# Attach cvms
library(cvms)

# Create a tibble with 5 classes and 10 observations
# Apply softmax to make sure the probabilities sum to 1
multiclass_probability_tibble(num_classes = 5,
                              num_observations = 10,
                              apply_softmax = TRUE)

# Using the rnorm function to generate the random numbers
multiclass_probability_tibble(num_classes = 5,
                              num_observations = 10,
                              apply_softmax = TRUE,
                              FUN = rnorm)

# Creating a custom generator function that
# exponentiates the numbers to create more "certain" predictions
rcertain <- function(n){
    (runif(n, min = 1, max = 100)^1.4)/100
}
multiclass_probability_tibble(num_classes = 5,
                              num_observations = 10,
                              apply_softmax = TRUE,
                              FUN = rcertain)
```

`participant.scores` *Participant scores*

Description

Made-up experiment data with 10 participants and two diagnoses. Test scores for 3 sessions per participant, where participants improve their scores each session.

Format

A data frame with 30 rows and 5 variables:

participant participant identifier, 10 levels

age age of the participant, in years

diagnosis diagnosis of the participant, either 1 or 0

score test score of the participant, on a 0-100 scale

session testing session identifier, 1 to 3

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

`precomputed.formulas` *Precomputed formulas*

Description

Fixed effect combinations for model formulas with/without two- and three-way interactions. Up to eight fixed effects in total with up to five fixed effects per formula.

Format

A data frame with 259,358 rows and 5 variables:

formula_ combination of fixed effects, separated by `"+"` and `"*"`

max_interaction_size maximum interaction size in the formula, up to `3`

max_effect_frequency maximum count of an effect in the formula, e.g. the `3` A's in `"A * B + A * C + A * D"`

num_effects number of unique effects included in the formula

min_num_fixed_effects minimum number of fixed effects required to use the formula, i.e. the index in the alphabet of the last of the alphabetically ordered effects (letters) in the formula, so 4 for the formula: `"A + B + D"`

Details

Effects are represented by the first eight capital letters.

Used by `combine_predictors`.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

`reconstruct_formulas` *Reconstruct model formulas from results tibbles*

Description

Maturing

In the results tibble from cross_validate and validate, the model formulas have been split into the columns Dependent, Fixed and Random. Quickly reconstruct the model formulas from these columns.

Usage

```
reconstruct_formulas(results, topn = NULL)
```

Arguments

`results`	Data frame with results from `cross_validate()` or `validate()`. (tbl) Must contain at least the columns "`Dependent`" and "`Fixed`". For random effects the "`Random`" column should be included.
`topn`	Number of top rows to return. Simply applies head() to the results tibble.

Value

List of model formulas.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

`select_metrics` *Select columns with evaluation metrics and model definitions.*

Description

Maturing

When reporting results, we might not want all the nested tibbles and process information columns. This function selects the evaluation metrics and model formulas only.

Usage

```
select_metrics(results, include_definitions = TRUE,
  additional_includes = NULL)
```

Arguments

`results`	Results tibble from `cross_validate()` or `validate()`.
`include_definitions`	Whether to include the `Dependent`, `Fixed` and (possibly) `Random` columns. (Logical)
`additional_includes`	Names of additional columns to select. (Character)

Details

The first element in the `Family` column is used to identify the relevant columns.

Value

The results tibble with only metric and model definition columns.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

---

validate — *Validate regression models on a test set*

Description

Stable

Train gaussian or binomial models on a full training set and validate it by predicting the test/validation set. Returns results in a tibble for easy reporting, along with the trained models.

Usage

```
validate(train_data, models, test_data = NULL,
  partitions_col = ".partitions", family = "gaussian", link = NULL,
  control = NULL, REML = FALSE, cutoff = 0.5, positive = 2,
  metrics = list(), err_nc = FALSE, rm_nc = FALSE,
  parallel = FALSE, model_verbose = FALSE)
```

Arguments

`train_data`	Data Frame.
`models`	Model formulas as strings. (Character) E.g. `c("y~x","y~z")`. Can contain random effects. E.g. `c("y~x+(1|r)","y~z+(1|r)")`.
`test_data`	Data Frame. If specifying `partitions_col`, this can be `NULL`.
`partitions_col`	Name of grouping factor for identifying partitions. (Character) Rows with the value `1` in `partitions_col` are used as training set and rows with the value `2` are used as test set. N.B. Only used if `test_data` is `NULL`.
`family`	Name of family. (Character) Currently supports `"gaussian"` and `"binomial"`.
`link`	Link function. (Character) E.g. `link = "log"` with `family = "gaussian"` will use `family = gaussian(link = "log")`. See `stats::family` for available link functions.

Default link functions:
Gaussian: `'identity'`.
Binomial: `'logit'`.

`control`	Construct control structures for mixed model fitting (i.e. `lmer` and `glmer`). See `lme4::lmerControl` and `lme4::glmerControl`. N.B. Ignored if fitting `lm` or `glm` models.
`REML`	Restricted Maximum Likelihood. (Logical)
`cutoff`	Threshold for predicted classes. (Numeric) N.B. **Binomial models only**
`positive`	Level from dependent variable to predict. Either as character or level index (`1` or `2` - alphabetically). E.g. if we have the levels `"cat"` and `"dog"` and we want `"dog"` to be the positive class, we can either provide `"dog"` or `2`, as alphabetically, `"dog"` comes after `"cat"`. Used when calculating confusion matrix metrics and creating ROC curves. N.B. Only affects evaluation metrics, not the model training or returned predictions. N.B. **Binomial models only**.
`metrics`	List for enabling/disabling metrics. E.g. `list("RMSE" = FALSE)` would remove RMSE from the results, and `list("Accuracy" = TRUE)` would add the regular accuracy metric to the classification results. Default values (TRUE/FALSE) will be used for the remaining metrics available. Also accepts the string `"all"`. N.B. Currently, disabled metrics are still computed.
`err_nc`	Raise error if model does not converge. (Logical)
`rm_nc`	Remove non-converged models from output. (Logical)
`parallel`	Whether to validate the list of models in parallel. (Logical) Remember to register a parallel backend first. E.g. with `doParallel::registerDoParallel`.
`model_verbose`	Message name of used model function on each iteration. (Logical)

Details

Packages used:

Models:
Gaussian: stats::lm, `lme4::lmer`
Binomial: `stats::glm`, `lme4::glmer`

Results: Gaussian:
r2m : `MuMIn::r.squaredGLMM`
r2c : `MuMIn::r.squaredGLMM`
AIC : `stats::AIC`
AICc : `MuMIn::AICc`
BIC : `stats::BIC`
Binomial:
Confusion matrix: `caret::confusionMatrix`
ROC: `pROC::roc`
MCC: `mltools::mcc`

Value

List containing tbl (tibble) with results and the trained model object. The tibble contains:

Shared across families:

A nested tibble with **coefficients** of the models from all iterations.

Count of **convergence warnings**. Consider discarding models that did not converge on all iterations. Note: you might still see results, but these should be taken with a grain of salt!

Count of **other warnings**. These are warnings without keywords such as "convergence".

Count of **Singular Fit messages**. See `?lme4::isSingular` for more information.

Nested tibble with the **warnings and messages** caught for each model.

Specified **family**.

Specified **link** function.

Name of **dependent** variable.

Names of **fixed** effects.

Names of **random** effects, if any.

———————————————————-

Gaussian Results:

———————————————————-

RMSE, **MAE**, **r2m**, **r2c**, **AIC**, **AICc**, and **BIC**.

A nested tibble with the **predictions** and targets.

———————————————————-

Binomial Results:

———————————————————-

Based on predictions of the test set, a confusion matrix and ROC curve are used to get the following:

ROC:

AUC, **Lower CI**, and **Upper CI**

Confusion Matrix:

Balanced Accuracy, **F1**, **Sensitivity**, **Specificity**, **Positive Prediction Value**, **Negative Prediction Value**, **Kappa**, **Detection Rate**, **Detection Prevalence**, **Prevalence**, and **MCC** (Matthews correlation coefficient).

Other available metrics (disabled by default, see `metrics`): **Accuracy**.

Also includes:

A tibble with **predictions**, predicted classes (depends on `cutoff`), and the targets.

A tibble with the sensativities and specificities from the **ROC** curve.

Author(s)

Ludvig Renbo Olsen, `<r-pkgs@ludvigolsen.dk>`

See Also

Other validation functions: `cross_validate_fn`, `cross_validate`

Examples

```
# Attach packages
library(cvms)
library(groupdata2) # partition()
library(dplyr) # %>% arrange()

# Data is part of cvms
data <- participant.scores

# Set seed for reproducibility
set.seed(7)

# Partition data
# Keep as single data frame
# We could also have fed validate() separate train and test sets.
data_partitioned <- partition(data,
                              p = 0.7,
                              cat_col = 'diagnosis',
                              id_col = 'participant',
                              list_out=FALSE) %>%
    arrange(.partitions)

# Validate a model

# Gaussian
validate(data_partitioned,
         models = "score~diagnosis",
         partitions_col = '.partitions',
         family='gaussian',
         REML = FALSE)

# Binomial
validate(data_partitioned,
         models = "diagnosis~score",
         partitions_col = '.partitions',
         family='binomial')

# Use non-default link functions

validate(data_partitioned,
         models = "score~diagnosis",
         partitions_col = '.partitions',
         family = 'gaussian',
         link = 'log',
         REML = FALSE)

## Feed separate train and test sets

# Partition data to list of data frames
# The first data frame will be train (70% of the data)
# The second will be test (30% of the data)
data_partitioned <- partition(data, p = 0.7,
                              cat_col = 'diagnosis',
                              id_col = 'participant',
                              list_out=TRUE)
train_data <- data_partitioned[[1]]
```

```
test_data <- data_partitioned[[2]]

# Validate a model

# Gaussian
validate(train_data,
         test_data = test_data,
         models = "score~diagnosis",
         family='gaussian',
         REML = FALSE)
```

Index