Package 'predRupdate'

August 23, 2024

Title Prediction Model Validation and Updating

Version 0.2.0

Description Evaluate the predictive performance of an existing (i.e. previously developed) prediction/ prognostic model given relevant information about the existing prediction model (e.g. coefficients) and a new dataset. Provides a range of model updating methods that help tailor the existing model to the new dataset; see Su et al. (2018) <doi:10.1177/0962280215626466>. Techniques to aggregate multiple existing prediction models on the new data are also provided; see Debray et al. (2014) <doi:10.1002/sim.6080> and Martin et al. (2018) <doi:10.1002/sim.7586>).

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

URL `https://github.com/GlenMartin31/predRupdate`, `https://glenmartin31.github.io/predRupdate/`

BugReports `https://github.com/GlenMartin31/predRupdate/issues`

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports utils, stats, survival, pROC, ggplot2, rlang, ggpubr

Depends R (>= 2.10)

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Glen P. Martin [aut, cre, cph] (<`https://orcid.org/0000-0002-3410-9472`>), David Jenkins [aut, cph], Matthew Sperrin [aut, cph]

Maintainer Glen P. Martin <`glen.martin31@gmail.com`>

Repository CRAN

Date/Publication 2024-08-23 14:20:02 UTC

Contents

`dummy_vars` *Create dummy variables for all categorical/factor variables in a data.frame*

Description

Create dummy/indicator variables for all categorical variables in a data.frame. Can be used as a pre-processing step before calling other functions within the package.

Usage

```
dummy_vars(df)
```

Arguments

`df`	a data.frame on which to make dummy variables for each categorical/factor variable, based on contrasts.

Value

a data.frame matching `df` but where each categorical variable in `df` is replaced with indicator variables. All combinations of the indicator/dummy variable are returned. Naming convention of the new dummy variables is variable_level. For example, a factor variable in `df` named "colour" with levels "red", "green" and "purple" will be replaced with three columns (the new dummy variables), named colour_red, colour_green and colour_purple.

See Also

`pred_input_info`

Examples

```
dummy_vars(data.frame("Colour" = factor(sample(c("red",
                                                  "azure",
                                                  "green",
                                                  "white"),
                                               500,
                                               replace = TRUE))))
```

`inv_logit` *Apply the inverse logit function to an input*

Description

`inv_logit` applies the inverse-logit transformation (expit/ logistic function) to convert a vector of values between -Inf and Inf, to values between 0 and 1. Used to convert the linear predictor of a logistic regression model into a probability.

Usage

```
inv_logit(x)
```

Arguments

`x`	Numeric vector with values between -Inf and Inf.

Value

Numeric vector of probabilities (i.e. values between 0 and 1)

See Also

`logit`

Examples

```
inv_logit(-2)
inv_logit(c(-2,-1,0,1,2))
```

logit *Apply a logit transformation to an input*

Description

`logit` applies the logit transformation to convert a vector of values between 0 and 1, to values between -Inf and Inf. Used to convert a probability from a logistic regression model onto the linear predictor scale.

Usage

```
logit(p)
```

Arguments

`p`	Numeric vector of probabilities (i.e. values between 0 and 1) that will be transformed.

Value

A numeric vector, with values between -Inf and Inf

See Also

`inv_logit`

Examples

```
logit(0.5)
logit(c(0.1, 0.2, 0.3))
```

map_newdata *Map new data to a predinfo object*

Description

This function takes a `predinfo` object and applies (maps) a new data to this object to check there is consistency between the two. This function is not usually called directly, but rather within other functions within the package, such as `pred_predict`.

Usage

```
map_newdata(
  x,
  new_data,
  binary_outcome = NULL,
  survival_time = NULL,
  event_indicator = NULL
)
```

Arguments

`x`	an object of class "predinfo".
`new_data`	data.frame upon which the prediction model should be applied (for subsequent validation/model updating/model aggregation).
`binary_outcome`	Character variable giving the name of the column in `new_data` that represents the observed binary outcomes (should be coded 0 and 1 for non-event and event, respectively). Only relevant for `model_type`="logistic"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any outcomes.
`survival_time`	Character variable giving the name of the column in `new_data` that represents the observed survival times. Only relevant for `model_type`="survival"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any survival outcomes.
`event_indicator`	Character variable giving the name of the column in `new_data` that represents the observed survival indicator (1 for event, 0 for censoring). Only relevant for `model_type`="survival"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any survival outcomes.

Details

This function maps a new dataset onto a `pred_info` object. The new dataset might be a validation dataset (to test the performance of the existing prediction model) and/or it might be the dataset on which one wishes to apply model updating methods to revise the model. In any case, this should be specified in `new_data` as a data.frame. Each row should be an observation (e.g. patient) and each variable/column should be a predictor variable. The predictor variables need to include (as a minimum) all of the predictor variables that are included in the existing prediction model (i.e., each of the variable names supplied to `pred_input_info`, through the `model_info` parameter, must match the name of a variables in `new_data`).

Any factor variables within `new_data` must be converted to dummy (0/1) variables before calling this function. `dummy_vars` can help with this.

`binary_outcome`, `survival_time` and `event_indicator` are used to specify the outcome variable(s) within `new_data`, if relevant (use `binary_outcome` if `model_type` = "logistic", or use `survival_time` and `event_indicator` if `model_type` = "survival"). For example, if validating an existing model, then these inputs specify the columns of `new_data` that will be used for assessing predictive performance of the predictions in the validation dataset. If `new_data` does not contain outcomes, then leave these inputs to the default of `NULL`.

Value

Returns a list of the predinfo object, the new_data, and outcomes.

Examples

```
#as above, this function is not usually called directly, but an example of
#such use is:
model1 <- pred_input_info(model_type = "logistic",
                          model_info = SYNPM$Existing_logistic_models[1,])
map_newdata(x = model1,
            new_data = SYNPM$ValidationData[1:10,],
            binary_outcome = "Y")
```

`pred_input_info` *Input information about an existing prediction model*

Description

Input coefficient information about one or multiple existing prediction model(s), for use in other functions in the package.

Usage

```
pred_input_info(
  model_type = c("logistic", "survival"),
  model_info,
  cum_hazard = NULL
)
```

Arguments

`model_type`	specifies the type of model that the existing prediction model is based on; possible options are: • `"logistic"` indicates that the existing model was based on a logistic regression model (default) • `"survival"` indicates that the existing model was based on a survival regression model If multiple models are being entered, then all models need to be of the same type - otherwise call function multiple times for each type of model.
`model_info`	a data.frame that contains the coefficients of the existing prediction model(s). Each column should be a predictor variable (with the name of the column being the name of the predictor variable), with the values being the coefficients, taken exactly as published from the existing prediction model(s). Multiple existing prediction models should be specified by entering multiple rows. If a predictor variable is not present in a given model then enter that cell of the data.frame as NA. See examples.

`cum_hazard`	A data.frame with two columns: (1) time, and (2) estimated cumulative baseline hazard at that time. The first column (time) should be named 'time' and the second (cumulative baseline hazard) should be named 'hazard'. Only relevant if `model_type` is "survival"; leave as NULL otherwise. If multiple existing models entered, and model_type = survival, then `cum_hazard` should be supplied as list of length equal to number of models.

Details

This function will structure the relevant information about one or more existing prediction model(s) into a standardised format, such that it can be used within other functions in the package.

First, the existing prediction model(s) will have a functional form (i.e. the linear predictor of the model); this will be taken as being a linear combination of the variables specified by the columns of `model_info`.

Second, each of the predictor variables of the existing prediction model(s) will have a published coefficient (e.g. log-odds-ratio or log-hazard-ratio), which should each be given as the values of `model_info`. If entering information about multiple existing prediction models, then `model_info` will contain multiple rows (one per existing model). Here, if a given model does not contain a predictor variable that is included in another model, then set as NA; see examples of this below.

In the case of `model_type` = "logistic", then `model_info` must contain a column named as "Intercept", which gives the intercept coefficient of each of the existing logistic regression models (taken exactly as previously published); this should be the first column of `model_info`.

If `model_type` = "survival", then the baseline cumulative hazard of the model(s) can be specified in `cum_hazard`. If the baseline cumulative hazard of the existing survival model is not available, then leave as NULL; this will limit any validation metrics that can be calculated.

Note, the column names of `model_info` should match columns in any new data that the existing model(s) will be applied to (i.e. any new data that will be provided to other functions within the package should have corresponding predictor variables entered through `model_info`). See `pred_predict`, `pred_validate`, `pred_update` and `pred_stacked_regression` for more information.

Value

`pred_input_info` returns an object of class "`predinfo`", with child classes per `model_type`. This is a standardised format, such that it can be used with other functions in the package. An object of class "`predinfo`" is a list containing the following components:

- M = the number of existing models that information has been entered about
- model_type = this is the type of model that the existing prediction model is based upon ("logistic" or "survival")
- coefs = this is the set of (previously estimated) coefficients for each predictor variable
- coef_names = gives the names of each predictor variable
- formula = this is the functional form of the model's linear predictor
- cum_hazard = if supplied, this is the cumulative baseline hazard of the existing model(s)

Examples

```
#Example 1 - logistic regression existing model
# create a data.frame of the model coefficients, with columns being variables
coefs_table <- data.frame("Intercept" = -3.4,
                          "SexM" = 0.306,
                          "Smoking_Status" = 0.628,
                          "Diabetes" = 0.499,
                          "CKD" = 0.538)
#pass this into pred_input_info()
Existing_Logistic_Model <- pred_input_info(model_type = "logistic",
                                           model_info = coefs_table)
summary(Existing_Logistic_Model)

#Example 2 - survival model example; uses an example dataset within the
#             package.
pred_input_info(model_type = "survival",
                model_info = SYNPM$Existing_TTE_models[2,],
                cum_hazard = SYNPM$TTE_mod2_baseline)

#Example 3 - Input information about multiple models
summary(pred_input_info(model_type = "logistic",
                        model_info = SYNPM$Existing_logistic_models))
```

`pred_predict` *Make predictions from an existing prediction model*

Description

Use an existing prediction model to estimate predicted risks of the outcome for each observation in a new dataset.

Usage

```
pred_predict(
  x,
  new_data,
  binary_outcome = NULL,
  survival_time = NULL,
  event_indicator = NULL,
  time_horizon = NULL
)
```

Arguments

`x`	an object of class "`predinfo`" produced by calling `pred_input_info`.
`new_data`	data.frame upon which predictions are obtained using the prediction model.

`binary_outcome`	Character variable giving the name of the column in `new_data` that represents the observed binary outcomes (should be coded 0 and 1 for non-event and event, respectively). Only relevant for `model_type`="logistic"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any outcomes.
`survival_time`	Character variable giving the name of the column in `new_data` that represents the observed survival times. Only relevant for `model_type`="survival"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any survival outcomes.
`event_indicator`	Character variable giving the name of the column in `new_data` that represents the observed survival indicator (1 for event, 0 for censoring). Only relevant for `model_type`="survival"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any survival outcomes.
`time_horizon`	for survival models, an integer giving the time horizon (post baseline) at which a prediction is required (i.e. the t at which P(T<t) should be estimated). Currently, this must match a time in x$cum_hazard. If left as NULL, no predicted risks will be returned, just the linear predictor.

Details

This function takes the relevant information about the existing prediction model (as supplied by calling `pred_input_info`), and returns the linear predictor and predicted risks for each individual/observation in `new_data`.

If the existing prediction model is based on logistic regression (i.e., if x$model_type == "logistic"), the predicted risks will be the predicted probability of the binary outcome conditional on the predictor variables in the new data (i.e., P(Y=1 | X)). If the existing prediction model is based on a time-to-event/survival model (i.e., if x$model_type == "survival"), the predicted risks can only be calculated if a baseline cumulative hazard is provided; in this case, the predicted risks will be one minus the survival probability (i.e., 1 - S(T>time horizon | X)).

`new_data` should be a data.frame, where each row should be an observation (e.g. patient) and each variable/column should be a predictor variable. The predictor variables need to include (as a minimum) all of the predictor variables that are included in the existing prediction model (i.e., each of the variable names supplied to `pred_input_info`, through the `model_info` parameter, must match the name of a variables in `new_data`). Any factor variables within `new_data` must be converted to dummy (0/1) variables before calling this function. `dummy_vars` can help with this. See examples.

`binary_outcome`, `survival_time` and `event_indicator` are used to specify the outcome variable(s) within `new_data` (use `binary_outcome` if `x$model_type` = "logistic", or use `survival_time` and `event_indicator` if `x$model_type` = "survival").

Value

`pred_predict` returns a list containing the following components:

- LinearPredictor = the linear predictor for each observation in the new data (i.e., the linear combination of the models predictor variables and their corresponding coefficients)
- PredictedRisk = the predicted risk for each observation in the new data

- TimeHorizon = for survival models, an integer giving the time horizon at which a prediction is made
- Outcomes = vector of outcomes/endpoints (if available).

See Also

pred_input_info

Examples

```
#Example 1 - logistic regression existing model - shows handling of factor variables
coefs_table <- data.frame("Intercept" = -3.4,
                          "Sex_M" = 0.306,
                          "Smoking_Status" = 0.628)
existing_Logistic_Model <- pred_input_info(model_type = "logistic",
                                           model_info = coefs_table)
new_df <- data.frame("Sex" = as.factor(c("M", "F", "M", "M", "F", "F", "M")),
                     "Smoking_Status" = c(1, 0, 0, 1, 1, 0, 1))
#new_df has a factor variable, so needs indicator variables creating before pred_predict:
new_df_indicators <- dummy_vars(new_df)
pred_predict(x = existing_Logistic_Model,
             new_data = new_df_indicators)

#Example 2 - survival model example; uses an example dataset within the
#             package. Multiple existing models
model2 <- pred_input_info(model_type = "survival",
                          model_info = SYNPM$Existing_TTE_models,
                          cum_hazard = list(SYNPM$TTE_mod1_baseline,
                                            SYNPM$TTE_mod2_baseline,
                                            SYNPM$TTE_mod3_baseline))
pred_predict(x = model2,
             new_data = SYNPM$ValidationData[1:10,],
             survival_time = "ETime",
             event_indicator = "Status",
             time_horizon = 5)
```

---

pred_stacked_regression *Perform Stacked Regression on Existing Prediction Models*

Description

This function takes a set of existing prediction models, and uses the new dataset to combine/aggregate them into a single 'meta-model', as described in Debray et al. 2014.

Usage

```
pred_stacked_regression(
  x,
  positivity_constraint = FALSE,
  new_data,
  binary_outcome = NULL,
  survival_time = NULL,
  event_indicator = NULL
)
```

Arguments

`x`	an object of class "`predinfo`" produced by calling `pred_input_info` containing information on at least two existing prediction models.
`positivity_constraint`	TRUE/FALSE denoting if the weights within the stacked regression model should be constrained to be non-negative (TRUE) or should be allowed to take any value (FALSE). See details.
`new_data`	data.frame upon which the prediction models should be aggregated.
`binary_outcome`	Character variable giving the name of the column in `new_data` that represents the observed binary outcomes (should be coded 0 and 1 for non-event and event, respectively). Only relevant for `model_type`="logistic"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any outcomes.
`survival_time`	Character variable giving the name of the column in `new_data` that represents the observed survival times. Only relevant for `x$model_type`="survival"; leave as `NULL` otherwise.
`event_indicator`	Character variable giving the name of the column in `new_data` that represents the observed survival indicator (1 for event, 0 for censoring). Only relevant for `x$model_type`="survival"; leave as `NULL` otherwise.

Details

This function takes a set of (previously estimated) prediction models that were each originally developed for the same prediction task, and pool/aggregate these into a single prediction model (meta-model) using stacked regression based on new data (data not used to develop any of the existing models). The methodological details can be found in Debray et al. 2014.

Given that the existing models are likely to be highly co-linear (since they were each developed for the same prediction task), it has been suggested to impose a positivity constraint on the weights of the stacked regression model (Debray et al. 2014.). If `positivity_constraint` is set to TRUE, then the stacked regression model will be estimated by optimising the (log-)likelihood using bound constrained optimization ("L-BFGS-B"). This is currently only implemented for logistic regression models (i.e., if `x$model_type`="logistic"). For survival models, positivity_constraint = FALSE.

`new_data` should be a data.frame, where each row should be an observation (e.g. patient) and each variable/column should be a predictor variable. The predictor variables need to include (as a minimum) all of the predictor variables that are included in the existing prediction models (i.e., each of the variable names supplied to `pred_input_info`, through the `model_info` parameter, must match the name of a variables in `new_data`).

Any factor variables within `new_data` must be converted to dummy (0/1) variables before calling this function. `dummy_vars` can help with this. See `pred_predict` for examples.

`binary_outcome`, `survival_time` and `event_indicator` are used to specify the outcome variable(s) within `new_data` (use `binary_outcome` if `x$model_type` = "logistic", or use `survival_time` and `event_indicator` if `x$model_type` = "survival").

Value

A object of class "predSR". This is the same as that detailed in `pred_input_info`, with the added element containing the estimates of the meta-model obtained by stacked regression.

References

Debray, T.P., Koffijberg, H., Nieboer, D., Vergouwe, Y., Steyerberg, E.W. and Moons, K.G. (2014), Meta-analysis and aggregation of multiple published prediction models. *Statistics in Medicine*, 33: 2341-2362

See Also

`pred_input_info`

Examples

```
LogisticModels <- pred_input_info(model_type = "logistic",
                                  model_info = SYNPM$Existing_logistic_models)
SR <- pred_stacked_regression(x = LogisticModels,
                              new_data = SYNPM$ValidationData,
                              binary_outcome = "Y")
summary(SR)

#Survival model example:
TTModels <- pred_input_info(model_type = "survival",
                            model_info = SYNPM$Existing_TTE_models,
                            cum_hazard = list(SYNPM$TTE_mod1_baseline,
                                              SYNPM$TTE_mod2_baseline,
                                              SYNPM$TTE_mod3_baseline))
SR <- pred_stacked_regression(x = TTModels,
                              new_data = SYNPM$ValidationData,
                              survival_time = "ETime",
                              event_indicator = "Status")
summary(SR)
```

`pred_update` *Perform Model Updating on an Existing Prediction Model*

Description

This function takes an existing (previously developed) prediction model and applies various model updating methods to tailor/adapt it to a new dataset. Various levels of updating are possible, ranging from model re-calibration to model refit.

Usage

```
pred_update(
  x,
  update_type = c("intercept_update", "recalibration", "refit"),
  new_data,
  binary_outcome = NULL,
  survival_time = NULL,
  event_indicator = NULL
)
```

Arguments

`x`	an object of class "`predinfo`" produced by calling `pred_input_info` containing information on exactly one existing prediction model.
`update_type`	character variable specifying the level of updating that is required.
`new_data`	data.frame upon which the prediction models should be updated.
`binary_outcome`	Character variable giving the name of the column in `new_data` that represents the observed binary outcomes (should be coded 0 and 1 for non-event and event, respectively). Only relevant for `model_type`="logistic"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any outcomes.
`survival_time`	Character variable giving the name of the column in `new_data` that represents the observed survival times. Only relevant for `x$model_type`="survival"; leave as `NULL` otherwise.
`event_indicator`	Character variable giving the name of the column in `new_data` that represents the observed survival indicator (1 for event, 0 for censoring). Only relevant for `x$model_type`="survival"; leave as `NULL` otherwise.

Details

This function takes a single existing (previously estimated) prediction model, and apply various model discrete model updating methods (see Su et al. 2018) to tailor the model to a new dataset.

The type of updating method is selected with the `update_type` parameter, with options: "intercept_update", "recalibration" and "refit". "intercept_update" corrects the overall calibration-in-the-large of the model, through altering the model intercept (or baseline hazard) to suit the new dataset.

This is achieved by fitting a logistic model (if the existing model is of type logistic) or time-to-event model (if the existing model if of type survival) to the new dataset, with the linear predictor as the only covariate, with the coefficient fixed at unity (i.e. as an offset). "recalibration" corrects the calibration-in-the-large and any under/over-fitting, by fitting a logistic model (if the existing model is of type logistic) or time-to-event model (if the existing model if of type survival) to the new dataset, with the linear predictor as the only covariate. Finally, "refit" takes the original model structure and re-estimates all coefficients; this has the effect as re-developing the original model in the new data.

`new_data` should be a data.frame, where each row should be an observation (e.g. patient) and each variable/column should be a predictor variable. The predictor variables need to include (as a minimum) all of the predictor variables that are included in the existing prediction model (i.e., each of the variable names supplied to `pred_input_info`, through the `model_info` parameter, must match the name of a variables in `new_data`).

Any factor variables within `new_data` must be converted to dummy (0/1) variables before calling this function. `dummy_vars` can help with this. See `pred_predict` for examples.

`binary_outcome`, `survival_time` and `event_indicator` are used to specify the outcome variable(s) within `new_data` (use `binary_outcome` if `x$model_type` = "logistic", or use `survival_time` and `event_indicator` if `x$model_type` = "survival").

Value

A object of class "`predUpdate`". This is the same as that detailed in `pred_input_info`, with the added element containing the estimates of the model updating and the update type.

References

Su TL, Jaki T, Hickey GL, Buchan I, Sperrin M. A review of statistical updating methods for clinical prediction models. *Stat Methods Med Res*. 2018 Jan;27(1):185-197. doi: 10.1177/0962280215626466.

See Also

`pred_input_info`

Examples

```
#Example 1 - update time-to-event model by updating the baseline hazard in new dataset
model1 <- pred_input_info(model_type = "survival",
                          model_info = SYNPM$Existing_TTE_models[1,],
                          cum_hazard = SYNPM$TTE_mod1_baseline)
recalibrated_model1 <- pred_update(x = model1,
                                   update_type = "intercept_update",
                                   new_data = SYNPM$ValidationData,
                                   survival_time = "ETime",
                                   event_indicator = "Status")
summary(recalibrated_model1)
```

pred_validate *Validate an existing prediction*

Description

Validate an existing prediction model, to calculate the predictive performance against a new (validation) dataset.

Usage

```
pred_validate(
  x,
  new_data,
  binary_outcome = NULL,
  survival_time = NULL,
  event_indicator = NULL,
  time_horizon = NULL,
  level = 0.95,
  cal_plot = TRUE,
  ...
)
```

Arguments

`x`	an object of class "`predinfo`" produced by calling `pred_input_info`.
`new_data`	data.frame upon which the prediction model should be evaluated.
`binary_outcome`	Character variable giving the name of the column in `new_data` that represents the observed binary outcomes (should be coded 0 and 1 for non-event and event, respectively). Only relevant for `model_type`="logistic"; leave as `NULL` otherwise. Leave as `NULL` if `new_data` does not contain any outcomes.
`survival_time`	Character variable giving the name of the column in `new_data` that represents the observed survival times. Only relevant for `x$model_type`="survival"; leave as `NULL` otherwise.
`event_indicator`	Character variable giving the name of the column in `new_data` that represents the observed survival indicator (1 for event, 0 for censoring). Only relevant for `x$model_type`="survival"; leave as `NULL` otherwise.
`time_horizon`	for survival models, an integer giving the time horizon (post baseline) at which a prediction is required. Currently, this must match a time in x$cum_hazard.
`level`	the confidence level required for all performance metrics. Defaults at 95%. Must be a value between 0 and 1.
`cal_plot`	indicate if a flexible calibration plot should be produced (TRUE) or not (FALSE).
`...`	further plotting arguments for the calibration plot. See Details below.

Details

This function takes an existing prediction model formatted according to pred_input_info, and calculates measures of predictive performance on new data (e.g., within an external validation study). The information about the existing prediction model should first be inputted by calling pred_input_info, before passing the resulting object to pred_validate.

new_data should be a data.frame, where each row should be an observation (e.g. patient) and each variable/column should be a predictor variable. The predictor variables need to include (as a minimum) all of the predictor variables that are included in the existing prediction model (i.e., each of the variable names supplied to pred_input_info, through the model_info parameter, must match the name of a variables in new_data).

Any factor variables within new_data must be converted to dummy (0/1) variables before calling this function. dummy_vars can help with this. See pred_predict for examples.

binary_outcome, survival_time and event_indicator are used to specify the outcome variable(s) within new_data (use binary_outcome if x$model_type = "logistic", or use survival_time and event_indicator if x$model_type = "survival").

In the case of validating a logistic regression model, this function assesses the predictive performance of the predicted risks against an observed binary outcome. Various metrics of calibration (agreement between the observed risk and the predicted risks, across the full risk range) and discrimination (ability of the model to distinguish between those who develop the outcome and those who do not) are calculated. For calibration, the observed-to-expected ratio, calibration intercept and calibration slopes are estimated. The calibration intercept is estimated by fitting a logistic regression model to the observed binary outcomes, with the linear predictor of the model as an offset. For calibration slope, a logistic regression model is fit to the observed binary outcome with the linear predictor from the model as the only covariate. For discrimination, the function estimates the area under the receiver operating characteristic curve (AUC). Various other metrics are also calculated to assess overall accuracy (Brier score, Cox-Snell R2).

In the case of validating a survival prediction model, this function assesses the predictive performance of the linear predictor and (optionally) the predicted event probabilities at a fixed time horizon against an observed time-to-event outcome. Various metrics of calibration and discrimination are calculated. For calibration, the observed-to-expected ratio at the specified time_horizon (if predicted risks are available through specification of x$cum_hazard) and calibration slope are produced. For discrimination, Harrell's C-statistic is calculated.

For both model types, a flexible calibration plot is produced (for survival models, the cumulative baseline hazard must be available in the predinfo object, x$cum_hazard). Specify parameter cal_plot to indicate whether a calibration plot should be produced (TRUE), or not (FALSE). The calibration plot is produced by regressing the observed outcomes against a cubic spline of the logit of predicted risks (for a logistic model) or the complementary log-log of the predicted risks (for a survival model). Users can specify the following additional parameters to pred_validate to modify the calibration plot:

- xlim as a numeric vector of length 2, giving the lower and upper range of the x-axis scale - defaults at 0 and 1. Changes here should match changes to the ylim such that the plot remains 'square'.
- ylim as a numeric vector of length 2, giving the lower and upper range of the y-axis scale - defaults at 0 and 1. Changes here should match changes to the xlim such that the plot remains 'square'.

- `xlab` string giving the x-axis label. Defaults as "Predicted Probability".
- `ylab` string giving the x-axis label. Defaults as "Observed Probability".
- `pred_rug` TRUE/FALSE of whether a 'rug' should be placed along the x-axis of the calibration plot showing the distribution of predicted risks. Defaults as FALSE in favour of examining the box-plot/violin plot that is produced.
- `cal_plot_n_sample` numeric value (less than nrow(new_data)) giving a random subset of observations to render the calibration plot over. The calibration plot is always created using all data, but for rendering speed in large datasets, it can sometimes be useful to render the plot over a smaller (random) subset of observations. Final (e.g. publication-ready) plots should always show the full plot, so a warning is created if users enter a value of cal_plot_n_sample.

Value

`pred_validate` returns an object of class "`predvalidate`", with child classes per `model_type`. This is a list of performance metrics, estimated by applying the existing prediction model to the new_data. An object of class "`predvalidate`" is a list containing relevant calibration and discrimination measures. For logistic regression models, this will include observed:expected ratio, calibration-intercept, calibration slope, area under the ROC curve, R-squared, and Brier Score. For survival models, this will include observed:expected ratio (if `cum_hazard` is provided to `x`), calibration slope, and Harrell's C-statistic. Optionally, a flexible calibration plot is also produced, along with a box-plot and violin plot of the predicted risk distribution.

The `summary` function can be used to extract and print summary performance results (calibration and discrimination metrics). The graphical assessments of performance can be extracted using `plot`.

See Also

`pred_input_info`

Examples

```
#Example 1 - multiple existing model, with outcome specified; uses
#            an example dataset within the package
model1 <- pred_input_info(model_type = "logistic",
                          model_info = SYNPM$Existing_logistic_models)
val_results <- pred_validate(x = model1,
                             new_data = SYNPM$ValidationData,
                             binary_outcome = "Y",
                             cal_plot = FALSE)
summary(val_results)
```

`pred_val_probs` *Validate Predicted Probabilities*

Description

This function is included for situations where one has a vector of predicted probabilities from a model and a vector of observed binary outcomes that we wish to validate the predictions against. See `pred_validate` for the main validation function of this package.

Usage

```
pred_val_probs(binary_outcome, Prob, cal_plot = TRUE, level = 0.95, ...)
```

Arguments

`binary_outcome`	vector of binary outcomes (coded as 1 if outcome happened, and 0 otherwise). Must be of same length as Prob
`Prob`	vector of predicted probabilities. Must be of same length of binary_outcome.
`cal_plot`	indicate if a flexible calibration plot should be produced (TRUE) or not (FALSE).
`level`	the confidence level required for all performance metrics. Defaults at 95%. Must be a value between 0 and 1.
`...`	further plotting arguments for the calibration plot. See Details below.

Details

This function takes a vector of observed binary outcomes, and a corresponding vector of predicted risks (e.g. from a logistic regression CPM), and calculates measures of predictive performance. The function is intended as a standalone way of validating predicted risks against binary outcomes outside of the usual pred_input_info() -> pred_validate() package workflow. See `pred_validate` for the main validation function of this package.

Various metrics of calibration (agreement between the observed risk and the predicted risks, across the full risk range) and discrimination (ability of the model to distinguish between those who develop the outcome and those who do not) are calculated. For calibration, the observed-to-expected ratio, calibration intercept and calibration slopes are estimated. The calibration intercept is estimated by fitting a logistic regression model to the observed binary outcomes, with the linear predictor of the model as an offset. For calibration slope, a logistic regression model is fit to the observed binary outcome with the linear predictor from the model as the only covariate. For discrimination, the function estimates the area under the receiver operating characteristic curve (AUC). Various other metrics are also calculated to assess overall accuracy (Brier score, Cox-Snell R2).

A flexible calibration plot is produced. Specify parameter `cal_plot` to indicate whether a calibration plot should be produced (TRUE), or not (FALSE). See `pred_validate` for details on this plot, and details of optional plotting arguments.

Value

An object of class "`predvalidate`", which is a list containing relevant calibration and discrimination measures. See `pred_validate` for details.

See Also

`pred_input_info`, `pred_validate`

Examples

```
# simulate some data for purposes of example illustration
set.seed(1234)
x1 <- rnorm(2000)
LP <- -2 + (0.5*x1)
PR <- 1/(1+exp(-LP))
y <- rbinom(2000, 1, PR)

#fit hypothetical model to the simulated data
mod <- glm(y[1:1000] ~ x1[1:1000], family = binomial(link="logit"))

#obtain the predicted risks from the model
pred_risk <- predict(mod, type = "response",
                     newdata = data.frame("x1" = x1[1001:2000]))

#Use pred_val_probs to validate the predicted risks against the
#observed outcomes
summary(pred_val_probs(binary_outcome = y[1001:2000],
                       Prob = pred_risk,
                       cal_plot = FALSE))
```

`SYNPM` *SYNthetic Prediction Models (SYNPM) and Validation dataset*

Description

A list containing: (1) information on some (synthetic) existing prediction models (representing those available/published, which we want to validate in another independent dataset); and (2) a synthetic dataset that we wish to validate/update the models on.

Usage

```
SYNPM
```

Format

A list with six elements.

1. The first element is a data frame with the information about three existing binary (logistic regression) models for a binary outcome at one year
2. The second element is a data frame with the information about three existing time-to-event (Cox) models for the time-to-event outcome
3. The third, fourth and fifth elements are the cumulative baseline hazard information for the three time-to-event model
4. The sixth element is the (synthetic) validation dataset on which we want to validate the existing models. The dataset has 20000 rows and 8 variables:

   Age The age of the individual at baseline

SexM The sex of the individual (1 = male; 0 = female)

Smoking_Status Indicates whether the individual was or is a smoker (1=previous/ current smoker, 0=non-smoker)

Diabetes Indicates whether the individual has diabetes (1=diabetic, 0=not diabetic)

Creatinine The Creatinine value for the individual (mg/dL)

ETime The time from baseline until either the event or censoring

Status Indicator of whether the patient experienced the event or was censored at ETime

Y Binary indicator of whether the individual experienced the event by 1 time-unit

Source

Simulated Data; see `https://github.com/GlenMartin31/predRupdate`

Index