Package 'EQUALPrognosis'

May 7, 2026

Title Analysing Prognostic Studies

Version 0.1.3

Date 2026-03-20

Author Kurinchi Gurusamy [aut, cre]

Maintainer Kurinchi Gurusamy <k.gurusamy@ucl.ac.uk>

Depends stats, ggplot2, survival

Imports base64enc, CalibrationCurves, mime, predtools, pROC, stringr

Description Functions that help with analysis of prognostic study data. This allows users with little experience of developing models to develop models and assess the performance of the prognostic models. This also summarises the information, so the performance of multiple models can be displayed simultaneously. This minor update fixes issues related to memory requirements with large number of simulations and deals with situations when there is overfitting of data. Gurusamy, K (2026)<https://github.com/kurinchi2k/EQUALPrognosis>.

License GPL (>= 3)

Encoding UTF-8

URL https://sites.google.com/view/equal-group/home

RoxygenNote 7.3.3

NeedsCompilation no

Repository CRAN

Date/Publication 2026-03-20 11:00:02 UTC

Contents

`calculate_actual_predicted` *Calculate actual and predicted values*

Description

This takes the datasets, prepared using the `prepare_datasets` function and the input parameters described below, creates a model, runs the model, and predicts the outcome.

Usage

```
calculate_actual_predicted(prepared_datasets, outcome_name, outcome_type,
outcome_time, outcome_count, develop_model, predetermined_model_text,
mandatory_predictors, optional_predictors, mandatory_interactions,
optional_interactions, model_threshold_method, scoring_system,
predetermined_threshold, higher_values_event, each_simulation,
bootstrap_sample, verbose)
```

Arguments

`prepared_datasets`
: Datasets prepared using the `prepare_datasets`.

`outcome_name`
: Name of the columnn that contains the outcome data. This must be a column name in the *'df'* provided as input.

`outcome_type`
: One of *'binary'*, *'time-to-event'*, *'quantitative'*. Count outcomes are included in *'quantitative'* outcome type and can be differentiated from continuous outcomes by specifying outcome_count as *TRUE*. Please see examples below.

`outcome_time`
: The name of the column that provides the follow-up time. This is applicable only for *'time-to-event'* outcome. For other outcome types, enter *NA*.

`outcome_count`
: *TRUE* if the outcome was a count outcome and *FALSE* otherwise.

`develop_model`
: *TRUE*, if you want to develop a model; *FALSE*, if you want to use a scoring system with a predetermined threshold (if applicable).

`predetermined_model_text`
: You can create the model text from the mandatory and optional predictors and interactions or finer control of the model, you can provide the model text directly.

`mandatory_predictors`
: Predictors that must be included in the model. These should be provided even if you provide the 'predetermined_model_text'.

`optional_predictors`
: Optional predictors that may be included in the model by `step`. These should be provided even if you provide the 'predetermined_model_text'.

`mandatory_interactions`
: Interactions that must be included in the model. These should be provided even if you provide the 'predetermined_model_text'.

`optional_interactions`
: Optional interactions that may be included in the model by `step`. These should be provided even if you provide the 'predetermined_model_text'.

`model_threshold_method`
: One of *'youden'*, *'topleft'*, *'heuristic'*. Please see description below.

`scoring_system`
: Name of the pre-existing scoring system. This is ignored if *develop_model* is *TRUE*.

`predetermined_threshold`
: Pre-determined threshold of the pre-existing scoring system. This is mandatory when *develop_model* is *FALSE* and when the *outcome_type* is *'binary'* or *'time-to-event'*. This is ignored if *develop_model* is *TRUE* or when the *outcome_type* is *'quantitative'*.

`higher_values_event`
: *TRUE* if higher values of the pre-existing system indicates event and *FALSE* otherwise. This is mandatory when *develop_model* is *FALSE* and when the *outcome_type* is *'binary'* or *'time-to-event'*. This is ignored if *develop_model* is *TRUE* or when the *outcome_type* is *'quantitative'*.

`each_simulation`
: The number of the simulation in the prepared datasets. Please see `prepare_datasets`.

`bootstrap_sample`
: TRUE if you are calculating the bootstrap and test performance and FALSE if you are calculating the apparent performance. Please see below and Collins et al, 2024.

`verbose`
: *TRUE* if the progress must be displayed and *FALSE* otherwise.

Details

General comment Most of the input parameters are already available from the generic and specific input parameters created using `create_generic_input_parameters` and `create_specific_input_parameters`. This function is used by the `perform_analysis` function which provides the correct input parameters based on the entries provided in `create_generic_input_parameters` and `create_specific_input_parameters`.

Overview This is a form of enhanced bootstrapping internal validation approach to calculate the optimism-corrected performance measures described by Collins et al., 2024. This involves calculating the apparent performance by developing the model in the entire dataset, repeated sampling with replacement (bootstrap sample), evaluating the performance of the model in each simulation of the bootstrap sample (bootstrap performance), evaluating the performance of the model (developed in the bootstrap sample of each simulation) on the 'test sample' i.e., all the subjects in the dataset from which the bootstrap sample was obtained (test performance), calculating the optimism as the difference between bootstrap performance and test performance in each simulation, calculating the average optimism, and finally subtracting the average optimism from the apparent performance to calculate the optimism-corrected performance measures (Collins et al, 2024).

The model development is performed using `glm` for all outcomes other than time-to-event outcomes and `coxph` for time-to-event outcomes. You can either provide a model text that you have developed

for finer control of the interactions to be considered or included or you can let the computer build the model text based on the mandatory and optional predictors and interactions.

Linear predictors The linear predictor describes the relationship between the outcome and the predictors, and is a function of the covariate (predictor) values and coefficients of the regression. It can be described by the following relation. *Linear predictor = alpha + beta_predictors * predictors + beta_(predictors_interactions) (if interactions between predictors are included) + error*.

However, except for linear regression, the linear predictor must be transformed to obtain the outcome. This is because of the way generalised linear regression attempts to create a linear relationship between the outcome and predictors.

If *'Y'* is the outcome, the linear predictor is *'logit Y'* for binary outcomes; therefore, inverse logit transformation must be performed to convert the linear predictor to obtain the probability of an outcome. For count outcomes, the linear predictor is *'log Y'*; therefore, exponential transformation is required to obtain the predicted number of events.

For time-to-event outcomes, the linear predictor gives the hazard of an event at various time points for a subject at the given covariate levels. The function `basehaz` provides a more clinically meaningful cumulative hazard of an event by time *'t'*, denoted as *'H(t)'*. The function `basehaz` provides the cumulative hazard of the event at each time point denoted as *'t'* for each subject in the 'training' set. The cumulative hazard of the event by time *'t'* of a new subject can be calculated using the relation mentioned in the description of `basehaz` (please see the section on calculating H(t;x)). Using this relation, one can find the closest time point of a 'new' subject to the time points in the output of `basehaz`, the corresponding cumulative hazard for the 'first' subject (or any other subject for whom the cumulative hazard at each time point is available), and the differences in covariate values between the 'new' subject and the 'first' subject to calculate the cumulative hazard of event by the follow-up time of the new subject. The survival probability is exp(-H(t)) (Simon et al. 2024); therefore, one can calculate the probability of event by time *'t'* as *1 - exp(-H(t))*.

For continous outcomes, no transformation of linear predictor is required to obtain the outcome.

Obtaining linear predictors In regression models, we can get the get the transformed values of the linear predictors (lp) (i.e., inverse logit transformation for binary outcome and exponential for count outcomes) based on the regression model directly. For example, using the type = "response" in `predict` function gives this information directly for all outcome types other than time-to-event outcomes, which are analysed with `coxph`. For time-to-event outcomes, the type = "expected" gives the cumulative hazard by *'t'* after adjusting for the covariates (`predict.coxph`), from which one can estimate the probability of the event by the time *'t'* using the relations described above.

Missing linear predictors When predicting using the regression models directly as described above, there must be no missing data for the predictors included in the model. One possibility is to not make a prediction at all. However, in real life some of these predictors will be missing but a decision must be made. One possibility is multiple imputation. However, some assumptions about the missing data can be difficult to verify (Heymans et al, 2022). Another possibility is to exclude the missing predictor (whose value is missing) from the regression equation. Although the coefficient values would have been different without the predictor, it is impossible to develop and validate for all scenarios of missing predictors. This function calculates the linear predictor by excluding the predictors which contain missing data (for that subject) using the regression model developed on subjects without missing data. If the coefficient values in the model indicates *NA* (which should alert people to overfitting the data or levels with sparse data), the variable level itself is removed from calculating the linear predictor. To a large extent, the method used assumes that external validation will be performed before changing clinical practice and the application of this

method compared to other methods of handling missing data must be compared as part of external validation.

Conversion of probabilities of event (linear predictors) for binary and time-to-event outcomes to event versus no event There are multiple ways of converting probabilities of event (linear predictors) for binary and time-to-event outcomes to event versus no event. For example, one can consider that the probabilities of event are from binomial distribution for binary outcomes. Alternatively, one can choose an 'optimal threshold' (on the training set) using the `roc` and `coords` functions. There are two types of threshold calculated by the `coords` function: *'Youden'* and *'closest to top left'*. For further information, please see `coords`. Occasionally, it may not be possible to obtain the threshold using `roc` and `coords`. A function that performs a rough estimation of the threshold based on prevalence is included in the source code of this function (please see 'calculate_heuristic_threshold' function, included as part of this function).

Intercept-slope adjustment In regression models, the intercept and slope can be adjusted (Van Calster et al., 2019). The calibration intercept and slope are calculated according to the supplement of Van Calster et al., 2019. The paper provides details only for logistic regression, but the procedures are based on glm, i.e., they are applicable in glm models. The relation is regression equation is Y = *calibration_intercept + beta * linear predictor*.

Note that the linear predictor in the equation is used as variable rather than as an offset term as with calculation of calibration intercept only. The linear predictors must be back-transformed to the original scale before their use in the calibration regression equation.

Robust methods for calibration slope adjustment for time-to-events are still being developed. Until such methods become widely available, this function uses similar principles as that described for binary outcomes for time-to-event outcomes. These should be considered experimental until further evaluation of the performance of calibration adjustment in external samples. It should be noted however, that for time-to-event outcomes, Cox regression does not have an intercept separately, as the intercept is included in the baseline hazard (SAS support 2017). Therefore, with regards to time-to-event outcomes, there is no change to the intercept, but there is a change to the slope when calibration adjusted models are created.

Model with with only the mandatory predictors but based on the coefficients of the entire model This is solely for research purposes. Potential use of such a model with only the mandatory predictors, but based on the coefficients of the entire model will be to find the added value of measurement of optional predictors, particularly when there is a single mandatory predictor, for example, a treatment. It will be practically impossible to develop all the possible models with missing optional predictors. This model has the potential to provide predictions in this situation.

Value

`actual_training`
: Actual values in the training sample.

`predicted_training`
: Predicted values in the training sample.

`predicted_training_calibration_adjusted`
: Predicted values after calibration adjustment.

`predicted_training_adjusted_mandatory_predictors_only`
: Predicted values of a model with only the mandatory predictors, but based on the coefficients of the entire model.

`actual_only_validation`
: Actual values in the 'out-of-sample' subjects, i.e., the subjects excluded from the model development in each simulation.

`predicted_only_validation`
: Predicted values in the 'out-of-sample' subjects

`predicted_only_validation_calibration_adjusted`
: Predicted values in the out-of-sample subjects after calibration adjustment

`predicted_only_validation_adjusted_mandatory_predictors_only`
: Predicted values in the out-of-sample subjects using a model with only the mandatory predictors, but based on the coefficients of the entire model.

`actual_all_subjects`
: Actual values in all subjects with outcomes.

`predicted_all_subjects`
: Predicted values in all subjects with outcomes.

`predicted_all_subjects_calibration_adjusted`
: Predicted values in all subjects with outcomes after calibration adjustment.

`predicted_all_subjects_adjusted_mandatory_predictors_only`
: Predicted values in all subjects using a model with only the mandatory predictors, but based on the coefficients of the entire model.

`lp_training` Linear predictors in the 'training' sample.

`lp_only_validation`
: Linear predictors in the 'out-of-sample' subjects.

`lp_all_subjects`
: Linear predictors in all subjects with outcomes.

`lp_training_calibration_adjusted`
: Linear predictors in the training sample after calibration adjustment.

`lp_only_validation_calibration_adjusted`
: Linear predictors in the 'out-of-sample' subjects after calibration adjustment.

`lp_all_subjects_calibration_adjusted`
: Linear predictors in all subjects with outcomes after calibration adjustment.

`lp_training_adjusted_mandatory_predictors_only`
: Linear predictors in the training sample using a model with only the mandatory predictors, but based on the coefficients of the entire model.

`lp_only_validation_adjusted_mandatory_predictors_only`
: Linear predictors in the 'out-of-sample' subjects using a model with only the mandatory predictors, but based on the coefficients of the entire model.

`lp_all_subjects_adjusted_mandatory_predictors_only`
: Linear predictors in all subjects with outcomes using a model with only the mandatory predictors, but based on the coefficients of the entire model.

`time_training` Follow-up time in training sample (applicable only for time-to-event outcomes.)

`time_only_validation`
: Follow-up time in 'out-of-sample' subjects (applicable only for time-to-event outcomes.)

`time_all_subjects`
: Follow-up time in all subjects with outcomes (applicable only for time-to-event outcomes.)

regression_model	The regression model
html_file	Some output in html format, which will be used for final output.
outcome	Whether calculations could be made.

Author(s)

Kurinchi Gurusamy

References

Collins GS, Dhiman P, Ma J, Schlussel MM, Archer L, Van Calster B, et al. Evaluation of clinical prediction models (part 1): from development to external validation. Bmj. 2024;384:e074819.

Heymans MW, Twisk JWR. Handling missing data in clinical research. J Clin Epidemiol. 2022 Nov;151:185-188.

SAS Support. https://support.sas.com/kb/24/457.html (accessed on 16 January 2026).

Simon G, Aliferis C. Appendix A: Models for Time-to-Event Outcomes. In: Simon GJ, Aliferis C, editors. Artificial Intelligence and Machine Learning in Health Care and Medical Sciences: Best Practices and Pitfalls [Internet]. Cham (CH): Springer. https://www.ncbi.nlm.nih.gov/books/NBK610554/ (accessed on 13 December 2025). 2024.

Van Calster B, McLernon DJ, van Smeden M, Wynants L, Steyerberg EW, Bossuyt P, et al. Calibration: the Achilles heel of predictive analytics. BMC Medicine. 2019;17(1):230.

See Also

prepare_datasets glm predict coxph basehaz predict.coxph roc coords

Examples

```
library(survival)
colon$status <- factor(as.character(colon$status))
# For testing, only 5 simulations are used here. Usually at least 300 to 500
# simulations are a minimum. Increasing the simulations leads to more reliable results.
# The default value of 2000 simulations should provide reasonably reliable results.
generic_input_parameters <- create_generic_input_parameters(
  general_title = "Prediction of colon cancer death", simulations = 5,
  simulations_per_file = 20, seed = 1, df = colon, outcome_name = "status",
  outcome_type = "time-to-event", outcome_time = "time", outcome_count = FALSE,
  verbose = FALSE)$generic_input_parameters
analysis_details <- cbind.data.frame(
  name = c('age', 'single_mandatory_predictor', 'complex_models',
           'complex_models_only_optional_predictors', 'predetermined_model_text'),
  analysis_title = c('Simple cut-off based on age', 'Single mandatory predictor (rx)',
                     'Multiple mandatory and optional predictors',
                     'Multiple optional predictors only', 'Predetermined model text'),
  develop_model = c(FALSE, TRUE, TRUE, TRUE, TRUE),
  predetermined_model_text = c(NA, NA, NA, NA,
  "cph(Surv(time, status) ~ rx * age, data = df_training_complete, x = TRUE, y = TRUE)"),
  mandatory_predictors = c(NA, 'rx', 'rx; differ; perfor; adhere; extent', NA, "rx; age"),
```

```
  optional_predictors = c(NA, NA, 'sex; age; nodes', 'rx; differ; perfor', NA),
  mandatory_interactions = c(NA, NA, 'rx; differ; extent', NA, NA),
  optional_interactions = c(NA, NA, 'perfor; adhere; sex; age; nodes', 'rx; differ', NA),
  model_threshold_method = c(NA, 'youden', 'youden', 'youden', 'youden'),
  scoring_system = c('age', NA, NA, NA, NA),
  predetermined_threshold = c('60', NA, NA, NA, NA),
  higher_values_event = c(TRUE, NA, NA, NA, NA)
)
write.csv(analysis_details, paste0(tempdir(), "/analysis_details.csv"),
          row.names = FALSE, na = "")
analysis_details_path <- paste0(tempdir(), "/analysis_details.csv")
# verbose is TRUE as default. If you do not want the outcome displayed, you can
# change this to FALSE
results <- create_specific_input_parameters(
  generic_input_parameters = generic_input_parameters,
  analysis_details_path = analysis_details_path, verbose = TRUE)
specific_input_parameters <- results$specific_input_parameters
# Set a seed for reproducibility - Please see details above
set.seed(generic_input_parameters$seed)
prepared_datasets <- {prepare_datasets(
  df = generic_input_parameters$df,
  simulations = generic_input_parameters$simulations,
  outcome_name = generic_input_parameters$outcome_name,
  outcome_type = generic_input_parameters$outcome_type,
  outcome_time = generic_input_parameters$outcome_time,
  verbose = TRUE)}
# There is no usually no requirement to call this function directly. This is used
# by the perform_analysis function to create the actual and predicted values.
specific_input_parameters_each_analysis <- specific_input_parameters[[1]]
actual_predicted_results_apparent <- {calculate_actual_predicted(
      prepared_datasets = prepared_datasets,
      outcome_name = generic_input_parameters$outcome_name,
      outcome_type = generic_input_parameters$outcome_type,
      outcome_time = generic_input_parameters$outcome_time,
      outcome_count = generic_input_parameters$outcome_count,
      develop_model = specific_input_parameters_each_analysis$develop_model,
      predetermined_model_text =
      specific_input_parameters_each_analysis$predetermined_model_text,
     mandatory_predictors = specific_input_parameters_each_analysis$mandatory_predictors,
      optional_predictors = specific_input_parameters_each_analysis$optional_predictors,
    mandatory_interactions = specific_input_parameters_each_analysis$mandatory_interactions,
    optional_interactions = specific_input_parameters_each_analysis$optional_interactions,
    model_threshold_method = specific_input_parameters_each_analysis$model_threshold_method,
      scoring_system = specific_input_parameters_each_analysis$scoring_system,
    predetermined_threshold = specific_input_parameters_each_analysis$predetermined_threshold,
      higher_values_event = specific_input_parameters_each_analysis$higher_values_event,
      each_simulation = 1, bootstrap_sample = FALSE, verbose = TRUE
    )}
  bootstrap_results <- lapply(1:generic_input_parameters$simulations,
  function(each_simulation) {
  calculate_actual_predicted(
    prepared_datasets = prepared_datasets,
    outcome_name = generic_input_parameters$outcome_name,
```

```
    outcome_type = generic_input_parameters$outcome_type,
    outcome_time = generic_input_parameters$outcome_time,
    outcome_count = generic_input_parameters$outcome_count,
    develop_model = specific_input_parameters_each_analysis$develop_model,
    predetermined_model_text =
      specific_input_parameters_each_analysis$predetermined_model_text,
    mandatory_predictors = specific_input_parameters_each_analysis$mandatory_predictors,
    optional_predictors = specific_input_parameters_each_analysis$optional_predictors,
   mandatory_interactions = specific_input_parameters_each_analysis$mandatory_interactions,
   optional_interactions = specific_input_parameters_each_analysis$optional_interactions,
   model_threshold_method = specific_input_parameters_each_analysis$model_threshold_method,
    scoring_system = specific_input_parameters_each_analysis$scoring_system,
   predetermined_threshold = specific_input_parameters_each_analysis$predetermined_threshold,
    higher_values_event = specific_input_parameters_each_analysis$higher_values_event,
    each_simulation = each_simulation, bootstrap_sample = TRUE, verbose = TRUE
  )
})
```

`calculate_performance` *Calculate performance of prognostic models*

Description

This function calculates the different performance measures of prognostic models and factors. Please see below for more details.

Usage

```
calculate_performance(outcome_type, time, outcome_count, actual, predicted,
develop_model, lp)
```

Arguments

`outcome_type`	One of *'binary'*, *'time-to-event'*, *'quantitative'*. Count outcomes are included in *'quantitative'* outcome type and can be differentiated from continuous outcomes by specifying outcome_count as *TRUE*. Please see examples below.
`time`	Times at which the outcome was measured. This is applicable only for *'time-to-event'* outcome. For other outcome types, enter *NA*.
`outcome_count`	*TRUE* if the outcome was a count outcome and *FALSE* otherwise.
`actual`	A vector of actual values.
`predicted`	A vector of predicted values.
`develop_model`	*TRUE*, if you a model was developed; *FALSE*, if a scoring system with a predetermined threshold (if applicable) was used.
`lp`	A vector of linear predictors (applicable only if you have developed a model.)

Details

General comment Most of the input parameters are already available from the generic and specific input parameters created using `create_generic_input_parameters` and `create_specific_input_parameters`. This function is used by the `compile_results` function which provides the correct input parameters based on the entries while using `create_generic_input_parameters` and `create_specific_input_parameters` and the output from `calculate_actual_predicted`.

Performance measures The performance was measured by the following parameters. **Accuracy** Number of correct predictions/number of participants in whom the predictions were made (Rainio et al., 2024). **Calibration** Three measures of calibration are used. *Observed/expected ratio* Please see Riley et al., 2024. Values closer to 1 are better; ratios < 1 indicate overestimation of risk by the model while ratios > 1 indicate underestimation of risk by the model (Riley et al., 2024).

We treated underestimation and overestimation equally, i.e., an observed-expected ratio of 0.8 was considered equivalent to 1/0.8 = 1.25. Therefore, we converted the observed-expected ratios to be in the same direction (less than 1) *('modified observed-expected ratio')*. This ensured that while calculating the test performance and bootstrap performance, lower numbers consistently indicated worse test performance (as they are more distant from 1) and higher numbers consistently indicated better performance (noting that the maximum value of the modified observed-expected ratio was 1). This modification also helps in interpretation of comparison for different models, some of which may overestimate the risk while others might underestimate the risk.

For assessing the calibration, when the expected events were zero, 0.5 were added to both the observed events and expected events.

Calibration intercept and *calibration slope*: *Calibration slope* quantifies the spread of the risk probabilities in relation to the observed events (Stevens et al., 2020). We used the methods described by Riley et al, 2024 to calculate the calibration intercept and slope for all outcomes other than time-to-event outcomes. Essentially, this involves the following regression equation: *Y = calibration intercept + coefficient * linear predictor*, where *'Y'* is the log odds of observed event, log risk of observed event, and the untransformed outcomes for binary, count, and continuous outcomes respectively.

Estimation in time-to-event is lot more uncertain and should be considered experimental. Please note that that Cox regression does not have an intercept separately, as the intercept is included in the baseline hazard (SAS Support, 2017).

Values closer to 1 indicate better performance when the intercept is close to 0; values further away from 1 indicate that the predictions are incorrect in some ranges (Van Calster et al., 2019; Riley et al., 2024; Stevens e al., 2020). The further away from 1, the worse the relationship between the log odds of observed event, log hazard, log risk of observed event, and the untransformed outcome with the linear predictor.

To allow easy comparison with lower values indicating closer to 1 and higher values indicating further away from 1, this function also calculates the *'modified calibration slope'* using the following formula: *'Modified calibration slope = absolute value (1-calibration slope)'*.

Calibration intercept (also called "calibration-in-the-large") in the calibration regression equation evaluates whether the observed event proportion equals the average predicted risk (Van Calster et al, 2019).

This function also calculates the *'modified calibration intercept'* as the absolute value of calibration intercept to allow lower values indicating closer to 0 and higher values indicating further away from 0.

C-statistic This is the area under the ROC curve and a measure of discrimination (Riley et al. 2025). This was calculated using roc. Higher values indicate better performance.

Value

`output` A dataframe of the calculated evaluation parameters

Author(s)

Kurinchi Gurusamy

References

Rainio O, Teuho J, Klén R. Evaluation metrics and statistical tests for machine learning. Scientific Reports. 2024;14(1):6086.

Riley RD, Archer L, Snell KIE, Ensor J, Dhiman P, Martin GP, et al. Evaluation of clinical prediction models (part 2): how to undertake an external validation study. BMJ. 2024;384:e074820.

SAS Support. https://support.sas.com/kb/24/457.html (accessed on 16 January 2026).

Stevens RJ, Poppe KK. Validation of clinical prediction models: what does the "calibration slope" really measure? Journal of Clinical Epidemiology. 2020;118:93-9.

Van Calster B, McLernon DJ, van Smeden M, Wynants L, Steyerberg EW, Bossuyt P, et al. Calibration: the Achilles heel of predictive analytics. BMC Medicine. 2019;17(1):230.

See Also

roc

Examples

```
library(survival)
colon$status <- factor(as.character(colon$status))
# For testing, only 5 simulations are used here. Usually at least 300 to 500
# simulations are a minimum. Increasing the simulations leads to more reliable results.
# The default value of 2000 simulations should provide reasonably reliable results.
generic_input_parameters <- create_generic_input_parameters(
  general_title = "Prediction of colon cancer death", simulations = 5,
  simulations_per_file = 20, seed = 1, df = colon, outcome_name = "status",
  outcome_type = "time-to-event", outcome_time = "time", outcome_count = FALSE,
  verbose = FALSE)$generic_input_parameters
analysis_details <- cbind.data.frame(
  name = c('age', 'single_mandatory_predictor', 'complex_models',
           'complex_models_only_optional_predictors', 'predetermined_model_text'),
  analysis_title = c('Simple cut-off based on age', 'Single mandatory predictor (rx)',
                     'Multiple mandatory and optional predictors',
                     'Multiple optional predictors only', 'Predetermined model text'),
  develop_model = c(FALSE, TRUE, TRUE, TRUE, TRUE),
  predetermined_model_text = c(NA, NA, NA, NA,
  "cph(Surv(time, status) ~ rx * age, data = df_training_complete, x = TRUE, y = TRUE)"),
  mandatory_predictors = c(NA, 'rx', 'rx; differ; perfor; adhere; extent', NA, "rx; age"),
  optional_predictors = c(NA, NA, 'sex; age; nodes', 'rx; differ; perfor', NA),
```

```
  mandatory_interactions = c(NA, NA, 'rx; differ; extent', NA, NA),
  optional_interactions = c(NA, NA, 'perfor; adhere; sex; age; nodes', 'rx; differ', NA),
  model_threshold_method = c(NA, 'youden', 'youden', 'youden', 'youden'),
  scoring_system = c('age', NA, NA, NA, NA),
  predetermined_threshold = c('60', NA, NA, NA, NA),
  higher_values_event = c(TRUE, NA, NA, NA, NA)
)
write.csv(analysis_details, paste0(tempdir(), "/analysis_details.csv"),
          row.names = FALSE, na = "")
analysis_details_path <- paste0(tempdir(), "/analysis_details.csv")
# verbose is TRUE as default. If you do not want the outcome displayed, you can
# change this to FALSE
results <- create_specific_input_parameters(
  generic_input_parameters = generic_input_parameters,
  analysis_details_path = analysis_details_path, verbose = TRUE)
specific_input_parameters <- results$specific_input_parameters
# Set a seed for reproducibility - Please see details above
set.seed(generic_input_parameters$seed)
prepared_datasets <- {prepare_datasets(
  df = generic_input_parameters$df,
  simulations = generic_input_parameters$simulations,
  outcome_name = generic_input_parameters$outcome_name,
  outcome_type = generic_input_parameters$outcome_type,
  outcome_time = generic_input_parameters$outcome_time,
  verbose = TRUE)}
# There is no usually no requirement to call this function directly. This is used
# by the perform_analysis function to create the actual and predicted values.
specific_input_parameters_each_analysis <- specific_input_parameters[[1]]
actual_predicted_results_apparent <- {calculate_actual_predicted(
      prepared_datasets = prepared_datasets,
      outcome_name = generic_input_parameters$outcome_name,
      outcome_type = generic_input_parameters$outcome_type,
      outcome_time = generic_input_parameters$outcome_time,
      outcome_count = generic_input_parameters$outcome_count,
      develop_model = specific_input_parameters_each_analysis$develop_model,
      predetermined_model_text =
      specific_input_parameters_each_analysis$predetermined_model_text,
     mandatory_predictors = specific_input_parameters_each_analysis$mandatory_predictors,
      optional_predictors = specific_input_parameters_each_analysis$optional_predictors,
    mandatory_interactions = specific_input_parameters_each_analysis$mandatory_interactions,
    optional_interactions = specific_input_parameters_each_analysis$optional_interactions,
    model_threshold_method = specific_input_parameters_each_analysis$model_threshold_method,
      scoring_system = specific_input_parameters_each_analysis$scoring_system,
    predetermined_threshold = specific_input_parameters_each_analysis$predetermined_threshold,
      higher_values_event = specific_input_parameters_each_analysis$higher_values_event,
      each_simulation = 1, bootstrap_sample = FALSE, verbose = TRUE
    )}
  bootstrap_results <- lapply(1:generic_input_parameters$simulations,
  function(each_simulation) {
  calculate_actual_predicted(
    prepared_datasets = prepared_datasets,
    outcome_name = generic_input_parameters$outcome_name,
    outcome_type = generic_input_parameters$outcome_type,
```

```
    outcome_time = generic_input_parameters$outcome_time,
    outcome_count = generic_input_parameters$outcome_count,
    develop_model = specific_input_parameters_each_analysis$develop_model,
    predetermined_model_text =
      specific_input_parameters_each_analysis$predetermined_model_text,
    mandatory_predictors = specific_input_parameters_each_analysis$mandatory_predictors,
    optional_predictors = specific_input_parameters_each_analysis$optional_predictors,
   mandatory_interactions = specific_input_parameters_each_analysis$mandatory_interactions,
   optional_interactions = specific_input_parameters_each_analysis$optional_interactions,
   model_threshold_method = specific_input_parameters_each_analysis$model_threshold_method,
    scoring_system = specific_input_parameters_each_analysis$scoring_system,
   predetermined_threshold = specific_input_parameters_each_analysis$predetermined_threshold,
    higher_values_event = specific_input_parameters_each_analysis$higher_values_event,
    each_simulation = each_simulation, bootstrap_sample = TRUE, verbose = TRUE
  )
})
apparent_performance <- {cbind.data.frame(
  performance = "apparent", simulation = NA,
  calculate_performance(
    outcome_type = generic_input_parameters$outcome_type,
    time = actual_predicted_results_apparent$time_all_subjects,
    outcome_count = generic_input_parameters$outcome_count,
    actual = actual_predicted_results_apparent$actual_all_subjects,
    predicted = actual_predicted_results_apparent$predicted_all_subjects,
    develop_model = specific_input_parameters_each_analysis$develop_model,
    lp = actual_predicted_results_apparent$lp_all_subjects
  ))}
```

`compile_results` *Run the analysis and compile the results.*

Description

This is wrapper function that takes generic_input_parameters created using `create_generic_input_parameters`, specific_input_parameters created using `create_specific_input_parameters` and the datasets prepared using `prepare_datasets`, and provides the results. For details, please see below.

Usage

```
compile_results(generic_input_parameters, specific_input_parameters,
prepared_datasets, verbose)
```

Arguments

`generic_input_parameters`

This is a list that contains common information across models. If one or more items are missing or incorrect, this may result in error. Therefore, we recommend that you use the `create_generic_input_parameters` function to create this input.

`specific_input_parameters`
: This is a list that contains information related to each model or scoring system. If one or more items are missing or incorrect, this may result in error. Therefore, we recommend that you use the `create_specific_input_parameters`.

`prepared_datasets`
: Datasets prepared using the `prepare_datasets`.

`verbose`
: *TRUE* if the progress must be displayed and *FALSE* otherwise.

Details

Overview This is wrapper function that takes generic_input_parameters created using `create_generic_input_parameters` specific_input_parameters created using `create_specific_input_parameters`, and the datasets prepared using `prepare_datasets`, and provides the results. It uses the `perform_analysis` function (which itself uses the `calculate_actual_predicted` to develop and run models and `calculate_performance` to calculate the performance). It also creates calibration curves for each model and summarises the information across the different scoring systems and models.

The following steps must be done sequentially. Please see example below which provides the details of how to perform the analysis. 1. *Process the data:* The dataset should be prepared correctly for the functions to work correctly. This can be done using `process_data`. 2. *Create the generic input parameters:* The generic input parameters should be provided. You can check that the input parameters are correct using `create_generic_input_parameters`. 3. *Create the specific input parameters:* The specific input parameters should be provided. You can check that the input parameters are correct using `create_specific_input_parameters`. 4. *Prepare the datasets:* The datasets for each simulation are prepared using `prepare_datasets`. 5. *Provide the correct parameters to this function:* Input 'generic_input_parameters', 'specific_input_parameters', and 'prepared_datasets' to obtain the results.

Preparing datasets for each simulation Please see `prepare_datasets`.

Calculation of actual and predicted values Please see `calculate_actual_predicted`.

Calculation of performance measures Please see `calculate_performance`.

Calculation of means and confidence intervals Please see `perform_analysis`.

Calibration curves Flexible calibration curves were created using the package `CalibrationCurves`, based on the paper by Van Calster et al, 2016.

The calibration curves have been described for only one simulation. It is impractical to interpret each calibration curve (one for each simulation). Therefore, the calibration curves are shown only for the apparent performance. As an experimental method, the linear predictors were averaged across the simulations and calibration curves for the average test performance are also provided.

When the flexible calibration curves resulted in errors or were not possible, for example, the average linear predictors across simulations for time-to-event outcomes, `calibration_plot` was used to create the calibration plots. Appropriate backtransformations were performed to reverse the transformations applied while calculating the linear predictors. For further information about transformations, please see `calculate_actual_predicted`.

Value

`results`
: These include the performance results for each model.

summary_results
: These include the summary performance results.

html_file_location
: This provides the location of the html file, which contains the results in html format. This can be downloaded and the content copied to word document or PowerPoint presentations.

Author(s)

Kurinchi Gurusamy

References

Collins GS, Dhiman P, Ma J, Schlussel MM, Archer L, Van Calster B, et al. Evaluation of clinical prediction models (part 1): from development to external validation. Bmj. 2024;384:e074819.

R Package "coxed". https://CRAN.R-project.org/package=coxed)

Van Calster B, Nieboer D, Vergouwe Y, De Cock B, Pencina MJ, Steyerberg EW. A calibration hierarchy for risk models was defined: from utopia to empirical data. Journal of Clinical Epidemiology. 2016;74:167-76.

See Also

process_data prepare_datasets perform_analysis calculate_actual_predicted calculate_performance CalibrationCurves

Examples

```
# Load packages ####
library(base64enc)
library(mime)
library(pROC)
library(survival)
library(ggplot2)
library(CalibrationCurves)
library(predtools)
colon$status <- factor(as.character(colon$status))
# For testing, only 5 simulations are used here. Usually at least 300 to 500
# simulations are a minimum. Increasing the simulations leads to more reliable results.
# The default value of 2000 simulations should provide reasonably reliable results.
generic_input_parameters <- create_generic_input_parameters(
  general_title = "Prediction of colon cancer death", simulations = 5,
  simulations_per_file = 20, seed = 1, df = colon, outcome_name = "status",
  outcome_type = "time-to-event", outcome_time = "time", outcome_count = FALSE,
  verbose = FALSE)$generic_input_parameters
analysis_details <- cbind.data.frame(
  name = c('age', 'single_mandatory_predictor', 'complex_models',
           'complex_models_only_optional_predictors', 'predetermined_model_text'),
  analysis_title = c('Simple cut-off based on age', 'Single mandatory predictor (rx)',
                     'Multiple mandatory and optional predictors',
                     'Multiple optional predictors only', 'Predetermined model text'),
  develop_model = c(FALSE, TRUE, TRUE, TRUE, TRUE),
```

```
  predetermined_model_text = c(NA, NA, NA, NA,
   "cph(Surv(time, status) ~ rx * age, data = df_training_complete, x = TRUE, y = TRUE)"),
 mandatory_predictors = c(NA, 'rx', 'rx; differ; perfor; adhere; extent', NA, "rx; age"),
  optional_predictors = c(NA, NA, 'sex; age; nodes', 'rx; differ; perfor', NA),
  mandatory_interactions = c(NA, NA, 'rx; differ; extent', NA, NA),
 optional_interactions = c(NA, NA, 'perfor; adhere; sex; age; nodes', 'rx; differ', NA),
  model_threshold_method = c(NA, 'youden', 'youden', 'youden', 'youden'),
  scoring_system = c('age', NA, NA, NA, NA),
  predetermined_threshold = c('60', NA, NA, NA, NA),
  higher_values_event = c(TRUE, NA, NA, NA, NA)
)
# For the demonstration, only the first row is run. Please remove the line
# below i.e., analysis_details <- analysis_details[1,]
analysis_details <- analysis_details[1,]
write.csv(analysis_details, paste0(tempdir(), "/analysis_details.csv"),
          row.names = FALSE, na = "")
analysis_details_path <- paste0(tempdir(), "/analysis_details.csv")
# verbose is TRUE as default. If you do not want the outcome displayed, you can
# change this to FALSE, as shown here
results <- create_specific_input_parameters(
  generic_input_parameters = generic_input_parameters,
  analysis_details_path = analysis_details_path, verbose = FALSE)
specific_input_parameters <- results$specific_input_parameters
# Set a seed for reproducibility - Please see details above
set.seed(generic_input_parameters$seed)
prepared_datasets <- {prepare_datasets(
  df = generic_input_parameters$df,
  simulations = generic_input_parameters$simulations,
  outcome_name = generic_input_parameters$outcome_name,
  outcome_type = generic_input_parameters$outcome_type,
  outcome_time = generic_input_parameters$outcome_time,
  verbose = FALSE)}
results <- compile_results(generic_input_parameters, specific_input_parameters,
                           prepared_datasets, verbose = FALSE)
# Results html_file_location
results$html_file_location
```

`create_generic_input_parameters` *Create generic input parameters*

Description

This function simply checks whether the input parameters are correct and if correct, creates a list from the input parameters. This also makes some corrections when possible (i.e., when there were minor correctable issues in the input parameters).

Usage

```
create_generic_input_parameters(general_title, simulations, simulations_per_file,
seed, df, outcome_name, outcome_type, outcome_time, outcome_count, verbose)
```

Arguments

general_title	A general title for your analysis
simulations	The number of simulations required. Usually at least 300 to 500 simulations are a minimum. Increasing the simulations leads to more reliable results. The default value of 2000 simulations should provide reasonably reliable results.
simulations_per_file	This is to manage the memory requirements. The default value of 20 simulations per file should work in most instances.
seed	Please see prepare_datasets for details.
df	The dataset used for the analysis. This must be provided as a dataframe. Data in files can be converted to dataframes with appropriate field types using process_data.
outcome_name	Name of the columnn that contains the outcome data. This must be a column name in the *'df'* provided as input.
outcome_type	One of *'binary'*, *'time-to-event'*, *'quantitative'*. Count outcomes are included in *'quantitative'* outcome type and can be differentiated from continuous outcomes by specifying outcome_count as *TRUE*. Please see examples below.
outcome_time	The name of the column that provides the follow-up time. This is applicable only for *'time-to-event'* outcome. For other outcome types, enter *NA*.
outcome_count	*TRUE* if the outcome was a count outcome and *FALSE* otherwise.
verbose	*TRUE* if the outcome message must be displayed and *FALSE* otherwise.

Value

outcome	The outcome containing the processing details. If some corrections were made, the corrections are included in the outcome. If there was a fatal error, the reason for the fatal error is provided.
generic_input_parameters	A list with information for further analyses. If there was a fatal error, the reason for the fatal error is displayed and generic_input_parameters is *NULL*.

Author(s)

Kurinchi Gurusamy

See Also

process_data prepare_datasets

Examples

```
# Correct parameters ####
# Binary outcome
# verbose is TRUE, therefore, the outcome message will be displayed
results <- create_generic_input_parameters(
general_title = "Prediction of penguin species", simulations = 2000,
simulations_per_file = 20, seed = 1, df = penguins, outcome_name = "species",
```

```
outcome_type = "binary", outcome_time = NA, outcome_count = FALSE, verbose = TRUE)
generic_input_parameters <- results$generic_input_parameters
generic_input_parameters

# Time-to-event outcome
library(survival)
# The field 'status' is provided as numeric. This must be converted to factor. In
# this example, we can convert this to factor using a command. For conversion of more
# columns, please use process_data function.
colon$status <- factor(as.character(colon$status))
# verbose is FALSE, therefore, the outcome message will not be displayed, but the
# outcome is stored.
results <- create_generic_input_parameters(
general_title = "Prediction of colon cancer death", simulations = 2000,
simulations_per_file = 20, seed = 1, df = colon, outcome_name = "status",
outcome_type = "time-to-event", outcome_time = "time", outcome_count = FALSE,
verbose = FALSE)
# Display outcome
results$outcome
# Display generic_input_parameters
generic_input_parameters <- results$generic_input_parameters
generic_input_parameters

# Continuous outcome
# verbose is not supplied, therefore, the outcome message will be displayed as
# this is the default.
results <- create_generic_input_parameters(
general_title = "Prediction of iris petal length", simulations = 2000,
simulations_per_file = 20, seed = 1, df = iris, outcome_name = "Petal.Length",
outcome_type = "quantitative", outcome_time = NA, outcome_count = FALSE)
generic_input_parameters <- results$generic_input_parameters
generic_input_parameters

# Count outcomes
results <- create_generic_input_parameters(
general_title = "Prediction of warp breaks", simulations = 2000,
simulations_per_file = 20, seed = 1, df = warpbreaks, outcome_name = "breaks",
outcome_type = "quantitative", outcome_time = NA, outcome_count = TRUE)
generic_input_parameters <- results$generic_input_parameters
generic_input_parameters

# Non fatal errors ####
results <- create_generic_input_parameters(
general_title = "", simulations = "Use default",
simulations_per_file = "Use default", seed = "Use default",
df = warpbreaks, outcome_name = "breaks",
outcome_type = "quantitative", outcome_time = "Use default", outcome_count = TRUE,
verbose = TRUE)
generic_input_parameters <- results$generic_input_parameters
generic_input_parameters

# Fatal error ####
# Note the dataframe name supplied within quotes.
```

```
results <- create_generic_input_parameters(
general_title = "", simulations = "Use default",
simulations_per_file = "Use default", seed = "Use default",
df = "warpbreaks", outcome_name = "breaks", outcome_type = "quantitative",
outcome_time = "Use default", outcome_count = TRUE, verbose = TRUE)
generic_input_parameters <- results$generic_input_parameters
generic_input_parameters
```

`create_specific_input_parameters` *Create specific input parameters*

Description

This function converts the analysis details that you provide in a *'csv'* file to a list that will be used for analysis. Only valid rows are included in this list. If there are invalid rows, the reasons for rejecting the rows are provided. Any revisions that make the rows valid are performed when possible.

Usage

```
create_specific_input_parameters(generic_input_parameters, analysis_details_path,
verbose)
```

Arguments

`generic_input_parameters`
: The generic input parameters that you generated with `create_generic_input_parameters` (recommended) or manually.

`analysis_details_path`
: The path to the *'csv'* file containing the following columns. *'name'*: The name of the analysis. Provide a short name. This will be displayed on graphs and will also be used for naming the analysis. The name must be unique. If the name does not follow the naming convention for R objects, a suitable name is created. *'analysis_title'*: The title of the analysis to be displayed. If the title is missing, the *'name'* is used as the *'analysis_title'*. *'develop_model'*: *TRUE*, if you want to develop a model; *FALSE*, if you want to use a scoring system with a predetermined threshold (if applicable). *'predetermined_model_text'*: You can create the model text from the mandatory and optional predictors and interactions or for finer control of the model, you can provide the model text directly. *'mandatory_predictors'*: Predictors that must be included in the model. These should be provided even if you provide the 'predetermined_model_text'. *'optional_predictors'*: Optional predictors that may be included in the model by `step`. These should be provided even if you provide the 'predetermined_model_text'. *'mandatory_interactions'*: Interactions that must be included in the model. These should be provided even if you provide the 'predetermined_model_text'. *'optional_interactions'*: Optional interactions that may be included in the model by `step`. These should be provided even if you provide the 'predetermined_model_text'.

'model_threshold_method': One of *'youden'*, *'topleft'*, *'heuristic'*. Please see description in `calculate_actual_predicted`. *'scoring_system'*: Name of the pre-existing scoring system. This is ignored if *develop_model* is *TRUE*. *'predetermined_threshold'*: Pre-determined threshold of the pre-existing scoring system. This is mandatory when *develop_model* is *FALSE* and when the *outcome_type* is *'binary'* or *'time-to-event'*. This is ignored if *develop_model* is *TRUE* or when the *outcome_type* is *'quantitative'*. *'higher_values_event'*: *TRUE* if higher values of the pre-existing system indicates event and *FALSE* otherwise. This is mandatory when *develop_model* is *FALSE* and when the *outcome_type* is *'binary'* or *'time-to-event'*. This is ignored if *develop_model* is *TRUE* or when the *outcome_type* is *'quantitative'*.

`verbose` *TRUE* if the outcome message must be displayed and *FALSE* otherwise.

Value

`outcome` The outcome containing the processing details. If some corrections were made, the corrections are included in the outcome. If some rows were not valid, the reason for the row not being valid is provided.

`specific_input_parameters`
A list with the information for further analyses.

Author(s)

Kurinchi Gurusamy

Examples

```
library(survival)
colon$status <- factor(as.character(colon$status))
generic_input_parameters <- create_generic_input_parameters(
  general_title = "Prediction of colon cancer death", simulations = 2000,
  simulations_per_file = 20, seed = 1, df = colon, outcome_name = "status",
  outcome_type = "time-to-event", outcome_time = "time", outcome_count = FALSE,
  verbose = FALSE)$generic_input_parameters
analysis_details <- cbind.data.frame(
  name = c('age', 'single_mandatory_predictor', 'complex_models',
           'complex_models_only_optional_predictors', 'predetermined_model_text'),
  analysis_title = c('Simple cut-off based on age', 'Single mandatory predictor (rx)',
                     'Multiple mandatory and optional predictors',
                     'Multiple optional predictors only', 'Predetermined model text'),
  develop_model = c(FALSE, TRUE, TRUE, TRUE, TRUE),
  predetermined_model_text = c(NA, NA, NA, NA,
  "cph(Surv(time, status) ~ rx * age, data = df_training_complete, x = TRUE, y = TRUE)"),
  mandatory_predictors = c(NA, 'rx', 'rx; differ; perfor; adhere; extent', NA, "rx; age"),
  optional_predictors = c(NA, NA, 'sex; age; nodes', 'rx; differ; perfor', NA),
  mandatory_interactions = c(NA, NA, 'rx; differ; extent', NA, NA),
  optional_interactions = c(NA, NA, 'perfor; adhere; sex; age; nodes', 'rx; differ', NA),
  model_threshold_method = c(NA, 'youden', 'youden', 'youden', 'youden'),
  scoring_system = c('age', NA, NA, NA, NA),
  predetermined_threshold = c('60', NA, NA, NA, NA),
  higher_values_event = c(TRUE, NA, NA, NA, NA)
```

```
)
write.csv(analysis_details, paste0(tempdir(), "/analysis_details.csv"),
          row.names = FALSE, na = "")
analysis_details_path <- paste0(tempdir(), "/analysis_details.csv")
# verbose is TRUE as default. If you do not want the outcome displayed, you can
# change this to FALSE
results <- create_specific_input_parameters(
  generic_input_parameters = generic_input_parameters,
  analysis_details_path = analysis_details_path, verbose = TRUE)
specific_input_parameters <- results$specific_input_parameters
```

`get_outcome_status_at_specific_time` *Get outcome status at specific time*

Description

Some may want to calculate the outcome status at a specified time to predict the outcome at a specific time. This function gets the outcome status at the specific time.

Usage

```
get_outcome_status_at_specific_time(df, status_field, time_field, specific_time)
```

Arguments

`df`	The dataframe which contains the data.
`status_field`	The field in the dataframe that indicates the status, i.e., event or no event.
`time_field`	The field in the dataframe that indicates the follow-up time.
`specific_time`	The time point at which the outcome status should be calculated.

Value

`outcome`	Whether the operation was successfully performed
`message`	Any information, particularly when the operation fails.
`new_data`	The data with the existing status and time fields replaced by the updated status and time. The original status and time are available with a prefix "unmodified_".

Author(s)

Kurinchi Gurusamy

Examples

```
library(survival)
# Use the dataset colon as example
# Replace existing status with the status at 365 days
results <- get_outcome_status_at_specific_time(df = colon,
status_field = "status", time_field = "time", specific_time = 365)
results$outcome
results$message
# Display first 10 rows to show how the status and time have been modified
results$new_data[1:10, c("status", "time", "unmodified_status", "unmodified_time")]
```

`guess_data_types` *Guess data types*

Description

This function removes any columns where there is no data and makes guesses on the data type. However, this relies on the data not being coded already. If the data has been coded, the metadata generated can be used as a template that can be modified and provided as input for `process_data`.

Usage

```
guess_data_types(data_file_path)
```

Arguments

`data_file_path`	The path for the data file where guessing the data types of columns is necessary.

Value

`outcome`	Whether the operation was successfully performed
`message`	Any information, particularly when the operation fails.
`data`	The data after removing the columns without any data.
`metadata`	Automated metadata is created based on the data. However, this relies on data not previously coded, for example, if the status is coded as 0 and 1 rather than 'absent' and 'present', the variable will be recognised as a quantitative variable rather than categorical variable.
`any_type`	All fields with data.
`quantitative`	Fields recognised as quantitative.
`numerical`	Fields recognised as continuous.
`count`	Fields recognised as count. Count data is recognised from the field name. If a field name starts with 'Number of', it is considered as count data.
`categorical`	Fields recognised as categorical data.
`nominal`	Fields recognised as nominal data. All categorical data with more than two levels are recognised as nominal data.

binary	Fields recognised as binary data. All categorical data with only two levels are recognised as binary data.
ordinal	Fields recognised as ordinal data. Any categorical data with more than two levels and with the second character of all the levels being an '_' are recognised as ordinal data.
date	Fields recognised as date.
time	Fields recognised as time.

Author(s)

Kurinchi Gurusamy

See Also

process_data

Examples

```
data_file_path <- paste0(tempdir(), "/df.csv")
write.csv(penguins, data_file_path, row.names = FALSE, na = "")
guessed_data_types <- guess_data_types(data_file_path = data_file_path)
guessed_data_types
```

---

perform_analysis *Perform analysis*

Description

This uses the calculate_actual_predicted to develop and run models and calculate_performance to calculate the mean and confidence intervals of the performance (please see details below).

Usage

```
perform_analysis(generic_input_parameters,
specific_input_parameters_each_analysis, prepared_datasets, verbose)
```

Arguments

generic_input_parameters	This is a list that contains common information across models. If one or more items are missing or incorrect, this may result in error. Therefore, we recommend that you use the create_generic_input_parameters function to create this input.
specific_input_parameters_each_analysis	This corresponds to each analysis, i.e., a model or scoring system. If one or more items are missing or incorrect, this may result in error. Therefore, we recommend that you use the create_specific_input_parameters.

`prepared_datasets`
: Datasets prepared using the `prepare_datasets`.

`verbose`
: *TRUE* if the progress must be displayed and *FALSE* otherwise.

Details

Preparing datasets for each simulation Please see `prepare_datasets`.

Calculation of actual and predicted values Please see `calculate_actual_predicted`, particulary for details of apparent performance, bootstrap performance, test performance, optimism as described by Collins et al, 2024.

Calculation of performance measures Please see `calculate_performance`.

Calculation of means and confidence intervals For calculating the average performance measures and their confidence intervals across multiple simulations, appropriate transformations were performed first. After this, the bias-corrected accelerated confidence intervals were calculated based on the "bca" function from coxed package, which is not maintained anymore (R, 2025). The bias-corrected accelerated confidence intervals of the transformed data were then back transformed.

The "enhanced bootstrapping internal validation approach" method described by Collins et al., 2024 provides only the mean optimism-corrected performance. However, we have optimism from multiple simulations. Therefore, rather than calculating the average and then subtracting it from the apparent performance, the optimism from each simulation was subtracted from the apparent performance. This allowed calculation of the confidence intervals of the optimism-corrected performance using the bca function (after appropriate transformation).

The performance measures of the calibration intercept-slope adjusted models were also assessed by the same method. We have also presented the performance of the models in the 'out-of-sample subjects', i.e., the subjects who were not included in the bootstrap sample.

Value

`apparent_performance`
: Model is developed in the entire dataset and performance evaluated in the same sample.

`bootstrap_performance`
: Model is developed in a subset of data (training set) and evaluated in the training dataset

`test_performance`
: Model developed in the training set is evaluated in the entire dataset.

`out_of_sample_performance`
: Performance in the sample that was not included in the training dataset

`optimism`
: Test performance - bootstrap performance

`average_optimism`
: Average of the optimism

`optimism_corrected_performance`
: Apparent performance - average optimism

`optimism_corrected_performance_with_CI`
: Please see details above.

`out_of_sample_performance_summary`
: Please see details above.

`apparent_performance_calibration_adjusted`
: For details of calibration adjustment see `calculate_actual_predicted`

`bootstrap_performance_calibration_adjusted`
: As above

`test_performance_calibration_adjusted`
: As above

`out_of_sample_performance_calibration_adjusted`
: As abovee

`optimism_calibration_adjusted`
: As above

`average_optimism_calibration_adjusted`
: As above

`optimism_corrected_performance_calibration_adjusted`
: As above

`optimism_corrected_performance_with_CI_calibration_adjusted`
: As above

`out_of_sample_performance_summary_calibration_adjusted`
: Summary of out-of-sample performance

`apparent_performance_adjusted_mandatory_predictors_only`
: For details of this model, used only for research purposes, see `calculate_actual_predicted`, section, 'Model with with only the mandatory predictors but based on the coefficients of the entire model'.

`bootstrap_performance_adjusted_mandatory_predictors_only`
: As above

`test_performance_adjusted_mandatory_predictors_only`
: As above

`out_of_sample_performance_adjusted_mandatory_predictors_only`
: As abovee

`optimism_adjusted_mandatory_predictors_only`
: As above

`average_optimism_adjusted_mandatory_predictors_only`
: As above

`optimism_corrected_performance_adjusted_mandatory_predictors_only`
: As above

`optimism_corrected_performance_with_CI_adjusted_mandatory_predictors_only`
: As above

`out_of_sample_performance_summary_adjusted_mandatory_predictors_only`
: Summary of out-of-sample performance

`actual_predicted_results_apparent`
: Output from `calculate_actual_predicted` retained for some later calculations.

`average_lp_all_subjects`
: Output from `calculate_actual_predicted` retained for some later calculations.

Author(s)

Kurinchi Gurusamy

References

Collins GS, Dhiman P, Ma J, Schlussel MM, Archer L, Van Calster B, et al. Evaluation of clinical prediction models (part 1): from development to external validation. Bmj. 2024;384:e074819.

See Also

prepare_datasets calculate_actual_predicted calculate_performance

Examples

```
library(survival)
colon$status <- factor(as.character(colon$status))
# For testing, only 5 simulations are used here. Usually at least 300 to 500
# simulations are a minimum. Increasing the simulations leads to more reliable results.
# The default value of 2000 simulations should provide reasonably reliable results.
generic_input_parameters <- create_generic_input_parameters(
  general_title = "Prediction of colon cancer death", simulations = 5,
  simulations_per_file = 20, seed = 1, df = colon, outcome_name = "status",
  outcome_type = "time-to-event", outcome_time = "time", outcome_count = FALSE,
  verbose = FALSE)$generic_input_parameters
analysis_details <- cbind.data.frame(
  name = c('age', 'single_mandatory_predictor', 'complex_models',
           'complex_models_only_optional_predictors', 'predetermined_model_text'),
  analysis_title = c('Simple cut-off based on age', 'Single mandatory predictor (rx)',
                     'Multiple mandatory and optional predictors',
                     'Multiple optional predictors only', 'Predetermined model text'),
  develop_model = c(FALSE, TRUE, TRUE, TRUE, TRUE),
  predetermined_model_text = c(NA, NA, NA, NA,
 "cph(Surv(time, status) ~ rx * age, data = df_training_complete, x = TRUE, y = TRUE)"),
 mandatory_predictors = c(NA, 'rx', 'rx; differ; perfor; adhere; extent', NA, "rx; age"),
  optional_predictors = c(NA, NA, 'sex; age; nodes', 'rx; differ; perfor', NA),
  mandatory_interactions = c(NA, NA, 'rx; differ; extent', NA, NA),
 optional_interactions = c(NA, NA, 'perfor; adhere; sex; age; nodes', 'rx; differ', NA),
  model_threshold_method = c(NA, 'youden', 'youden', 'youden', 'youden'),
  scoring_system = c('age', NA, NA, NA, NA),
  predetermined_threshold = c('60', NA, NA, NA, NA),
  higher_values_event = c(TRUE, NA, NA, NA, NA)
)
write.csv(analysis_details, paste0(tempdir(), "/analysis_details.csv"),
          row.names = FALSE, na = "")
analysis_details_path <- paste0(tempdir(), "/analysis_details.csv")
# verbose is TRUE as default. If you do not want the outcome displayed, you can
# change this to FALSE, as shown here
results <- create_specific_input_parameters(
  generic_input_parameters = generic_input_parameters,
  analysis_details_path = analysis_details_path, verbose = FALSE)
specific_input_parameters <- results$specific_input_parameters
# Set a seed for reproducibility - Please see details above
```

```
set.seed(generic_input_parameters$seed)
prepared_datasets <- {prepare_datasets(
  df = generic_input_parameters$df,
  simulations = generic_input_parameters$simulations,
  outcome_name = generic_input_parameters$outcome_name,
  outcome_type = generic_input_parameters$outcome_type,
  outcome_time = generic_input_parameters$outcome_time,
  verbose = FALSE)}
# There is no usually no requirement to call this function directly. This is used
# by the perform_analysis function to create the actual and predicted values.
specific_input_parameters_each_analysis <- specific_input_parameters[[1]]
results <- perform_analysis(generic_input_parameters,
specific_input_parameters_each_analysis, prepared_datasets, verbose = FALSE)
results$apparent_performance
```

`prepare_datasets` *Prepare simulated datasets from the entire dataset*

Description

This takes the dataset, prepared using the `process_data` function and takes subsets of data for each simulation. For details, please see below.

Usage

```
prepare_datasets(df, simulations, outcome_name, outcome_type, outcome_time, verbose)
```

Arguments

`df`	The dataset used for the analysis. This must be provided as a dataframe. Data in files can be converted to dataframes with appropriate field types using `process_data`.
`simulations`	The number of simulations required. Usually at least 300 to 500 simulations are a minimum. Increasing the simulations leads to more reliable results. The default value of 2000 simulations should provide reasonably reliable results.
`outcome_name`	Name of the columnn that contains the outcome data. This must be a column name in the *'df'* provided as input.
`outcome_type`	One of *'binary'*, *'time-to-event'*, *'quantitative'*. Count outcomes are included in *'quantitative'* outcome type and can be differentiated from continuous outcomes by specifying outcome_count as *TRUE*. Please see examples below.
`outcome_time`	The name of the column that provides the follow-up time. This is applicable only for *'time-to-event'* outcome. For other outcome types, enter *NA*.
`verbose`	*TRUE* if the progress must be displayed and *FALSE* otherwise.

Details

Overview The input parameters are part of the generic_input_parameters created with `create_generic_input_parameters`. In the first step, it excludes all rows where the outcome is not available. For 'time-to-event' outcomes, the rows without *outcome_time* are also excluded. This forms the basis for the 'all_subjects' dataset.

In the next step, subjects are sampled from the 'all_subjects' dataset. The sampling is done using random methods. The starting point used in the random number generator is called a 'seed'. This determines all the subsequent numbers generated. The size of the sample is the same as the original data set. This is done by sampling with replacement. The subjects included in this sample are included for model development and the sample is called 'training' dataset.

Some subjects are not included in the 'training' dataset. These 'out-of-sample' subjects are used only for validation. The dataset that includes only 'out-of-sample' subjects is called 'only_validation' dataset. It must be noted that some subjects in the 'all_subjects' will be included more than once in the 'training' dataset because of the nature of the sampling.

While sampling, if all the subjects have the same outcome, this dataset is not suitable for model development. Therefore, such simulations cannot be included in the analysis and therefore, excluded. In addition to sampling, some additional processing is performed. In the 'out-of-sample' evaluation, when there are ordinal factors absent in the 'training' dataset but present in the 'out-of-sample' dataset, it results in errors. To avoid this, some levels of ordinal factors are combined.

By default, no seed is used for initiating the random sequence. This is set as part of `create_generic_input_parameters`. However, you might want to set a seed for reproducibility. In the examples, the seed is set to 1. Choosing a different seed might give slightly different results compared to seed 1.

Value

`df_training_list`
: A list of 'training' datasets, one for each simulation.

`df_only_validation_list`
: A list of datasets containing 'out-of-sample' subjects, one for each simulation.

`df_all_subjects_list`
: This is the same for all simulations.

Author(s)

Kurinchi Gurusamy

See Also

`Random`

Examples

```
library(survival)
colon$status <- factor(as.character(colon$status))
# For testing, only 5 simulations are used here. Usually at least 300 to 500
# simulations are a minimum. Increasing the simulations leads to more reliable results.
# The default value of 2000 simulations should provide reasonably reliable results.
generic_input_parameters <- create_generic_input_parameters(
```

```
  general_title = "Prediction of colon cancer death", simulations = 5,
  simulations_per_file = 20, seed = 1, df = colon, outcome_name = "status",
  outcome_type = "time-to-event", outcome_time = "time", outcome_count = FALSE,
  verbose = FALSE)$generic_input_parameters
analysis_details <- cbind.data.frame(
  name = c('age', 'single_mandatory_predictor', 'complex_models',
           'complex_models_only_optional_predictors', 'predetermined_model_text'),
  analysis_title = c('Simple cut-off based on age', 'Single mandatory predictor (rx)',
                     'Multiple mandatory and optional predictors',
                     'Multiple optional predictors only', 'Predetermined model text'),
  develop_model = c(FALSE, TRUE, TRUE, TRUE, TRUE),
  predetermined_model_text = c(NA, NA, NA, NA,
  "cph(Surv(time, status) ~ rx * age, data = df_training_complete, x = TRUE, y = TRUE)"),
  mandatory_predictors = c(NA, 'rx', 'rx; differ; perfor; adhere; extent', NA, "rx; age"),
  optional_predictors = c(NA, NA, 'sex; age; nodes', 'rx; differ; perfor', NA),
  mandatory_interactions = c(NA, NA, 'rx; differ; extent', NA, NA),
  optional_interactions = c(NA, NA, 'perfor; adhere; sex; age; nodes', 'rx; differ', NA),
  model_threshold_method = c(NA, 'youden', 'youden', 'youden', 'youden'),
  scoring_system = c('age', NA, NA, NA, NA),
  predetermined_threshold = c('60', NA, NA, NA, NA),
  higher_values_event = c(TRUE, NA, NA, NA, NA)
)
write.csv(analysis_details, paste0(tempdir(), "/analysis_details.csv"),
          row.names = FALSE, na = "")
analysis_details_path <- paste0(tempdir(), "/analysis_details.csv")
# verbose is TRUE as default. If you do not want the outcome displayed, you can
# change this to FALSE
results <- create_specific_input_parameters(
  generic_input_parameters = generic_input_parameters,
  analysis_details_path = analysis_details_path, verbose = TRUE)
specific_input_parameters <- results$specific_input_parameters
# Set a seed for reproducibility - Please see details above
set.seed(generic_input_parameters$seed)
prepared_datasets <- {prepare_datasets(
  df = generic_input_parameters$df,
  simulations = generic_input_parameters$simulations,
  outcome_name = generic_input_parameters$outcome_name,
  outcome_type = generic_input_parameters$outcome_type,
  outcome_time = generic_input_parameters$outcome_time,
  verbose = TRUE)}
```

`process_data` *Process the data*

Description

This takes a dataset and the metadata for the dataset and creates R data frames in a format required for the subsequent steps.

Usage

```
process_data(data_file_path, metadata_file_path)
```

Arguments

`data_file_path`	Path to the dataset
`metadata_file_path`	Path to the metadata file

Details

The metadata should contain the following information as a minimum. *variable*: this is the name of the variable and should match the column names of the dataset. *data_type*: *'numerical'* for continuous variables, *'count'* for count variables, *'binary* for binary categorical variables, *'nominal'* for unordered categorical variables with more than 2 levels, *'ordinal'* for ordered categorical variables, *'date'* for variables stored as date, and *'time'* for variables stored containing the time of the day.

Optional information includes the following. *reference*: Reference category for *binary* and *nominal* variables. This should be a category existing in the variable. *ordinal levels*: the levels of ordinal data from lower to higher order, separated by ";". This must include all the levels in the data.

You can use `guess_data_types` as a starting point for the metadata, which is included in the output list of the `guess_data_types` function.

Value

`outcome`	Whether the operation was successfully performed
`message`	Any information, particularly when the operation fails.
`data_processed`	The data which has been modifed according to the metadata when correct parameters are provided

.

`any_type`	All fields.
`quantitative`	Fields recognised as quantitative.
`numerical`	Fields recognised as continuous.
`count`	Fields recognised as count.
`categorical`	Fields recognised as categorical data.
`nominal`	Fields recognised as nominal data
`binary`	Fields recognised as binary data.
`ordinal`	Fields recognised as ordinal data.
`date`	Fields recognised as date.
`time`	Fields recognised as time.

Author(s)

Kurinchi Gurusamy

See Also

guess_data_types

Examples

```
library(survival)
# Use the dataset colon as example
# Select only the survival for these examples (etype == 2)
data_file_path <- paste0(tempdir(), "/df.csv")
write.csv(colon[colon$etype == 2, ], data_file_path, row.names = FALSE, na = "")
metadata <- {data.frame(
  variable = c("id","study","rx","sex","age",
               "obstruct","perfor","adhere","nodes","status",
               "differ","extent","surg","node4","time",
               "etype"),
  data_type = c("nominal", "nominal", "nominal", "binary", "numerical",
                "binary", "binary", "binary", "count", "binary",
                "ordinal", "ordinal", "binary", "binary", "numerical",
                "nominal"),
  reference = c(NA, NA, "Obs", 0, NA,
                0, 0, 0, NA, 0,
                NA, NA, 0, 0, NA,
                NA),
  ordinal_levels = c(NA, NA, NA, NA, NA,
                     NA, NA, NA, NA, NA,
                     "1;2;3", "1;2;3;4", NA, NA, NA,
                     NA),
  comments = NA
)}
metadata_file_path <- paste0(tempdir(), "/metadata.csv")
write.csv(metadata, metadata_file_path, row.names = FALSE, na = "")
processed_data <- process_data(data_file_path, metadata_file_path)
```

Index