Package ‘aba’

December 16, 2021

Title  Automated Biomarker Analysis

Version  0.0.9

Description  A tool to fit clinical prediction models and plan clinical trials using biomarker data across multiple analysis factors (groups, outcomes, predictors).

License  GPL (>= 3)

Encoding  UTF-8

LazyData  true

RoxygenNote  7.1.2

Suggests  covr, knitr, plotly, rmarkdown, testthat (>= 3.0.0), methods, tidyselect, nime, yardstick, pins, Matrix, MuMIn, broom.mixed, emmeans, ggpurv, pROC, progress, OptimalCutpoints, tableone, longpower, withr, reactable, psychometric, readr, writexl

Config/testthat/edition  3

Imports  purrr, dplyr, broom, tibble, ggplot2, glue, magrittr, stringr, rlang, tidyr, stats, generics

VignetteBuilder  knitr

Depends  R (>= 2.10)


NeedsCompilation  no

Author  Nicholas Cullen [aut, cre]

Maintainer  Nicholas Cullen <nickcullen31@gmail.com>

Repository  CRAN

Date/Publication  2021-12-16 20:50:05 UTC
R topics documented:

aba_adjust ........................................... 3
aba_control ............................................ 4
aba_demographics ..................................... 5
aba_emmeans ........................................... 7
aba_fit .................................................. 8
aba_longpower ......................................... 9
aba_model .............................................. 11
aba_plot ................................................ 13
aba_plot_coef .......................................... 14
aba_plot_metric ....................................... 15
aba_plot_roc .......................................... 17
aba_read ............................................... 18
aba_robust ............................................ 19
aba_screen ............................................ 20
aba_selection .......................................... 22
aba_summary .......................................... 24
aba_write ............................................. 25
adnimerge ............................................. 27
all_combos ............................................. 28
as_reactable .......................................... 29
as_table ................................................ 30
everyone ............................................... 31
fit.abaModel .......................................... 31
set_covariates ......................................... 32
set_data ............................................... 33
set_groups ............................................. 34
set_outcomes .......................................... 35
set_predictors ......................................... 36
set_stats .............................................. 37
stat_ancova .......................................... 39
stat_glm ................................................ 40
stat_Lm .................................................. 41
stat_lme .............................................. 43
stat_mmrm ............................................. 44
stat_retest ........................................... 46
stat_roc ............................................... 47

Index .................................................... 50
Create an aba_adjust object.

Description

Adjust the p-values (model and/or coefficients) of an abaSummary object.

Usage

aba_adjust(
  method = c("none", "bonferroni", "fdr", "hochberg", "holm", "hommel", "BH", "BY"),
  by = c("group", "outcome", "stat"),
  form = c("metric", "coef")
)

Arguments

  method          string. The method to adjust with. See p.adjust.
  by              vector. The groupings to use for adjustment. Possible choices: group, outcome, stat, predictor_set
  form            vector. Whether to adjust both metrics and coefs, or just one.

Value

an abaSummary object. The abaSummary passed to aba_adjust but with p-values changed according to how the user specified.

Examples

df <- adnmerge %>% dplyr::filter(VISCODE == 'bl')

model <- df %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
  c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)) %>%
  set_stats('glm') %>%
  fit()

# no adjustment
model_summary <- model %>% aba_summary()

# default - correct within group, outcome, and stat (x4 comparisons)
model_summary_adj <- model %>%
  aba_summary(adjust = aba_adjust(method='bonferroni'))
aba_control

Create an aba control object.

Description

The aba control which determines how an aba summary will be calculated and printed to console.

Usage

aba_control(
  include_intercept = FALSE,
  include_covariates = TRUE,
  pval_digits = 4,
  aic_digits = 0,
  metric_digits = 2,
  coef_digits = 2
)

Arguments

include_intercept
  boolean. Whether to include intercept in coefs
include_covariates
  boolean. Whether to include covariates in coefs
pval_digits
  integer. How many decimals of a p-value to show
aic_digits
  integer. How many decimals of AIC value to show
metric_digits
  integer. Default value of how many decimals to show for model metrics (e.g., auc, adj.r.squared, etc)
coef_digits
  integer. Default value of how many decimals to show for model coefficients

Value

a list with the control parameters specified
**Examples**

```r
df <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# standard example
model <- df %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(CSF_ABETA_STATUS_bl) %>%
  set_predictors(
    PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats('glm') %>%
  aba_fit()

# no control -> default
model_summary <- model %>% aba_summary()
print(model_summary)

# add a control object - don't include covariate coefficients
my_control <- aba_control(include_covariates = FALSE)
model_summary2 <- model %>% aba_summary(control = my_control)
print(model_summary2)
```

---

**aba_demographics**  
*Create a demographics table from a fitted aba model.*

**Description**

This function allows you to automatically create a demographics table from a fitted aba model. The variables in the table will be inferred from the spec of the model (predictors, covariates, outcomes, etc.), although this can be customized.

**Usage**

```r
aba_demographics(
  object,
  strata = NULL,
  include_predictors = TRUE,
  include_covariates = TRUE,
  include_outcomes = TRUE,
  add_vars = NULL,
  data_filter = NULL
)
```
Arguments

object        abaModel. The fitted aba model to create demographics table from.

strata        string (optional). How to stratify the demographics table.

include_predictors  boolean. Whether to include predictors in table.

include_covariates  boolean. Whether to include covariates in table.

include_outcomes  boolean. Whether to include outcomes in table.

add_vars    character vector (optional). Any additional variables to add to the demographics table. These variables should be present in the data from the aba model.

data_filter  logical expression (optional). If this is specified, the data from the aba model will be further filtered before the table is made.

Details

Note that support is weaker for longitudinal data right now.

Value

a TableOne object (see tableone package).

Examples

model <- aba_model() %>%
  set_data(adnimerge %>% dplyr::filter(VISCODE == 'bl')) %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(
    PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats('glm') %>%
  aba_fit()

my_table <- model %>% aba_demographics(strata = 'DX_bl')
print(my_table)
aba_emmeans

\textit{Calculated estimated marginal means.}

\textbf{Description}

This function estimates the estimated marginal means (also known as least-square means) and, if relevant, the treatment effects for mmrm, lme, and ancova models.

\textbf{Usage}

\begin{verbatim}
aba_emmeans(model)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{model} \abaModel. The fitted \aba model to run \texttt{emmeans} on.
\end{itemize}

\textbf{Details}

This function is based on the \emmeans::\texttt{emmeans} function. This function will only be run for the \texttt{stats} which are supported by \texttt{emmeans}.

\textbf{Value}

an \abaEmmeans object. This object contains the emmeans, the paired comparisons (i.e., treatment effect), and the sample size at each visit.

\textbf{Examples}

\begin{verbatim}
# process data: take first 4 visits, only MCI, use CSF abeta as "treatment",
# and create endpoint as change from baseline in cognition at each visit
df <- adnimerge %>%
dplyr::filter(
  VISCODE %in% c('bl', 'm06', 'm12', 'm24'),
  !is.na(CSF_ABETA_STATUS_bl),
  DX_bl %in% c('MCI')
) %>%
dplyr::mutate(
  TREATMENT = factor(CSF_ABETA_STATUS_bl, levels=c(0,1),
                     labels=c('Placebo','Treatment')),
  ADAS13 = ADAS13 - ADAS13_bl,
  CDRSB = CDRSB - CDRSB_bl,
  MMSE = MMSE - MMSE_bl
)

# fit mmrm model for different endpoints, adjusted for covariates
model <- df %>% aba_model() %>%
  set_outcomes(CDRSB, ADAS13, MMSE) %>%
  set_covariates(

\end{verbatim}
Fit an aba model.

Description
Calling `aba_fit` will trigger the fitting of all statistical models which have been specified for the model. This will result in fits for each group - outcome - stat combination.

Usage
`aba_fit(object, ...)`

Arguments
- `object`  
  aba model The aba model to be fitted.
- `...`  
  additional parameters.

Details
Note that this function is identical to the generic `fit()` function which is also provided for compatibility with the greater R ecosystem.

Value
`abaModel`

Examples
```r
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')
model_spec <- aba_model() %>%
  set_data(data) %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
```
aba_longpower

```r
  c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)
) %>%
  set_stats('glm')

model <- model_spec %>% aba_fit()
```

Description

This function allows users to calculate power, required sample size, or percent change (i.e., treatment effect) on a fitted aba model that uses longitudinal stats such as `stat_lme`. Whichever argument is left as NULL will be the argument for which this function solves. Users can specify a range of values for any function arguments to test different assumptions.

Usage

```r
aba_longpower(
  object,
  n = NULL,
  pct_change = NULL,
  power = NULL,
  t_length = NULL,
  t_freq = NULL,
  sig_level = 0.05,
  dropout = 0
)
```

Arguments

- **object**: aba model. The fitted aba model on which to calculate power. This model should feature a longitudinal aba stat (e.g., `stat_lme`)
- **n**: integer. The total minimum required sample size.
- **pct_change**: double between 0 - 1. The expected treatment effect which means the percent by which the slope will decrease in the treatment group. Note that "absolute" treatment effects are not currently supported.
- **power**: double between 0 - 1. The expected power to detect the treatment effect.
- **t_length**: double. The expected duration of the clinical trial. Note that this value should be on the same scale as the `time` argument used to fit the longitudinal aba stat. A longer trial will generally result in smaller sample size or higher power.
- **t_freq**: double. The expected frequency of endpoint sampling during the trial. This value should be less than `t_length`. Note that this value should also be on the same scale as the `time` argument used to fit the longitudinal aba stat.
sig_level: double between 0 - 1. The required alpha level. There is usually little reason to change this from the default of 0.05.

dropout: double between 0 - 1. The expected overall drop-out rate during the trial. Note that we handle dropout in a fairly crude way by simply multiplying the sample size by (1 + dropout) instead of, say, sampling dropout from some distribution at each visit.

Details

Power is calculated using the \texttt{lmpower} function from the \texttt{longpower} package in R. Please see documentation there to get a better understanding of the actual formulas involved.

Value

An aba longpower object which can be plotted and printed in special ways and which contains all the resulting calculations.

Examples

```r
# use only two year follow-up data; filter by some basic AD trial criteria
data <- adnimerge %>%
  dplyr::filter(
    VISCODE %in% c('bl', 'm06', 'm12', 'm24'),
    DX_bl %in% c('MCI', 'AD'),
    CDR_bl %in% c(0.5, 1),
    MMSE_bl >= 20, MMSE_bl <= 28
  )

# fit an aba model with an lme stat to get a longitudinal model
model <- data %>% aba_model() %>%
  set_outcomes(CDRSB, ADAS13) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats(stat_lme(id = 'RID', time = 'YEARS_bl')) %>%
  fit()

# summarize aba model - not necessary here but good to see results
model_summary <- model %>% summary()

# run power analysis on the fitted aba model with various assumptions
# e.g., treatment effect between 25 - 35%; power between 80 - 90%
pwr <- model %>%
  aba_longpower(
    n = NULL,
    pct_change = c(0.25, 0.30, 0.35),
    power = c(0.8, 0.85, 0.9),
    t_length = 2,
    t_freq = 0.25,
    dropout = 0.2
  )

# generate a standard results figure from the power results
```

aba_model

Create an aba model.

Description

An aba model is the foundational object in the aba package. It is composed of the following:

- data: a data.frame to be used to fit the statistical models
- spec: the specification for the aba model composed of the following:
  - groups: subsets of the data
  - outcomes: dependent variables in statistical fits.
  - covariates: independent variables which should always be included in statistical fits.
  - predictors: independent variables which will vary across different statistical fits.
- results: the resulting fitted statistics.

Usage

aba_model(
  data = NULL,
  groups = NULL,
outcomes = NULL,
predictors = NULL,
covariates = NULL,
stats = NULL,
verbose = FALSE)

Arguments

- **data**: data.frame the data to use for the object
- **groups**: vector or list of logical statements as strings. Groups are subsets of the data on which different models will be fit.
- **outcomes**: vector or list of strings Outcomes are the dependent variables in the statistical fits.
- **predictors**: vector or list of strings Predictors are independent variables which you want to vary. You can include variables on their own or in combination with others. A collection of variables is referred to as a predictor and unique variables are referred to as a term.
- **covariates**: vector of strings Covariates are independent variables which remain fixed across all statistical fits and are therefore always included with the different combinations of predictors.
- **stats**: string or abaStat object(s) with stat_ prefix. Stats are the actual statistical models which you want to fit on the data. Their primary functions are to 1) generate a suitable model formula given the outcome - covariate - predictor combination, and 2) to actually fit the statistical model.
- **verbose**: logical. Whether to give a progress bar during model fitting. This can be useful if the fitting procedure is going to take a long time.

Value

An aba model which can be fitted using the aba_fit() function and which can be modified in any manner.

Examples

```r
# use built-in data and only take the baseline visit
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# Create aba model w/ data, groups, outcomes, covariates, predictors, stats.
# Note that we start with piping the data into the aba_model... This is # possible because `data` is the first argument of the `aba_model()` function # and is useful because it gives auto-completion of variables names in Rstudio.
model <- data %>% aba_model() %>%
set_groups(everyone(), DX_bl %in% c('MCI','AD')) %>%
set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
set_covariates(AGE, GENDER, EDUCATION) %>%
set_predictors(
```

aba_plot 13

aba_plot

Plot an aba object

Description

This is a generic function for plotting an aba object. The resulting plot will depend on the type of aba object. The supported objects are the following:

- `aba_emmeans`
- `aba_longpower`
- `aba_robust`

Usage

`aba_plot(object, ...)`
Arguments

object  aba object. The object to plot.

Value

a ggplot with plotted results depending on the aba object.

Description

Plot coefficients of an aba model summary

Usage

aba_plot_coef(
  object,
  x = "term",
  group = "predictor",
  facet = c("outcome", "group"),
  coord_flip = FALSE,
  palette = "jama",
  plotly = FALSE
)

Arguments

object  an aba model summary. The object to plot - this should be the result of an
        aba_summary() call.

x         string. The model spec factor to use as the x axis. Defaults to predictor sets.

group     string. The model spec factor to use as the group variable in ggplot - this corre-
          sponding to "group", "fill", and "color" in ggplot. Defaults to outcome.

facet     string. The model spec factor to use as the group variable in ggplot - this corre-
          sponding to "facet_wrap" in ggplot. Defaults to group.

coord_flip  logical. Whether to flip the x and y axes. This can be useful when there are a
             large amount of predictor sets and you want to view metrics vertically.

palette    string. Which ggpubr palette to use. See ggpubr::set_palette.

plotly     logical. Whether to use plot.ly instead of standard ggplot. Defaults to false. 
            Using ggplotly can be useful if you want interactivity on web pages.

Value

a ggplot of the specified aba model summary coefficients
**aba_plot_metric**

**Examples**

```r
# fit aba model
model <- aba_model() %>%
  set_data(adnimerge %>%
    dplyr::filter(VISCODE == 'bl')) %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(
    PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_stats(stat_glm(std.beta=TRUE)) %>%
  fit()

# summarise aba model to calculate metrics
model_summary <- model %>% aba_summary()

# plot the coefficients using default
coeef_plot <- model_summary %>% aba_plot_coef(coord_flip=TRUE)

# compare predictor coefficients across outcomes
coeff_plot2 <- model_summary %>%
  aba_plot_coef(
    x = 'outcome', group='predictor',
    facet=c('term','group'), coord_flip=TRUE
  )
```

---

**aba_plot_metric**

Plot metrics of an aba model summary

**Description**

Plot metrics of an aba model summary

**Usage**

```r
aba_plot_metric(
  object,
  metric = NULL,
  x = "predictor",
  group = "outcome",
  facet = "group",
  coord_flip = FALSE,
  palette = "jama",
  plotly = FALSE
)
```
Arguments

- **object**: an aba model summary. The object to plot - this should be the result of an aba_summary() call.
- **metric**: string. The performance metric to plot (e.g., AIC, AUC, adj.r.squared).
- **x**: string. The model spec factor to use as the x axis. Defaults to predictor sets.
- **group**: string. The model spec factor to use as the group variable in ggplot - this corresponding to "group", "fill", and "color" in ggplot. Defaults to outcome.
- **facet**: string. The model spec factor to use as the group variable in ggplot - this corresponding to "facet_wrap" in ggplot. Defaults to group.
- **coord_flip**: logical. Whether to flip the x and y axes. This can be useful when there are a large amount of predictor sets and you want to view metrics vertically.
- **palette**: string. Which ggpubr palette to use. See ggpubr::set_palette.
- **plotly**: logical. Whether to use plot.ly instead of standard ggplot. Defaults to false. Using ggplotly can be useful if you want interactivity on web pages.

Value

a ggplot of the specified aba model summary metric.

Examples

# fit aba model
model <- aba_model() %>%
  set_data(adnimerge %>% dplyr::filter(VISCODE == 'bl')) %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(  
    PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,  
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)  
  ) %>%
  set_stats('glm') %>%
  fit()

# summarise aba model to calculate metrics
model_summary <- model %>% aba_summary()

# plot the metrics using default (defaults to AUC)
metric_plot <- model_summary %>% aba_plot_metric()

# coord flip
metric_plot2 <- model_summary %>% aba_plot_metric(coord_flip=TRUE)

# compare predictor sets within each outcome instead of the opposite
metric_plot3 <- model_summary %>%
  aba_plot_metric(x = 'outcome', group='predictor')
AbA Plot ROC

Plot ROC curves from an AbA model

Description

This function plots ROC curves across group - outcome - stat combinations and currently supports `stat_glm`.

Usage

`aba_plot_roc(object)`

Arguments

- `object` of type `abaSummary`. A summary of an AbA model with `stat_glm` type.

Value

A ggplot with roc curves for all predictor sets across each group - outcome - stat combination.

Examples

```r
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit glm model with binary outcome variables
model <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
  c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)) %>%
  set_stats(stat_glm(std.beta = TRUE)) %>%
  fit()

# summarise glm model
model_summary <- model %>% summary()

fig <- model_summary %>% aba_plot_roc()
```
aba_read  

Read an aba object from file

Description
This function allows you to read back into memory an aba object which was previously saved. This function is not relevant for loading results tables as you can just use read.csv or read_excel and the like. Note that this function essentially just wraps readRDS for reading an Rda object.

Usage
aba_read(filename)

Arguments
filename  
string. The filename where the aba object is saved.

Value
an aba object

Examples
# create temp files to save to
tmp_filename_rda <- tempfile(fileext = '.Rda')

# grab built-in data
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit a standard aba model
model <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(
    PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_stats('glm') %>%
  fit()

# create a model summary
model_summary <- model %>% aba_summary()

# save model summary as an object which can be loaded back into memory
model_summary %>% aba_write(tmp_filename_rda, format = 'object')

# load summary back to file to show it works
model_summary2 <- aba_read(tmp_filename_rda)

# delete temp files
aba_robust

Evaluate the robustness of an aba model to systematic and random error.

Description
This function allows you to test how adding bias to predictor values or how adding random error to predictor values affects the model coefficients and performance metrics (e.g., AUC, R2, etc) as a result. This function is useful when you have test-retest estimates of biomarkers and want to test what effect this has on diagnostic or prognostic modelling.

Usage
aba_robust(model, bias = NULL, variation = NULL, ntrials = 100, verbose = TRUE)

Arguments
- model: an aba model. The fitted aba model to perform robustness analysis on.
- bias: double or list of doubles. If one value is given, this is the percent value added or subtracted to all predictor values at each trial. If this is a list, the names of the list should be the predictors to apply bias to and the values should be the bias to apply to each predictor.
- variation: double or list of doubles. This is the percent value which represents the standard deviation of a normal distribution. The random error values will be randomly sampled from this normal distribution for each data row (participant) at each trial.
- ntrials: integer. Number of trials to run. A trial represents a different random sampling of the variation distribution. This does not have any effect for bias because the bias value is always the same.
- verbose: logical. Whether to include a progress bar to track trials.

Value
an abaRobust object which contains results from the robustness analysis that displays how model coefficients and metrics changed when bias and variation was injected into the predictors.

Examples

```r
# read and process data
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit a standard model to predict a binary outcome
model <- data %>% aba_model() %>%
```
set_groups(everyone()) %>%
set_outcomes(CSF_ABETA_STATUS_bl) %>%
set_predictors(PLASMA_PTAU181_bl, PLASMA_NFL_bl) %>%
set_stats(stat_roc(method='Youden', direction = '<')) %>%
aba_fit()

# summarise model (these are the original results)
model_summary <- model %>% aba_summary()

# specify test-retest variation for predictors (defined as percent change)
# this can be theoretical values (e.g. 5, 10, 15, 20) or derived from
test-retest studies where you measured the biomarkers twice
variation <- list('PLASMA_PTAU181_bl' = 9.5,
                  'PLASMA_NFL_bl' = 20.2)

# test robustness of the fitted aba model to this robustness
model_robust <- model %>%
               aba_robust(variation = variation,
                           ntrials = 10,
                           verbose = TRUE)

# plot results using the generic plot function
fig <- model_robust %>% aba_plot_metric()

---

**aba_screen**  
*Create an aba screen object.*

**Description**

This function runs a clinical trial screening analysis based on a fitted aba model with glm stats. You can supply different inclusion thresholds which represent predicted probabilities from the glm stats, and you can also supply cost multipliers and the required sample size in order to perform a cost-benefit analysis. This analysis uses bootstrap sampling to generate confidence intervals.

**Usage**

```r
aba_screen(
  object,
  threshold,
  cost_multiplier,
  include_n,
  ntrials = 100,
  verbose = TRUE
)
```
Arguments

object an aba model. The fitted aba model which you want to use as the screening algorithm.

threshold double or vector of doubles between 0 and 1. The threshold represents the percentage of individuals who will be invited to take the inclusion test. Note that the threshold value is calculated in a relative manner based on the values in the data population, not based on an absolute risk value.

cost_multiplier double or vector of doubles. The cost multiplier represents how much more expensive it is to perform the main inclusion test versus the screening test. Larger values mean that the main inclusion test/biomarker is much more expensive than the screening test and will therefore result in larger cost savings by using the screening model to identify individuals who are at low risk to be positive on the main inclusion test.

include_n integer. The number of participants who you expect to be included in the clinical trial. This is therefore the number of individuals who must pass the screening test and who then must pass the main inclusion test.

ntrials integer. The number of bootstrap trials to run in order to generate the confidence interval.

verbose logical. Whether to show a progress bar for each trial.

Value

an abaScreen object

Examples

# use built-in data
df <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# first, fit an aba model to predict amyloid PET status from plasma markers
# In this scenario, PET is the "inclusion" marker and plasma is the
# "screening" marker. PET is expensive and plasma is cheap, so we want to
# use plasma markers to decide who should undergo PET scans in order to
# minimize the risk of negative (i.e., wasted) PET scans.
model <- df %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(PET_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_PTAU181_bl, PLASMA_NFL_bl, c(PLASMA_PTAU181_bl, PLASMA_NFL_bl)) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats('glm') %>%
  fit()

# summarise the model just to show the plasma biomarkers do in fact
aba_selection

Run model selection on an aba model.

Description

This function allows you to run model selection on a fitted aba model. The function supports both forward and backward selection algorithms, both AIC and p-value as selection criteria, and arbitrary thresholds.

Usage

aba_selection(
  model,
  method = c("forward", "backward"),
  criteria = c("aic", "pval"),
  threshold = NULL,
  verbose = FALSE
)

Arguments

  model  abaModel. The fitted aba model to run selection on.
  method string. The selection algorithm to use (forward or backward).
  criteria string. Which metric to use when selecting the next model (aic or pval).
  threshold numeric. Which threshold to use for the selected metric (defaults to -2 for aic; defaults to 0.1 for pval).
  verbose logical. Whether to print out results of each selection round.
aba_selection

Details

Forward selection starts from covariates-only and tests the addition of all predictor sets individually, then adds the predictor set which improves the model criteria the most. Backward selection starts from the inclusion of all covariates + predictor sets and tests the removal of all predictor sets individually, then removes the predictor set which improves the model criteria the most. If there are no predictor sets whose addition/removal results in an improvement in the selected criteria by a value at least as good as the selected threshold, then the selection stops and the current model is frozen. Also, note that the model selection procedure is run separately for each group - outcome - stat combination.

Value

an abaSelection object which contains model summary information such as coefficients and metrics for each selection round across the different groups/outcomes/stats.

Examples

def <- aba::adnimerge %>% dplyr::filter(VISCODE == 'bl')

# standard model selection
model <- df %>% aba_model() %>%
  set_outcomes(ConvertedToAlzheimers) %>%
  set_predictors(
    CDRSB_bl, ADAS13_bl, MMSE_bl,
    CSF_ABETA_bl, CSF_PTAU_bl, CSF_TAU_bl,
    PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    MRI_HIPP_bl,
    PET_ABETA_bl
  ) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats('glm') %>%
  aba_fit()

model_summary <- model %>% aba_summary()

# default selection - forward selection by AIC with threshold = -2
model_selection <- model %>% aba_selection(verbsoe=TRUE)

# selection with p-value and threshold = 0.1
model_selection <- model %>%
  aba_selection(criteria = 'pval', threshold=0.1, verbose=TRUE)

# selection by group
model2 <- model %>%
  set_predictors(
    c(CDRSB_bl,ADAS13_bl,MMSE_bl),
    c(CSF_ABETA_bl,CSF_PTAU_bl,CSF_TAU_bl),
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl),
    c(MRI_HIPP_bl, PET_ABETA_bl)
  )
aba_summary

Summarise a fitted aba model.

Description

This function concisely summarises coefficients and metrics for the stat fits from the different group - outcome - stat combinations. This is the primary function to use if you want to see the results of a fitted aba model. It is also the way to generate publication-ready tables of model results.

Usage

aba_summary(
  object,
  control = aba_control(),
  adjust = aba_adjust(),
  verbose = FALSE
)

Arguments

object  abaModel. The fitted aba model which you want to summarise.
control abaControl. An aba control object which allows users to customize the summary process – e.g., whether to include covariates in the table.
adjust abaAdjust. An aba adjust object which allows users to specify p-value adjustment using a variety of methods and across arbitrary model factors.
verbose logical. Whether to provide a progress bar to track status.

Value

an abaSummary object which contains coefficients and metrics from the different statistical fits summarised into publication-ready tables.

Examples

# use built-in data
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit an aba model
model <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(PET_ABETA_STATUS_b1) %>%
  set_predictors(PLASMA_PTAU181_b1, PLASMA_NFL_b1, c(PLASMA_PTAU181_b1, PLASMA_NFL_b1)) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats('glm') %>%
  fit()

# default aba summary
model_summary <- model %>% aba_summary()

# create an aba control object to customize the summary
my_control <- aba_control(include_covariates = FALSE)

# summarise model with th custom aba control - notice covariates
# wont be included in the tables when you print the summary to console
model_summary2 <- model %>% aba_summary(control = my_control)

aba_write

Write an aba object to file.
Description

This is a generic function for writing an aba object to file. Objects can be written to file as a "table" (formatted), as "raw" (long-form results), or as an "object" (actual aba object).

Usage

aba_write(
  object,
  filename,
  format = c("table", "raw", "object"),
  separate = FALSE
)

Arguments

object an aba object. The object to save to file.
filename string. The filename to save to. Supported extensions include "csv", "xls", and "xlsx".
format string. How to save the object to file. Options include "table" (formatted results like you see when you print the object to the console), "raw" (long-form results like what you see when you call object$results), or "object" (the actual aba object which can be later be loaded into memory and used again).
separate logical. Whether to save the results in separate files (for csv) or separate sheets (for excel) based on group - outcome - stat combinations. This argument is ignored if format == "object".

Value

N/A

Examples

# create temp files to save to
tmp_filename_csv <- tempfile(fileext = '.csv')
tmp_filename_rda <- tempfile(fileext = '.Rda')

# grab built-in data
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit model
model <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)) %>%
  set_stats('glm') %>%
  aba_write(object, filename, format = c("table", "raw", "object"), separate = FALSE)
```

fit()

# summarise model
model_summary <- model %>% summary()

# save model summary to file as table
model_summary %>% aba_write(tmp_filename_csv)

# save model summary to file as raw long-form results
model_summary %>% aba_write(tmp_filename_csv, format = 'raw')

# save model summary as an object which can be loaded back into memory
model_summary %>% aba_write(tmp_filename_rda, format = 'object')

# load summary back to file to show it works
model_summary2 <- aba_read(tmp_filename_rda)
print(model_summary2)

# delete temp files
removed <- file.remove(tmp_filename_csv)
removed <- file.remove(tmp_filename_rda)
```

---

**adnimerge**

*A sample of ADNI data in long format*

**Description**

A sample of ADNI data in long format. This data contains longitudinal follow-up visits from CU and MCI patients, with baseline measures of plasma biomarkers (abeta, p-tau, nfl), CSF biomarkers, MRI measures, cognition, and demographics. Longitudinal outcomes include cognition (MMSE, CDRSB) and conversion to AD dementia and conversion to all-cause dementia. There is also binarized biomarker status for many of the variables.

**Usage**

`adnimerge`

**Format**

A data frame with 15,598 rows and 28 variables.

**Source**

[https://adni.loni.usc.edu](https://adni.loni.usc.edu)
all_combos

Create all possible combinations of a set of variables

Description

This function creates all possible combinations of a set of variables. The variables should be given as strings and sep. This function can be used inside of a call to set_predictors when creating an aba model.

Usage

all_combos(...)

Arguments

... strings. Variable names from which all possible combinations will be created. Each variable string should be separated by a comma.

Value

A list of vectors of all possible combinations of the variables

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit model with all combinations of three variables
model <- data %>% aba_model() %>%
  set_groups(  
    everyone(),
    DX_bl %in% c('MCI', 'AD')
  ) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(  
    all_combos('PLASMA_ABETA_bl', 'PLASMA_PTAU181_bl', 'PLASMA_NFL_bl')
  ) %>%
  set_stats(  
    stat_glm(std.beta = TRUE)
  ) %>%
  fit()

model_summary <- model %>% aba_summary()
**as_reactable**

*Convert an aba summary to a interactive react table*

**Description**

This function allows you to format an aba summary in the same way which it is printed to the console using the `print` function. And then it will be converted to an interactive react table that can be explored in the Rstudio viewer or in a Shiny app.

**Usage**

```r
as_reactable(object)
```

**Arguments**

- `object` abaSummary. The aba summary to format as a reacttable.

**Value**

A reactable object from the reactable package

**Examples**

```r
# use built-in data
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit an aba model
model <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(PET_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_PTAU181_bl, PLASMA_NFL_bl, c(PLASMA_PTAU181_bl, PLASMA_NFL_bl)) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats('glm') %>%
  fit()

# default aba summary
model_summary <- model %>% aba_summary()

# convert summary to table
my_table <- model_summary %>% as_reactable()
```
as_table

Convert an aba summary to a nicely formatted table

Description

This function allows you to format an aba summary in the same way which it is printed to the console using the print function. However, only one dataframe will result (i.e., the tables will not be split by group - outcome - stat combinations).

Usage

as_table(object)

Arguments

object       abaSummary. The aba summary to format as a table.

Value

a tibble

Examples

# use built-in data
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit an aba model
model <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(PET_ABETA_STATUS_bl) %>%
  set_predictors(
    PLASMA_PTAU181_bl,
    PLASMA_NFL_bl,
    c(PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats('glm') %>%
  fit()

# default aba summary
model_summary <- model %>% aba_summary()

# convert summary to table
my_table <- model_summary %>% as_table()
everyone

Use all data rows as a group in an aba model.

Description
This is a helper function which allows you to specify a group in an aba model that does not have any filtering conditions. This is useful when you want to specify an aba model with one sub-group of the data but also want to fit models on the entire data. This function is really only necessary to be used instead of a call to set_groups when building an aba model.

Usage
everyone()

Value
This function actually just returns a value of TRUE.

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit model with one subgroup (DX_bl) and also the entire data
model <- data %>% aba_model() %>%
  set_groups(
    everyone(),
    DX_bl %in% c('MCI', 'AD')
  ) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl) %>%
  set_stats( stat_glm(std.beta = TRUE) ) %>%
  fit()

model_summary <- model %>% aba_summary()

fit.abaModel
Fit an aba model.

Description
Calling fit will trigger the fitting of all statistical models which have been specified for the model. This will result in fits for each group - outcome - stat combination.
set_covariates

Usage

## S3 method for class 'abaModel'
fit(object, ...)

Arguments

object       aba model The aba model to be fitted.
...           additional parameters.

Details

Note that this function is identical to the generic `aba_fit()` function.

Value

abaModel

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

model_spec <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(
    PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_stats('glm')

model <- model_spec %>% aba_fit()

Description

Covariates are the independent variables which you want to always be included in your statistical models - regardless of the groups, outcomes, or predictors. Only one set of covariates can be supplied. If you want to test multiple sets of covariates, then you should specify them as predictors or you should create a new, separate model. This function supports both string inputs and actual variables. The inputs should be separated by a comma, where all variables together is the single covariate set.

Usage

set_covariates(object, ...)

Set the covariates of an aba model.
set_data

Arguments

object

an aba model. The model for which you want to set covariates.

... strings or variables. This comma-separated collection of values will become the single set of covariates. If you supply actual variables, then the data of the aba model should already be set.

Value

An aba model with covariates set.

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# set with variables
model <- aba_model() %>%
  set_data(data) %>%
  set_covariates(AGE, GENDER, EDUCATION)

# supply strings - data does not need to be set first here. But it will # result in an error if these variables do not exist in the eventual data.
model <- aba_model() %>%
  set_covariates('AGE', 'GENDER', 'EDUCATION')

---

code

Description

The raw data will be used to fit all of the statistical models. This data will be processed according to what is specified in the aba stat objects.

Usage

set_data(model, data)

Arguments

model an aba model. The model on which data will be set.

data dataframe or tibble. The data to set.

Value

An aba model with data set.
Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# set data in the traditional way
model <- aba_model() %>% set_data(data)

# pipe data into an `aba_model()` call to get access to auto-completion on
# variables from RStudio upon further pipes. This is useful for setting
# other specs because it will reduce the chance of typos on variable names.
model <- data %>% aba_model()

Description

Groups are the filtered subsets of data which you want to fit statistical models on. This function supports both string inputs and logical functions of variables (provided that the data is already set for the aba model). The inputs should be separated by a comma, where each input is a different group. You can also specify labels for each group.

Usage

set_groups(object, ..., labels = NULL)

Arguments

object An aba model. The model for which you want to set groups.

... comma-separated strings or logical expressions. This specifies the subsets of the
data by which the aba model will filter.

labels vector of strings. Optional labels for printing & plotting.

Details

Note that `everyone()` or "everyone()" can be used to specify a group with no filtering. This can be useful when you want to fit models on the entire group and on a sub-group.

Value

An aba model with groups set to the given input.
Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# set groups based on logical expressions. Here, data must be supplied first.
model <- data %>% aba_model() %>%
  set_groups(
    everyone(),
    DX_bl == 'CU',
    (DX_bl %in% c('MCI', 'AD')) & (CSF_ABETA_bl < 880)
  )
print(model)

# set groups based on logical expressions. Here, data must be supplied first.
model <- data %>% aba_model() %>%
  set_groups(
    everyone(),
    DX_bl == 'CU',
    (DX_bl %in% c('MCI', 'AD')) & (CSF_ABETA_bl < 880),
    labels = c('All participants', 'CU-only', 'Ab+ MCI & AD')
  )
print(model)

# set groups based on strings. No data is required to be supplied first.
model <- aba_model() %>%
  set_groups(
    "everyone()",
    "DX_bl == 'CU'",
    "(DX_bl %in% c('MCI', 'AD')) & (CSF_ABETA_bl < 880)"
  )
print(model)

set_outcomes

Set the outcomes of an aba model.

Description

Outcomes are the dependent variables of the statistical models. This function supports both string inputs and actual variables as found in tidy-selection. The inputs should be separated by a comma, where each input is a different outcome You can also specify labels for each outcome.

Usage

set_outcomes(object, ..., labels = NULL)

Arguments

object An aba model. The model for which you want to set outcomes
... strings or variables. Each comma-separated value will be a new outcome. If you give variables, then the data of the aba model should already be set.

labels vector of strings. Optional labels for printing & plotting.

Value

An aba model with outcomes set.

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# set with variables
model <- aba_model() %>%
  set_data(data) %>%
  set_outcomes(CDRSB, ADAS13, MMSE)

# supply labels
model <- aba_model() %>%
  set_data(data) %>%
  set_outcomes(CDRSB, ADAS13, MMSE, labels=c('CDR-SB', 'ADAS-13', 'MMSE'))

# supply strings - data does not need to be set first here. But it will # result in an error if these variables do not exist in the eventual data.
model <- aba_model() %>%
  set_outcomes('CDRSB', 'ADAS13', 'MMSE')

---

**set_predictors**

Set the predictors of an aba model.

Description

Predictors are the independent variables which you want to vary as a factor in your statistical models across different groups, outcomes, and stats. Predictors can be supplied as individual variables or as collections of variables, so we refer to a unit of predictors as a "predictor". This function supports both string inputs and actual variables. This function also supports tidy-selection functions like contains and starts_with which allows convenient selection of many variables at once with common names.

Usage

```
set_predictors(object, ..., labels = NULL)
```

Arguments

- **object** An aba model. The model for which you want to set predictors
- **...** strings or variables or tidy-selection functions. Each comma-separated value will be a new predictor set. If you supply actual variables, then the data of the aba model should already be set.
- **labels** vector of strings. Optional labels for printing & plotting.
Value

An aba model with predictors set.

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# set with variables - this will result in four "predictor sets".
model <- aba_model() %>%
  set_data(data) %>%
  set_predictors(
    PLASMA_ABETA_bl, 
    PLASMA_PTAU181_bl, 
    PLASMA_NFL_bl, 
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  )

# set with tidy selection functions - but this is only one "predictor set", 
# not multiple individual predictor sets.
model <- aba_model() %>%
  set_data(data) %>%
  set_predictors(
    starts_with('PLASMA')
  )

# automatically generate all possible combinations of variables
model <- aba_model() %>%
  set_data(data) %>%
  set_predictors(
    all_combos(c('PLASMA_ABETA_bl', 'PLASMA_PTAU181_bl', 'PLASMA_NFL_bl'))
  )

# supply strings - data does not need to be set first here. But it will 
# result in an error if these variables do not exist in the eventual data.
model <- aba_model() %>%
  set_data(data) %>%
  set_predictors(
    'PLASMA_ABETA_bl', 
    'PLASMA_PTAU181_bl', 
    'PLASMA_NFL_bl', 
    c('PLASMA_ABETA_bl', 'PLASMA_PTAU181_bl', 'PLASMA_NFL_bl')
  )

set_stats

Set the stats of an aba model
set_stats

Description
Stats are the objects which specify 1) how model formulas should be created from the model specification, and 2) how to actual fit statistical models. Stats also have their own parameters which you can specify to change how the stat is fit. Multiple stats can be specified for an aba model. The best way to see all the available stats is the type `aba::stat_` in the console and look at the auto-completion.

Usage
```r
set_stats(.model, ..., labels = NULL)
```

Arguments
- `.model`: an aba model. The model on which to set stats.
- `...`: strings or aba stat object. Each comma-separated value will be a different stat. If you specify a string, then the default stat params will be used. Some stats require that you actually call them (e.g. `stat_lme`) because they require other parameters like `id` and `time` variables.
- `labels`: vector of strings. Labels for printing & plotting.

Details
There is a broad collection of stats implemented in aba which we plan to add to. Please feel free to request more. Also, there are certain extra parameters which are common to all stats. These include `std.beta` which determines whether to z-score all variables prior to model fitting, and `complete.cases` which determines whether to only use individuals with all available data within each group - outcome but across all predictor sets.

Value
An abaModel object with stats sets.

Examples
```r
# create default stat object by specifying only a string
model <- aba_model() %>%
  set_stats('glm')

# pass an actual stat object. This is useful to specify extra params
# such as `std.beta` and `complete.cases` which is common to all stats.
model <- aba_model() %>%
  set_stats(
    stat_glm(std.beta = TRUE, complete.cases = FALSE)
  )

# some stats such as `lme` require parameters
# those variables are expected to exist in the eventual data
model <- aba_model() %>%
  set_stats(

stat_ancova

stat_lme(id = 'RID', time = 'YEARS_bl')

# you can see these extra stat params when you print the model
print(model)

---

stat_ancova

Create an ancova stat object.

Description

This function creates an ancova stat object which can be passed as input to the set_stats() function when building an aba model. This stat performs a traditional ancova analysis using the lm function with change in endpoint as outcome, adjustment for baseline covariates and baseline outcome, and also a treatment variable whose effect on the endpoint you care about.

Usage

stat_ancova(
  treatment,
  baseline_suffix = "bl",
  std.beta = FALSE,
  complete_cases = TRUE
)

Arguments

treatment string. The treatment variable whose effect on the outcome you care about. If you don’t have a treatment, then you can just use stat_lm with a change variable as the outcome.

baseline_suffix string. The suffix to add to each outcome variable in order to pick up the associated baseline variable. You must adjust for the baseline outcome in ancova, and there is no other way to specify a different predictor for each outcome. So if the outcomes are e.g. "CDRSB" and "MMSE", then a baseline_suffix of "bl" will mean that each ancova fit with "CDRSB" as outcome will have "CDRSB_bl" added to the formula and every fit with "MMSE" as outcome will have "MMSE_bl" added. This means that these baseline variables must actually exist in the data!

std.beta logical. Whether to standardize model predictors and covariates prior to analysis.

complete_cases logical. Whether to only include the subset of data with no missing data for any of the outcomes, predictors, or covariates. Note that complete cases are considering within each group - outcome combination but across all predictor sets.
Value

An abaStat object with ancova stat type.

Examples

```r
# filter to 24 month visit; calculate change in cognition to use as outcome;
# assume abeta status as "treatment" variable.
# The goal is to see if "treatment" has an effect on 2y cognitive decline
data <- adnimerge %>%
  dplyr::filter(
    VISCODE == 'm24',
    DX_bl %in% c('MCI', 'AD'),
    !is.na(CSF_ABETA_STATUS_bl)
  ) %>%
  dplyr::mutate(
    CDRSB = CDRSB - CDRSB_bl,
    ADAS13 = ADAS13 - ADAS13_bl,
    TREATMENT = factor(CSF_ABETA_STATUS_bl, levels=c(1,0),
                        labels=c('Placebo','Treatment'))
)

# fit model. note that baseline outcome will be added based on the suffix.
# e.g., fits with "CDRSB" as outcome will also add "CDRSB_bl" to the formula.
ancova_model <- data %>% aba_model() %>%
  set_outcomes(CDRSB, ADAS13) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats(
    stat_ancova(treatment = 'TREATMENT', baseline_suffix = 'bl')
  ) %>%
  fit()

# summarise model. treatment effect will be shown in the treatment coefficient
ancova_summary <- ancova_model %>% summary()
```

---

**stat_glm**

Create a glm stat object.

**Description**

This function creates a glm stat object which can be passed as input to the set_stats() function when building an aba model. This stat performs a traditional logistic regression analysis using the glm function with a binary outcome. Coefficients will be presented as odds ratios. Default metrics include AUC.

**Usage**

```r
stat_glm(std.beta = FALSE, complete.cases = TRUE)
```
Arguments

**std.beta**

logical. Whether to standardize model predictors and covariates prior to analysis.

**complete.cases**

logical. Whether to only include the subset of data with no missing data for any of the outcomes, predictors, or covariates. Note that complete cases are considering within each group-outcome combination but across all predictor sets.

Value

An abaStat object with glm stat type.

Examples

```r
data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit glm model with binary outcome variables
model <- data %>% aba_model() %>%
  set_groups(everyone()) %>%
  set_outcomes(ConvertedToAlzheimers, CSF_ABETA_STATUS_bl) %>%
  set_predictors(
    PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_stats(
    stat_glm(std.beta = TRUE)
  ) %>%
  fit()

# summarise glm model
model_summary <- model %>% summary()

# plot glm results
fig1 <- model_summary %>% aba_plot_coef()
fig2 <- model_summary %>% aba_plot_metric()
fig3 <- model_summary %>% aba_plot_roc()
```

Description

This function creates a lm stat object which can be passed as input to the `set_stats()` function when building an aba model. This stat performs a traditional linear regression analysis using the lm function. Coefficients will be presented as beta coefficients. Default metrics include adjusted R2.
Usage

stat_lm(std.beta = FALSE, complete.cases = TRUE)

Arguments

std.beta logical. Whether to standardize model predictors and covariates prior to analysis.

complete.cases logical. Whether to only include the subset of data with no missing data for any of the outcomes, predictors, or covariates. Note that complete cases are considering within each group - outcome combination but across all predictor sets.

Value

An abaStat object with lm stat type.

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit lm model with continuous outcome variables
model <- data %>% aba_model() %>%
  set_groups(
    everyone(),
    DX_bl %in% c('MCI', 'AD')
  ) %>%
  set_outcomes(CDRSB_bl, MMSE_bl) %>%
  set_predictors(
    PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl,
    c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)
  ) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats(
    stat_lm(std.beta = TRUE)
  ) %>%
  fit()

# summarise model
model_summary <- model %>% summary()

# plot results
fig1 <- model_summary %>% aba_plot_coef()
fig2 <- model_summary %>% aba_plot_metric()
stat_lme

Create an lme stat object.

Description

This function creates an lme stat object which can be passed as input to the set_stats() function when building an aba model. This stat performs a linear mixed effects model analysis using the lme function from the nlme package. Please note that the default mode is to include an interaction term between the time variable and each predictor - i.e., time*predictor will be in the model formula - but this does not happen for covariates. Also, this model fits random intercepts and random slopes. The data for this model should be in long format with one row per subject-visit.

Usage

stat_lme(id, time, std.beta = FALSE, complete.cases = TRUE)

Arguments

id string. This is the variable in the data which represents the subject id to be used for random intercepts and random slopes.

time string. This is the time variable in the data which represents the time from baseline that the visit occurred.

std.beta logical. Whether to standardize model predictors and covariates prior to analysis.

complete.cases logical. Whether to only include the subset of data with no missing data for any of the outcomes, predictors, or covariates. Note that complete cases are considering within each group - outcome combination but across all predictor sets.

Value

An abaStat object with lme stat type.

Examples

data <- adnimerge %>%
dplyr::filter(VISCODE %in% c('bl', 'm06', 'm12', 'm24'))

model <- data %>% aba_model() %>%
  set_groups(
    everyone(),
    DX_bl %in% c('MCI', 'AD')
  ) %>%
  set_outcomes(CDRSB, ADAS13) %>%
  set_predictors(
    PLASMA_ABETA_bl,
```r
PLASMA_PTAU181_bl,
PLASMA_NFL_bl,
c(PLASMA_ABETA_bl, PLASMA_PTAU181_bl, PLASMA_NFL_bl)
) %>%
set_covariates(AGE, GENDER, EDUCATION) %>%
set_stats(
  stat_lme(id = 'RID', time = 'YEARS_bl')
) %>%
fit()

model_summary <- model %>% aba_summary()
```

---

**stat_mmmr**

Create an mmmr stat object.

**Description**

This function creates an mmmr stat object which can be passed as input to the set_stats() function when building an aba model. This stat performs a MMRM analysis using the gls function from the nlme package. Please note that the default mode is to include an interaction term between the time variable and each predictor - i.e., time*predictor will be in the model formula - but this does not happen for covariates. The data for this model should be in long format with one row per subject-visit.

**Usage**

```r
stat_mmmr(
  id, 
  time, 
  treatment = NULL, 
  baseline_suffix = "bl", 
  std.beta = FALSE, 
  complete.cases = TRUE
)
```

**Arguments**

- **id**  
  string. This is the variable in the data which represents the subject id to be used for random intercepts and random slopes.

- **time**  
  string. This is the time variable in the data which represents the time from baseline that the visit occurred. This should be a categorical variable or a continuous variable where the values are shared by all subjects. The fact that time visits should be common across all subjects is a major operational difference from stat_lme, among other differences.

- **treatment**  
  string. The treatment variable whose effect on the outcome you care about. This is useful for aba_emmeans and other functions.
baseline_suffix
string. The suffix to add to each outcome variable in order to pick up the associated baseline variable. You must adjust for the baseline outcome in mmrm, and there is no other way to specify a different predictor for each outcome. So if the outcomes are e.g. "CDRSB" and "MMSE", then a baseline_suffix of "bl" will mean that each mmrm fit with "CDRSB" as outcome will have "CDRSB_bl" added to the formula and every fit with "MMSE" as outcome will have "MMSE_bl" added. This means that these baseline variables must actually exist in the data. Also, there will always be an interaction between the baseline outcome variable and the time variable.

std.beta
logical. Whether to standardize model predictors and covariates prior to analysis.

complete.cases
logical. Whether to only include the subset of data with no missing data for any of the outcomes, predictors, or covariates. Note that complete cases are considering within each group - outcome combination but across all predictor sets.

Value
An abaStat object with mmrm stat type.

Examples

data <- adnimerge %>%
dplyr::filter(VISCODE %in% c('bl','m06','m12','m24'))

model <- data %>% aba_model() %>%
  set_groups(
    DX_bl %in% c('MCI', 'AD')
  ) %>%
  set_outcomes(CDRSB, ADAS13) %>%
  set_predictors(
    PLASMA_ABETA_bl,
    PLASMA_PTAU181_bl,
    PLASMA_NFL_bl
  ) %>%
  set_covariates(AGE, GENDER, EDUCATION) %>%
  set_stats(
    stat_mmrm(id = 'RID', time = 'VISCODE')
  ) %>%
  fit()

model_summary <- model %>% aba_summary()
stat_retest

Create a retest stat object.

Description

This function creates a retest stat object which can be passed as input to the set_stats() function when building an aba model. This stat performs a test-retest analysis on data in long format. It can be used to calculate the bias and variance of biomarkers (or any variables, for that matter) when measured multiple times. Moreover, the result of a model fit with this stat can be subsequently passed to the aba_robust() object in order to test the effect of test-retest bias/variance on clinical prediction models which can also be fit as an aba model.

Usage

```r
stat_retest(
  id,
  time,
  method = c("percent_change"),
  std.beta = FALSE,
  complete.cases = FALSE
)
```

Arguments

- **id**: string. This is the subject id variable in the dataset. This is necessary to keep track of which values belong to which individuals.
- **time**: string. This is the time variable in the dataset. This is necessary to keep track of which values belong to which time point.
- **method**: string. This is the method used to calculate the difference between outcome values across time points. Options are: percent_change calculated by 100 * (x - y) / y where x is the earlier time and y is the later time.
- **std.beta**: logical. Whether to standardize the model outcomes and predictors/outcomes prior to analysis.
- **complete.cases**: logical. Whether to only include the subset of data with no missing data for any of the outcomes, predictors, or covariates. Note that complete cases are considering within each group - outcome combination but across all predictor sets.

Value

An abaStat object with retest stat type.
Examples

```r
# use longitudinal data in healthy controls as pseudo "test-retest"
data <- adnimerge %>%
  dplyr::filter(
    VISCODE %in% c('bl', 'm06', 'm12'),
    DX_bl == 'CU'
  )

# fit model over two groups and two endpoints
model <- data %>% aba_model() %>%
  set_groups(
    everyone(),
    CSF_ABETA_STATUS_bl == 1,
    labels = c('CU', 'CU AB-')
  )
  set_outcomes(
    ADAS13, MMSE,
    labels = c('ADAS13', 'MMSE')
  )
  set_stats(
    stat_retest(id = 'RID', time = 'VISCODE')
  )
  aba_fit()

# summarise model to get bias and variance estimates
model_summary <- model %>% aba_summary()

# plot model results like any other summary
g <- model_summary %>% aba_plot_coef(
  x = 'term', group = 'group', facet = c('outcome', 'predictor'), coord_flip = TRUE
)
```

stat_roc

Create a roc stat object.

Description

This function creates a roc stat object which can be passed as input to the `set_stats()` function when building an aba model. This stat performs a traditional ROC / cutpoint analysis from a binary outcome using the `optimal.cutpoints` function from the `OptimalCutpoints` package. Note that outcomes for this model should be binary and coded as 0 = healthy and 1 = disease. Coefficients will be presented as the optimal cutpoint for the model derived from Youden’s index (or whatever method is specified). Default metrics include AUC.

Usage

```r
stat_roc(
  direction = "<",
  method = "Youden",
)```
Arguments

- **direction** ('<' or '>'). Which direction to interpret as being further from the healthy value. '<' is the default value and is interpreted as increasing predictor values are worse. '>' is therefore interpreted as higher predictor values are closer to healthy (outcome value of 0).
- **method** (string). Which method to use to calculate the optimal cutoff value. See the `OptimalCutpoints::optimal.cutpoints` function for more info.
- **std.beta** (logical). Whether to standardize model predictors and covariates prior to analysis.
- **complete.cases** (logical). Whether to only include the subset of data with no missing data for any of the outcomes, predictors, or covariates. Note that complete cases are considering within each group - outcome combination but across all predictor sets.

Value

An abaStat object with glm stat type.

Examples

data <- adnimerge %>% dplyr::filter(VISCODE == 'bl')

# fit a roc model to predict a binary outcome
model <- data %>% aba_model() %>%
  set_groups(
    everyone(),
    DX_bl %in% c('MCI', 'AD')
  ) %>%
  set_outcomes(CSF_ABETA_STATUS_bl) %>%
  set_predictors(PLASMA_PTAU181_bl, PLASMA_NFL_bl) %>%
  set_stats(
    stat_roc(method='Youden')
  ) %>%
  fit()

# summarise model
model_summary <- model %>% summary()

# if using predictors where higher values are better, then flip direction
model2 <- model %>%
  set_predictors(PLASMA_ABETA_bl) %>%
  set_stats(
    stat_roc(direction = '->')
  ) %>%
  fit()
fit()
model2_summary <- model2 %>% aba_summary()
Index

* datasets
  adnmerge, 27
  aba_adjust, 3
  aba_control, 4
  aba_demographics, 5
  aba_emmeans, 7
  aba_fit, 8
  aba_longpower, 9
  aba_model, 11
  aba_plot, 13
  aba_plot_coef, 14
  aba_plot_metric, 15
  aba_plot_roc, 17
  aba_read, 18
  aba_robust, 19
  aba_screen, 20
  aba_selection, 22
  aba_summary, 24
  aba_write, 25
  adnmerge, 27
  all_combos, 28
  as_reactable, 29
  as_table, 30

  everyone, 31

  fit.abaModel, 31

  set_covariates, 32
  set_data, 33
  set_groups, 34
  set_outcomes, 35
  set_predictors, 36
  set_stats, 37
  stat_ancova, 39
  stat_glm, 40
  stat_lm, 41
  stat_lme, 43
  stat_mmrm, 44

  stat_retest, 46
  stat_roc, 47