Package 'proteomicsHF'

February 16, 2021

Title ARIC Proteomics of Heart Failure -- Biomarker Discovery Pipeline

Version 1.1.0

Date 2021-02-19

Depends R (>= 3.6), stats, graphics, grDevices, utils, magrittr, rlang

Imports methods, tidyverse (>= 1.3.0), dplyr (>= 1.0.2), purrr (>= 0.3.4), readr (>= 1.4.0), tibble (>= 3.0.4), tidyr (>= 1.1.2), haven (>= 2.3.1), labelled (>= 2.7.0), mice (>= 3.12.0), survival (>= 3.2), glue (>= 1.4.0), tictoc (>= 1.0), furrr (>= 0.2.1), EnhancedVolcano (>= 1.4.0), ggplot2 (>= 3.2.1), glmnet (>= 4.0), randomForestSRC (>= 2.9.3), ggRandomForests (>= 2.0.1), WGCNA (>= 1.69), gridExtra (>= 2.3), fastcluster (>= 1.1), dynamicTreeCut (>= 1.63)

Description Functions necessary to analyze and replicate results of biomarker discovery pipeline for SOMALogic 5K Assay proteins associated with incident heart failure. Results are cached and derived from the Atherosclerosis Risk in Communities (ARIC) cohort from visits three and five. Includes functions to set adjustment variables, impute missing variables, summarize data, conduct uni-protein and multi-protein analyses and visualize data. Also includes a number of utility functions for data manipulation and reporting.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Author Pranav Dorbala [aut, cre]

Maintainer Pranav Dorbala <pdorbala@bwh.harvard.edu>

Suggests knitr, rmarkdown

VignetteBuilder knitr

R topics documented:

`add.data` *Augment Data Variables*

Description

This function serves to add variables to the master tibble dataframe. Variables are joined in on the id column.

Usage

```
add.data(new.path, old.data, scaled = F, mods, filts = NULL)
```

Arguments

`new.path`	Path to Stata .dta file containing new variables
`old.data`	Tibble of old study data to merge with
`scaled`	Boolean indicating whether new data should be scaled
`mods`	List of paths to formatted csv files creating new adjustment variables. Format of csv is detailed below. This parameter is required. See `get.adjust()` for formatting.
`filts`	List of path (singular) to formatted csv of the data filters to be applied to the dataset. See `get.adjust()` for formatting.

Details

See `get.adjust()` for csv formatting details.

Value

Tibble with new variables joined in.

Examples

```
## Not run:
fifth.visit.adtl <- add.data(new.path = 'data/ARIC_proteomics_addlvar_01062021.dta',
                             old.data = haven::read_dta('data/ARICmaster_121820.dta'),
                             mods = list('data/visit-five/adtlvars.csv'))

## End(Not run)
```

`cox.arry` *Cox Array Main Function*

Description

This is the main access point for using parallel processing to complete numerous Cox regressions simultaneously.

Usage

```
cox.arry(proteins, data, time, outc, adjust, .src)
```

Arguments

`proteins`	This is the vector of all proteins to be considered for Uniprotein, Adjusted Cox Regressions.
`data`	This is the tibble dataframe with the covariates for regression
`time`	This is a character vector of the Survival time to event Outcome.
`outc`	This is a character vector of the Survival event indicator Outcome.
`adjust`	This is a vector of all the adjustment variables.
`.src`	This is an indicator of whether the models are the derivation or validation models. It is 1 when the data is the derivation dataset, and 2 when it is the validation dataset.

Details

This function utilizes the `furrr::future_map()` function to complete many regressions quickly. Each protein is regressed individually against the time and outcome variable specified, while adjusted for the provided variables.

The .src input is specified to distinguish data between derivation data and validation data. Derivation data should be `.src = 1` while validation data should be `.src = 2`.

Value

a Tibble containing the results of the multiple regression models.

Examples

```
## Not run:
  data <- haven::read_dta('~/proteomics/studydata.dta') %>% ...
      (merge study data with SomaData, see [get.visit()] for specifics)

  univariate.results <- cox.arry(labels$term, data,
                                 time = "fuptime",
                                 outc = "hfdiag",
                                 adjust = adjust,
                                 .src = 1)

 validation.results <- cox.arry(labels$term, data.validation,
                                 time = "fuptime",
                                 outc = "hfdiag",
                                 adjust = adjust,
                                 .src = 2)


## End(Not run)
```

`cox.modl` *Cox Model Wrapper Function*

Description

This function serves to wrap the survival coxph function for easier usage in evaluating many cox models simultaneously. This function is accessed by `cox.arry()` in order to parallelize the univariate comparisons of every protein against the outcome.

Usage

```
cox.modl(
  .data,
  time = "fuptime",
  outcome = "hfdiag",
  protein,
  .adjust = adjust,
  extend = 100
)
```

Arguments

`.data`	The data tibble containing the variables to be regressed.
`time`	A character vector of the outcome variable time to event
`outcome`	A character vector of the outcome variable event indicator
`protein`	A character vector indicating which protein is being regressed.
`.adjust`	A vector of all adjustment variables for this model.
`extend`	Number of asterisks to add to Pvalue table indicator.

Details

This function returns the effect estimate for the "protein" of interest adjusted. The function also creates a string descriptor to print output as table for `.csv` files. The significance of the p value is indicated by each marker denoting a power of ten smaller.

Value

A list with three components:

hazr The Hazard Ratio of the Protein

cint A Vector of length 2 with Lower and Upper Confidence Interval

pval The P value of the model covariate

desc A string that combines the relevant information (such as hazard ratio and confidence interval) needed to display in a table.

Examples

```
## Not run:

  data <- haven::read_dta('~/proteomics/studydata.dta')

  cox <- cox.modl(data, time = 'adjudhfdate', outcome = 'adjudhf_bwh',
                  protein = 'SeqId_7655_11', .adjust = adjust)


## End(Not run)
```

`elastic.net` *Main Elastic Net Wrapper Function*

Description

This function is a wrapper function to the `glmnet::cv.glmnet()` function to conduct LASSO / ElasticNet regularized regression. The regression optimizes the C statistic, forces the adjustment variables, and runs the algorithm using parallel processing.

Usage

```
elastic.net(data, terms, adjust, time, outc, .alpha = 1)
```

Arguments

`data`	The tibble containing the merged Protein and Study Data.
`terms`	The terms to consider in the LASSO Regression.
`adjust`	The vector of Adjustment variables to force
`time`	A character vector of the Survival time to event outcome variable.
`outc`	A character vector of the Survival event indicator outcome variable.
`.alpha`	The optional alpha parameter to change LASSO to Elastic Net.

Value

model The returned cv.glmnet LASSO Elastic Net Model.

coefs The coefficients of the Non-zero covariates retained.

terms The Final retained Proteins from this model.

Examples

```
## Not run:

 fifth.lasso <- elastic.net(fifth.visit,
                            bonferroni$kept,
                            adjust, time, outc)


## End(Not run)
```

`get.adjust` *Filter Study Data Set to Variables of Interest*

Description

This function filters and creates new variables from the study data. This function takes csv file inputs that detail the new variable names and formulas created from the variable names in the Stata file.

Usage

```
get.adjust(stata, mods, filts = NULL)
```

Arguments

`stata`	Study (ARIC or HUNT) dataset in stata format
`mods`	List of paths to formatted csv files creating new adjustment variables. Format of csv is detailed below. Files should be titled `adjusted.csv` and the other `outcomes.csv`. This parameter is required
`filts`	List of path (singular) to formatted csv of the data filters to be applied to the dataset

Details

This function uses csv files to inject commands into `dplyr::mutate()` and `dplyr::filter()` functions. The format of the csv file to properly execute the desired commands is detailed below. The `mods` parameter is required and must be a list of path(s) to the csv files. Convention is to include one file with the outcome variables–including heart failure diagnosis variables and time to event variables and another file including adjustment variables, such as demographics and clinical history. The filter csv should include filtering conditions, such as excluding patients with prevalent heart failure, and the format of that csv is also detailed below.

Value

Tibble with data from all variables found in the mods csv files.

CSV Formatting

Adjustment and Outcome Variables:

Files must include a header for three columns–Variable, Expression, Label. Then, a new row must be created for each new variable, its expression, and its label. Expressions are R code snippets that calculate variables. Data in tibble should be assumed to be attached–same as when tidyverse formats its variables.

!! First row in adjustment variables must always be the study identifier for merging purposes. !!

Examples::

Adjustment Variables::

Variable	Expression	Label
id	id	ARIC COHORT STUDY ID
age	v5age51	Visit 5 Age
bmi	v5_bmi51	Visit 5 BMI
race	as.factor(race == 1)	Subject Race (Black == 1)

This table is coded in csv as:

```
Variable, Expression, Label
id, id, ARIC COHORT STUDY ID
age, v5age51, Visit 5 Age
bmi, v5_bmi51, Visit 5 BMI,
race, as.factor(race == 1), Subject Race (Black == 1)
```

Outcomes Variables::

Variable	Expression	Label
hfdiag	!is.na(adjudhf_bwh)	Incident Heart Failure Dx
fuptime	as.double(adjudhfdate - v5date51)	V5 HF Follow Up Time

This table is coded in csv as:

```
Variable, Expression, Label
hfdiag, !is.na(adjudhf_bwh), Incident Heart Failure Dx
fuptime, as.double(adjudhfdate - v5date51), V5 HF Follow Up Time
```

!! The First Row Must be the primary outcome, and the last row must be the primary time to event variable. This quirk may be patched in the coming versions.

Filters CSV Format::

These files filter data according to exclusion criteria. Most used to remove subjects with prevalent heart failure.

Data Filters and Exclusions::

Files must include headers for three columns–Variable, Operator, Expression. These headers represent the new variable, its filtering operation (==, >, >=, <, <=, etc.), and the expression for this operation respectively.

Examples::

Variable	Operator	Expression
v5_prevhf52	==	FALSE
fuptime	>	0

This table is coded in csv as:

Variable, Operator, Expression
v5_prevhf52, ==, FALSE
fuptime, >, 0

Examples

```
## Not run:
fifth.visit.study <-
  get.adjust(stata = haven::read_dta('data/ARICmaster_121820.dta'),
             mods  = list('data/visit-five/outcomes.csv',
                          'data/visit-five/adjusted.csv'),
             filts = list('data/visit-five/filters.csv'))

fifth.visit.echo <-
  get.adjust(stata = haven::read_dta('data/ARICmaster_121820.dta'),
             mods  = list('data/visit-five/echovars.csv'))

## End(Not run)
```

`get.data` *Data Joining, Imputation, and Scaling*

Description

Data Joining, Imputation, and Scaling

Usage

```
get.data(master, soma.data, adjust)
```

Arguments

`master`	Master study data tibble from `haven::read_dta()`
`soma.data`	Parsed SomaLogic proteomic data tibble from `get.soma()`
`adjust`	List of Adjustment variables. Must match `adjusted.csv` files, but should not include id variable.

Value

Tibble of joined and cleaned proteomics and study data.

Examples

```
## Not run:
labels <- get.labels('data/proteinMapping.csv', sid,
                     uniprot.full.name,
                     entrezgenesymbol,
                     flag2 == 0)

soma.data.fifth <- get.soma('data/somaV5.txt', labels = labels)

fifth.visit.study <-
  get.adjust(stata = haven::read_dta('data/ARICmaster_121820.dta'),
             mods  = list('data/visit-five/outcomes.csv',
                          'data/visit-five/adjusted.csv'),
             filts = list('data/visit-five/filters.csv'))

adj <- readr::read_csv('data/visit-five/adjusted.csv')[[1]][-1]

visit.data <- get.data(fifth.visit.study, soma.data.fifth, adj)

## End(Not run)
```

`get.filtered` *Filter Data given series of filter commands*

Description

This function is a helper function to take a series of filter instructions (from the `filters.csv` file) and uses them to filter a tibble of study data.

Usage

```
get.filtered(data, filts)
```

Arguments

`data`	The Tibble of Data from Study to be Filtered
`filts`	The list of file path(s) to `filters.csv`. Most likely a list of length 1.

Details

This command writes an intermediate csv file to the working directory, named `filres.csv`. This intermediate file stores the data about how many subjects were filtered by each condition. This data is later used to create consort diagrams.

Value

returns a filtered tibble of study data. Does not do any error checking.

Examples

```
## Not run:

filts <- '~/proteomics/filters.csv'
data  <- haven::read_dta('~/proteomics/studydata.dta')

filtered.data <- get.filtered(data, filts)

OR

data <- haven::read_dta('~/proteomics/studydata.dta') %>%
    get.filtered('~/proteomics/filters.csv')


## End(Not run)
```

`get.labels` *Get Protein Names and Labels from SeqId Identifiers*

Description

This is a function to generate a dictionary to translate from SeqId to Protein Name and Entrez Gene Symbol. Output from this function is used in creating tables, figures, and labels for SeqIds. This function also requires an input for data cleaning flags. Proteins that are flagged will not be included in the final label frame.

Usage

```
get.labels(path, term, name, gene, flag)
```

Arguments

`path`	File Path to SomaLogic Data Dictionary
`term`	Column Variable ID for the SeqId term
`name`	Column Variable ID for the UniProt Full Name Column
`gene`	Column Variable ID for the Entrez Gene Symbol Column
`flag`	Filtering Function for Proteins that Pass QC

Details

The term, name, gene, and flag functions must be provided as-is (not as strings). This is a quirk of using tidyverse and dplyr.

Value

Tibble with columns `term` for SeqId, `name` for UniProt Name, and `gene` for Entrez Gene Symbol. Data Frame rownames are set as the SeqId identifiers.

Examples

```
## Not run:
labels <- get.labels('data/proteinMapping.csv', sid,
                     uniprot.full.name,
                     entrezgenesymbol,
                     flag2 == 0)

## End(Not run)
```

`get.scaled` *Scale Data*

Description

This function scales data after tables are made. Scales numeric adjustment and proteomic variables. Done separately to maintain scale and values for table 1 and figure generation. Numeric adjustment variables must be specified. See `numeric.id()` for more details.

Usage

```
get.scaled(data, adjust, proteins = NULL)
```

Arguments

`data`	Tibble of Data to be scaled
`adjust`	List of NUMERIC ADJUSTMENT VARS ONLY – output of `numeric.id()`.
`proteins`	List of Proteins to Scale

Value

Tibble where all numeric adjustment variables and protein variables are scaled to mean 0 and sd 1.

Examples

```
## Not run:
labels <- get.labels('data/proteinMapping.csv', sid,
                     uniprot.full.name,
                     entrezgenesymbol,
                     flag2 == 0)

soma.data.fifth <- get.soma('data/somaV5.txt', labels)

fifth.visit.study <-
  get.adjust(stata = haven::read_dta('data/ARICmaster_121820.dta'),
             mods  = list('data/visit-five/outcomes.csv',
                          'data/visit-five/adjusted.csv'),
             filts = list('data/visit-five/filters.csv'))

adj <- readr::read_csv('data/visit-five/adjusted.csv')[[1]][-1]

visit.data <- get.data(fifth.visit.study, soma.data.fifth, adj)
```

```
num.vars <- numeric.id(visit.data, adj)
visit.data.scaled <- get.scaled(visit.data, num.vars, labels$term)

## End(Not run)
```

`get.score` *LASSO Score Constructor*

Description

This function uses the LASSO Model retained coefficients to recreate the linear predictor. This predictor is then adjusted and regressed against the outcome. This predictor is only based on the protein values, thus adjustment variables must be included.

Usage

```
get.score(data, coefs, adjust, time, outc)
```

Arguments

`data`	The tibble containing the merged proteomic and study data.
`coefs`	The LASSO coefficients output from `elastic.net()`.
`adjust`	The vector of adjustment variables
`time`	The character vector of the Survival outcome time to event variable
`outc`	The character vector of the Survival outcome event indicator

Value

cox A Cox Model of the LASSO Score

score The actual double vector constructed score

Examples

```
## Not run:

fifth.score <- get.score(fifth.visit, lasso$coefs, adjust,
                         "fuptime", "hfdiag")


## End(Not run)
```

get.soma *Load Raw SomaLogic Proteomic Data From Text File*

Description

This is the function to read and process raw text file proteomics data from SomaLogic. Data can provided with a label to preserve mapping from SeqId to UniProt Full Name. Study ID is preserved in the ID data column.

Usage

```
get.soma(path, labels = NULL, id = "SampleId")
```

Arguments

path	File path to the SomaLogic text file with protein data.
labels	Output of get.labels() function protein mapping dictionary
id	Variable name of Study Identifier in SomaLogic Text File used to map study subjects to clinical data.

Value

Tibble containing raw proteomic data for all measured subjects. Tibble will be labelled if labelling dictionary is provided. Sample ID to map to subject study id is stored in the id variable.

Examples

```
## Not run:
labels <- get.labels('data/proteinMapping.csv', sid,
                     uniprot.full.name,
                     entrezgenesymbol,
                     flag2 == 0)

soma.data.fifth <- get.soma('data/somaV5.txt', labels = labels)
soma.data.third <- get.soma('data/somaV3.txt', labels = labels)

## End(Not run)
```

get.visit *Get Master Visit Data File with Proteomics*

Description

This is the outward facing wrapper function to process and load data for analysis. This function will read SomaLogic and Study Data, Impute missing adjusting covariates, filter data for exclusions

Usage

```
get.visit(soma.data, path.mods, master)
```

Arguments

soma.data	SomaLogic Proteomic Raw Data from get.soma()
path.mods	Char vector path to folder structure containing new adjustment variables, filtering conditions, and outcome variables. This folder must contain csv files formatted as detailed in get.adjust(). This function expects that adjustment variables will be in adjusted.csv, outcome variables in outcomes.csv and filters in filters.csv.
master	Tibble of master data file with study data. Output of haven::read_dta() function.

Value

Tibble of study data merged with Proteomics Data, labelled, and filtered.

Examples

```
## Not run:
labels <- get.labels('data/proteinMapping.csv', sid,
                     uniprot.full.name,
                     entrezgenesymbol,
                     flag2 == 0)
aric.master <- haven::read_dta('data/ARICmaster_121820.dta')
soma.data <- get.soma('data/somaV5.txt', labels)

fifth.visit <- get.visit(soma.data = soma.data,
                         path.mods = 'data/visit-five',
                         labels = labels,
                         master = aric.master))

## End(Not run)
```

---

numeric.id *Get Numeric Variables (for scaling)*

Description

Numeric Values will likely need to be scaled for survival analysis. This function identifies which variables are numeric, separating them from categorical factor variables. used in get.visit() and add.data()

Usage

```
numeric.id(data, vars)
```

Arguments

data	Tibble of data containing variables to be considered
vars	Vector of all variables to be considered

Value

Vector containing column names of numeric variables

Examples

```
## Not run:
new.vars <- readr::read_csv('data/visit-five/echovars.csv')[[1]]
the.data <- haven::read_dta('data/ARICmaster_121820.dta')
num.vars <- numeric.id(vars = new.vars, data = the.data)

## End(Not run)
```

`rf` *Random Survival Forest Wrapper Function*

Description

This function wraps the `randomForestSRC::rfsrc()` function. It first runs an initial tuning algorithm, identifying the most appropriate parameters for random forest model fit. Then, the model runs the more detailed random forest tree generation step. The tree is then returned. The time variable must be stored in a variable named time, and outcome in variable named outc. These variable translations are done automatically in the functions accessed by users.

Usage

```
rf(data, seed)
```

Arguments

`data`	Tibble of data containing the variables to be regressed.
`seed`	The random seed to standardize random forest creation

Value

rand.tree The random forest grown in this algorithm

tune.tree The tuning forest and parameters derived

vars.tree The variables used to grow this forest

Examples

```
## Not run:

tree <- rf(visit.data, seed=101010)

TRUE USAGE:

tree.list <- filter.rf(visit.data, time="fuptime", outc="hfdiag",
                       iter = 30, adjust = adjustment.variables)

## End(Not run)
```

terms.to.keep *Filtering Function to Identify which Proteins are retained*

Description

This function calculates which proteins are retained at specified significance intervals. For example, when the univariate results are provided, this function returns all proteins that are bonferroni significant.

Usage

```
## S3 method for class 'to.keep'
terms(data, .pval, .src, .terms = NULL)
```

Arguments

data	The Univariate (and Validation) Tibble with Regression Results. Output of cox.arry().
.pval	The significance level to examine which proteins are retained.
.src	The tibble in which we examine retained proteins. .src = 1 means the Derivation dataset if the derivation and validation sets are row-bound.
.terms	Vector of terms to limit examination of significance to. Not necessary to be specified.

Details

This function can also be used to check which proteins are retained in validation.

Value

kept A vector containing all Proteins that were retained by the filter

data A Tibble with the Hazard Ratio and Significance of Retained Proteins

Examples

```
## Not run:
plot.data <-
  dplyr::bind_rows(univariate.results,
                   validation.results) %>%
  dplyr::mutate(Name = labels[term, "name"])

bonferroni <-
  terms.to.keep(plot.data,
                .pval = 0.05/length(proteins), 1)
falsediscr <-
  terms.to.keep(plot.data,
                .pval = 0.05, 1)

retained <-
  terms.to.keep(plot.data,
                .pval = 0.05, 2,
                .terms = bonferroni$kept)
```

```
## End(Not run)
```

`valid.rf` *Validation Tree Creation*

Description

This function generates a random forest from a set of derived important variables. For example, if a set of important random forest variables were calculated from the ARIC Visit 5 Data, this function then uses the HUNT data to create a tree from those variables. This tree is then returned, evaluating how these variables perform in informativeness in the new dataset.

Usage

```
valid.rf(data, time, outc, valid.terms)
```

Arguments

`data`	The tibble containing the Validation dataset
`time`	The time to event variable for this analysis
`outc`	The event indicator variable for this analysis
`valid.terms`	The pre-calculated important terms needed to generate the tree.

Value

tree This is the Random Forest grown using the supplied terms

vsel This is the calcuation of relative variable importance, showing how the provided terms replicate in the validation dataset.

Examples

```
## Not run:

hunt.data <- haven::read_dta('hunt_data.dta')
time.var <- 'fuptime'
outc.var <- 'hfdiag'

valid.terms <- readRDS('ARIC_random_forest_derived_terms.RDS')

validation.tree <- valid.rf(data = hunt.data, time = time.var,
                            outc = outc.var, valid.terms = valid.terms)


## End(Not run)
```

volcanoPlot *Volcano Plot Generator To Summarize Univariate Protein Significance*

Description

This function takes in plot data and uses it to construct a Volcano Plot to visualize significance. The legend is dependent on the derivation and validation sets. As currently implemented, this function must have both datasets bound.

Usage

```
volcanoPlot(data, .pval1, .pval2, suffix, ...)
```

Arguments

data	The plot data (with vars hazr and pval) to create volcano plot with.
.pval1	The significance indicator for scaling 0.05 for the derivation dataset. The number of comparisons must be provided (i.e. 4877 for bonferroni or 1 for FDR).
.pval2	The significance indicator for scaling 0.05 for the validation dataset. The number of comparisons must be provided (i.e. 4877 for bonferroni or 1 for FDR).
suffix	The Suffix that is used on the data join to separate the Derivation and Validation Sets.
...	Additional Parameters (such as title) to be passed to EnhancedVolcano::EnhancedVolcano()

Value

A Volcano Plot, comparing Hazard Ratio against -log10(pvalue)

Examples

```
## Not run:

plot.data.visit.five <-
  dplyr::left_join(univariate.results,
                   validation.results,
                   by = "term",
                   suffix = c(".V5", ".V3"))

  volcanoPlot(plot.data.visit.five,
              length(proteins),
              length(bonferroni$kept),
              suffix = c(".V5", ".V3"))


## End(Not run)
```

Index