Package ‘PEACOK’

November 10, 2020

Type Package

Title Phenome Exome Association and Correlation Of Key phenotypes

Version 1.0.7

Date 2020-02-14

Author Quanli Wang, Slave Petrovski

Maintainer Quanli Wang <quanliwang20@hotmail.com>

Description
This tool set re-implements, customizes and enhances the phenome scan functionlities of the original PHESANT R programs as described in https://github.com/MRCIEU/PHESANT. In this implementation, while trying to make it compatiable with the original approach, we focus on seperating phenotype matrix generation from statistical association tests and allowing statistical tests to be performed seperately on different computing envrionments.

License GPL (>= 3)

Depends MASS, optparse, tidyr, dplyr, methods

Imports nnet, data.table, lmtest,R.utils, Matrix

R topics documented:

`PEACOK-package` *Phenome Exome Association and Correlation Of Key phenotypes*

Description

This tool set re-implements, customizes and enhances the phenome scan functionlities of the original PHESANT R programs as described in https://github.com/MRCIEU/PHESANT. In this implementation, while trying to make it compatiable with the original approach, we focus on seperating phenotype matrix generation from statistical association tests and allowing statistical tests to be performed seperately on different computing envrionments.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.
An overview of how to use the package, including the most important functions

Author(s)

Quanli Wang, Slave Petrovski

Maintainer: Quanli Wang <quanliwang20@hotmail.com>

References

Literature or other references for background information

See Also

To be done

Examples

```
#example here
```

---

annotate.tree.code *Add annotation columns to data code that has tree structures*

Description

Add annotation columns to data code that has tree structures. For a given datacode file with tree structures, it annotate it by add more columns indicating tree level, root node meaning as well as replacing comma in orignal meaning column with pipe (|) for better csv ewxperience.

Usage

```
annotate.tree.code(data_code_file)
```

Arguments

`data_code_file`	an original datacoding file downloaded from UKB website which has a tree structure in node.

Value

Returns a dataframe with extra annotation columns beside the original columns.

Examples

```
  #datacode <- annotate.tree.code(data_code_file)
```

---

auto.update.variable.info *Update and consolidate a previous version of variable info file with current version of UKB Data Dictionary.*

Description

Taking a previous version of the variable info file, and a copy of the UKB Data Dictionary, this function will consolidate and update the previous version based on the new Data Dictionary and output a new version that generally needs to be curated. It also outputs list of fields that are added or removed from the data dictionary.

Usage

```
auto.update.variable.info(old_variable_info_file, UKB_data_dictionary_file)
```

Arguments

`old_variable_info_file`
: A previous versionn of variable info file, which was curated to have PHESANT specific columns for all varibles.

`UKB_data_dictionary_file`
: A newer version of UKB Data Dictionary, which can be ontained by calling download.data.dictionary function.

Value

Return a list of two data frames: updated_fields for all added/removed fields and variable_info for consolidated variable info.

Examples

```
#library(PEACOK)
#args <- commandArgs(T)

#variable_info_base <- "../variable-info"
#output_path <- file.path(variable_info_base, 'update-outcome-info')
#download.data.dictionary(output_path)

#old_variable_info <- file.path(variable_info_base,'outcome-info.tsv')
#UKB_data_dictionary <- file.path(output_path,"Data_Dictionary_Showcase.csv")
#updated_variable_info <- file.path(output_path,'new-field-list.tsv')
#new_variable_info <- file.path(output_path,'outcome-info-new.tsv')

#updated_info <- auto.update.variable.info(old_variable_info, UKB_data_dictionary)
#if (is.null(updated_info$updated_fields)) {
#    cat("\nInfo: New new fields were added or deleted")
#} else {
#    cat(paste0("Info: Updated fields are writen to file: ", updated_variable_info))
#    fwrite(updated_info$updated_fields, sep='\t', file=updated_variable_info)
#}
# Write out, and make sure it's tab separated.
#fwrite(updated_info$variable_info, sep='\t', file = new_variable_info)
#cat(paste0("\nInfo: Updated variable info is writen to file: ", new_variable_info))
```

`binary.logistic.regression` *Perform binary logistic regression*

Description

Performs binary logistic regression and stores result in 'results-logistic-binary' results file.

Usage

```
binary.logistic.regression(opt, varName, varType, thisdata, isExposure, phenoStartIdx)
```

Arguments

`opt`	The list of input options provided by user.
`varName`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`varType`	The value type of the variable. Should always be "Categorical single" here and used for logging purpose only.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`isExposure`	If the variable is labeled as trait of interest.
`phenoStartIdx`	The column index of the first phenotype.

Details

The variable will be skipped if less than 500 samples have value or there are more than two levels or at least one level has less than 10 observations.

Value

Return nothing and all output and lof information are written to output directory specified by user.

`choose.reference.category` *Choose Reference Category*

Description

Convert an integer vector with possible NA's to a factor and use the category/factor with most number of examples as reference.

Usage

```
choose.reference.category(pheno)
```

Arguments

`pheno`	An integer vector with possible NA's.

Value

A factor with the most number of examples as reference.

`download.data.dictionary` *Download UKB Data Dictionary*

Description

Download UKB Data Dictionary to user specified folder.

Usage

```
download.data.dictionary(output_path)
```

Arguments

`output_path`	The output folder where the UKB Data Dictionary file will be downloaded.

Value

If successful, the current UKB Data Dictionary will be downloaded to the folder sepcificed by user.

Examples

```
# Download the UKB Data Dictionary to Current working directory
# download.data.dictionary(getwd())
```

`downsample.by.gender` *Downsample Phenotypes by Gender*

Description

Downsample controls or cases based on gender to reduce the effect of gender imbalance.

Usage

```
downsample.by.gender(samples, phenotypes, alpha = 0.05, seed = 1234)
```

Arguments

`samples`	A dataframe with rows for samples and two columns. The first column is "userID", identifying the samples. The second column is 'is.female", of boolean type, indicating if a sample is female or not.
`phenotypes`	A dataframe with rows for samples and columns for phenotypes. The first column is "userID", identifying the samples.
`alpha`	The significant level(p value) when testing if a phenotype is gender imbalanced. The default is 0.05.
`seed`	The random seed for reproducibility during downsampling. The default value is 1234.

Details

For each input phenotype, 1. If all cases are female, only females are retained in controls. 2. If all cases are male, only mailes are retained in controls. 3). If there are both females and males in cases, it checks gender imbalance using Fisher Exact Test at the signifiant level alpha defined by user(default value is 0.05). 4). If gender imbalance is detected, it downsamples either males or females in controls so the ratio of controls over cases are matched as closely as possible for gender.

Value

Return a dataframe that has the same structure as input phenotypes dataframe but with excluded samples marked as NA.

`get.is.exposure` *Check field exposure*

Description

Check whether a field denotes the trait of interest, as specified.

Usage

```
get.is.exposure(vl, varName)
```

Arguments

`vl`	variable list object
`varName`	the field ID to be checked

Value

Return true if the field has a "YES" value in TRAIT_OF_INTEST column.

`init.data` *Validates, loads input data and inilizes working environment to run PEACOK*

Description

Validates, loads input data and inilizes working environment to run PEACOK.

Usage

```
init.data(opt)
```

Arguments

`opt`	The list of input options provided by user.

Value

Return a list objerct containing all the data used to run PEACOK.

1. data: data frame for all phenotypes
2. vl: variable list for all phenotypes
3. confounders: data frame for all confounders
4. phenoStartIdx: the column index for first phenotype column in data data frame
5. phenoVars: a list of all phenotypes, excluding user ID

Examples

```
# input <- init.data(opt)
```

`init.variable.lists` *Load the variable information and data code information files*

Description

Load the variable information and data code information files

Usage

```
init.variable.lists(variablelistfile, datacodingfile)
```

Arguments

`variablelistfile`	the input phenotype variable list file
`datacodingfile`	the input data coding file for the DATA_CODING field for phenotypre

Details

Load the variable information and data code information files

Value

return a list contain with two variables:

`phenoInfo`	A data frame that holds information about phenotype varibles from UKB
`dataCodeInfo`	A data frame that provides further information about the DATA_CODING field of a phenotype from phenoInfo

Examples

```
  #vl <- init.variable.lists(opt$variablelistfile, opt$datacodingfile)
```

irnt *Inverse Rank Normal Transformation*

Description

Perform an inverse rank normal transformation for input phenotypes

Usage

```
irnt(pheno,seed)
```

Arguments

`pheno`	A vector of numeric vlues with NA allowed.
`seed`	Random seed for tie breakers, default to 1234.

Value

A vector of the transformed values.

is.tree.code *Check if the input UKB coding file has a tree structure.*

Description

Check if the input UKB coding file has a tree structure. Return TRUE if the coding has a tree structure and FALSE if not.

Usage

```
is.tree.code(data_code_file)
```

Arguments

`data_code_file`	The file path/name of a UKB data coding file.

Value

Return TRUE if the coding has a tree structure and FALSE if not.

`load.confounders` *Loads confounder variables*

Description

Loads confounder variables from phenotype file or confounder file based on user options. If user provides a confunder file, all confounder variables will be extracted from it. Otherwise, confounder variables will be read from phenotype file and a set of predefined confounders will be extracted.

Usage

```
load.confounders(opt, phenotypes)
```

Arguments

`opt`	The list of input options provided by user.
`phenotypes`	The phenotype data from load.phenotypes function.

Details

If user provides a confunder file, all confounder variables will be extracted from it. Otherwise, confounder variables will be read from phenotype file and a set of predefined confounders will be extracted. When reading from phenotype file for confounders, the following variables will be extracted:

1. age: x21022_0_0
2. sex: x31_0_0
3. genetic: x22000_0_0, with genetic batch used to create genetic chip variable, optional, present only if opt$genetic is set.
4. genetic principal components and assessment centre, optional, present only if opt$genetic and opt$sensitivity are set: x22009_0_1 through x22009_0_10, and x54_0_0.
5. assessment centre only if opt$genetic is false and opt$sensitivity is true: x54_0_0.

Value

Returns a dataframe for all confounder variables, with first column being UserID.

Examples

```
#conf <- load.confounders(opt, phenotype)
```

`load.data` *Loads phenotype, trait of interest data files*

Description

Validates and loads phenotype, trait of interest, and relted fields from data files.

Usage

```
load.data(opt, vl)
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list

Details

Taking user options as input, this function validates and loads phenotype, trait of interest, and relted fields from data files.

1. loads phenotype and trait of interest data files
2. creates phenotype / trait of interest data frame
3. creates confounder data frame
4. parses and creates indicator data frame
5. returns an object holding these three data frames

Value

Returns a list of dataframes:

1. phenotype: data frame for all phenotypes
2. confounders: data frame for all confounders
3. inds: data frame for all related indicators

Examples

```
#d <- load.data(opt, vl)
```

`load.phenotypes` *Load phenotypes from phenotype file*

Description

Load all or part of phenotypes from phenotype file based on user option.

Usage

```
load.phenotypes(opt)
```

Arguments

`opt`	The list of input options provided by user.

Details

Load all or part of phenotypes from phenotype file based on user option.

Value

Return a data frame object that conbtains all the required phenotypes.

Examples

```
#phenotype = load.phenotypes(opt)
```

`load.trait.of.interest` *Load trait of interest*

Description

Load trait of interest, either from separate trait of interest file, or from phenotype file.

Usage

```
load.trait.of.interest(opt, phenotypes)
```

Arguments

`opt`	The list of input options provided by user.
`phenotypes`	The phenotype data from load.phenotypes function.

Details

Load trait of interest, either from separate trait of interest file, or from phenotype file.

Value

Returns a data frame of two columns for User ID and trait of interest.

Examples

```
#toi <- load.trait.of.interest(opt, phenotype)
```

`opt` *Default options for "test" dataset*

Description

Default options to be used when user runs the program with "–test" option in command line. The files used in "–test" mode will be updated from inst/extdata folder of the package.

Details

The fields are: List of 15

1. userId : chr "userId"
2. test : logi TRUE
3. sensitivity : logi FALSE
4. genetic : logi TRUE
5. save : logi FALSE
6. confidenceintervals: logi TRUE
7. standardise : logi TRUE
8. help : logi FALSE
9. resDir : chr ""
10. phenofile : chr "inst/extdata/phenotypes.csv"
11. variablelistfile : chr "inst/extdata/variable-lists/outcome-info.tsv"
12. datacodingfile : chr "inst/extdata/variable-lists/data-coding-ordinal-info.txt"
13. traitofinterestfile: chr "inst/extdata/exposure.csv"
14. traitofinterest : chr "exposure"
15. varTypeArg : chr "all"

Please note that the resDir field will be replaced by actual working directory.

Examples

```
data(opt)
```

`option_list` *User Input Option List*

Description

This data object defines the options that the program accepts.

`pkg.env` *package level envrionment*

Description

A package level envrionment used to hold package level variables.

`process.data.code` *Download and parse UKB data codings*

Description

Optionally Download the UKB data coding files and then parse UKB into a format that can be used by the package.

Usage

```
process.data.code(variable_info_base, outcome_info_file, download)
```

Arguments

`variable_info_base`	The folder while all coding will be stored.
`outcome_info_file`	The "outcone-info"" file as ddefined in PHESANT.
`download`	Indicator if all data coding files should be downloaded or not. If false, variable_info_base should have an copy of previously downloaded data coding files.

Details

This function allows user to download and parse the data codings for "Categorical single" and "Categorical multiple" fields based on the infomation provided in "outcome-info" file, which lists all the fields to be recognized by the package.

Value

A data frame that holds extra info about each data codings, which can be then merged with previously curated data coding file.

`process.options` *Validate and parse user options*

Description

Validate and parse user options

Usage

```
process.options(test = FALSE)
```

Arguments

`test`	Indicates if the test data should be used instead.

Examples

```
# opt <- parse.options()
```

`read.geno.matrix` *Read the whole or a trunk of the Collapsing Matrix by rows.*

Description

Read the whole or a trunk of Collapsing Matrix by rows.

Usage

```
read.geno.matrix(matrix_file, gene_start = NULL, gene_end = NULL)
```

Arguments

`matrix_file`	The collapsing matrix to be read.
`gene_start`	The start gene of the matrix if read by trunk.
`gene_end`	The end gene of the matrix if read by trunk.

Value

Returns a data frame that holds the whole or a trunk of the collapsing matrix.

`read.pheno.matrix` *Read the whole or a trunk of the Phenotype Matrix by columns.*

Description

Read the whole or a trunk of the Phenotype Matrix by columns.

Usage

```
read.pheno.matrix(matrix_file, pheno_start = NULL, pheno_end = NULL)
```

Arguments

`matrix_file`	The Phenotype matrix file to be read.
`pheno_start`	The start index of the phenotype columns to be read.
`pheno_end`	The end index of the phenotype columns to be read.

Details

Returns a data frame that holds the whole or a trunk of the phenotype matrix.

`run` *Run an PEACOK analysis with user specificed options*

Description

Run an PEACOK analysis with user specificed options

Usage

```
run(opt)
```

Arguments

`opt`	The list of input options provided by user.

Value

No return value. All output will be writen to the user specificed oiutput folder.

Examples

```
# run(opt)
```

`run.categorical.ordered` *Run association test for Categorical Ordered phenotypes*

Description

Run association test for "Categorical Ordered" phenotypes.

Usage

```
run.categorical.ordered(pheno, geno, confounders)
```

Arguments

`pheno`	the phenotype vector
`geno`	the genotype vector
`confounders`	the confounder matrix

Value

Return a list of test statistics and summaries

`run.categorical.unordered` *Fitting multinomial log-linear models with confounders.*

Description

Given input phenotype and genotype vectors as well as coresponding covariates this function fits a multinomial log-linear model, assuming that the input phenotypes have more than two levels with unordered data.

Usage

```
run.categorical.unordered(pheno, geno, confounders)
```

Arguments

`pheno`	a vector with more than two distinct values as unordered categorical values.
`geno`	a numeric vector, typically binary vector in this package.
`confounders`	A dataframe with rows for samples and columns for confounder covariates.

`run.continuous` *Run association test for continuous phenotypes*

Description

Run association test for continuous phenotypes.

Usage

```
run.continuous(pheno,geno,confounders)
```

Arguments

`pheno`	the phenotype vector
`geno`	the genotype vector
`confounders`	the confounder matrix

Value

Return a list of test statistics and summaries

`run.geno.pheno.logistic` *Run association tests for a set of input genotypes and binary phenotypes using logistic regression.*

Description

Run logistic association tests for a set of input genotypes and binary phenotypes with matching confounders. If an annotation information is also provided, it will be used for added annotation. User can also provide the range/block of genotypes and phenotypes so association tests will be performed for choosen gentype/phenotype blocks.

Usage

```
run.geno.pheno.logistic(genotypes, phenotypes, file.confounder, file.annotaiton = NULL, geno_start
```

Arguments

`genotypes`	The input genotype file.(add format info)
`phenotypes`	The input phenotype file.
`file.confounder`	The companion confounder file for the input phenotype file.(add format info)
`file.annotaiton`	Optional. If provided, the info in annotaion file will be used to annotate the output.
`geno_start`	Optional. Default to the 1.See argument geno_end

`geno_end`	Optional. Default to the maxinum number of genotypes. (geno_start, geno_end), if provided, together will define the range of genotypes that will be used to run the association tests.
`pheno_start`	Optional. Default to the 1.See argument pheno_end
`pheno_end`	Optional. Default to the maxinum number of phenotypes. (pheno_start, pheno_end), if provided, together will define the range of phenotypes that will be used to run the association tests.
`ignoreConfounder`	Optional, default to FALSE. If setting to TRUE, the confounders will not be used in analysis even if it was provided.
`verbose`	Optional Default to 0. It currently takes values 0, 1 or 2 and the default value is 0. If the value is 0, no progress information will be printed. If the value is 1, it will print progress at the genotype levele. If the value is set to 2, i will print progress at the (genotype, phenotype) level.

`run.geno.pheno.logistic2`

Run association tests for a set of input genotypes and binary phenotypes using logistic regression with fewer output.

Description

Run logistic association tests for a set of input genotypes and binary phenotypes with matching confounders. If an annotation information is also provided, it will be used for added annotation. User can also provide the range/block of genotypes and phenotypes so association tests will be performed for choosen gentype/phenotype blocks.

Usage

```
run.geno.pheno.logistic2(genotypes, phenotypes, file.confounder, file.annotaiton = NULL, geno_star
```

Arguments

`genotypes`	The input genotype file.(add format info)
`phenotypes`	The input phenotype file.
`file.confounder`	The companion confounder file for the input phenotype file.(add format info)
`file.annotaiton`	Optional. If provided, the info in annotaion file will be used to annotate the output.
`geno_start`	Optional. Default to the 1.See argument geno_end
`geno_end`	Optional. Default to the maxinum number of genotypes. (geno_start, geno_end), if provided, together will define the range of genotypes that will be used to run the association tests.
`pheno_start`	Optional. Default to the 1.See argument pheno_end
`pheno_end`	Optional. Default to the maxinum number of phenotypes. (pheno_start, pheno_end), if provided, together will define the range of phenotypes that will be used to run the association tests.

`ignoreConfounder`	Optional, default to FALSE. If setting to TRUE, the confounders will not be used in analysis even if it was provided.
`verbose`	Optional Default to 0. It currently takes values 0, 1 or 2 and the default value is 0. If the value is 0, no progress information will be printed. If the value is 1, it will print progress at the genotype levele. If the value is set to 2, i will print progress at the (genotype, phenotype) level.

`run.geno.pheno.nonbinary` *Run association tests for a set of input genotypes and non binary phenotypes.*

Description

Run association tests for a set of input genotypes and non binary phenotypes with matching confounders. If an annotation file is also provided, it will be used for added annotation. User can also provide the range/block of genotypes and phenotypes so association tests will be performed for choosen gentype/phenotype blocks.

Usage

```
run.geno.pheno.nonbinary(genotypes, phenotypes, file.confounder,
                         file.annotaiton = NULL, ordered = FALSE, unordered =FALSE,
                           geno_start = NULL, geno_end = NULL,
                           pheno_start = NULL, pheno_end = NULL,
                        ignoreConfounder = FALSE, pheno_pheno = FALSE, verbose = 0)
```

Arguments

`genotypes`	The input genotype file.(add format info)
`phenotypes`	The input phenotype file.
`file.confounder`	The companion confounder file for the input phenotype file.(add format info)
`file.annotaiton`	Optional. If provided, the info in annotaion file will be used to annotate the output.
`ordered`	Indicator if the input phenotype is Categorical Ordered.
`unordered`	Indicator if the input phenotype is "Categorical Unordered.
`geno_start`	Optional. Default to the 1.See argument geno_end
`geno_end`	Optional. Default to the maxinum number of genotypes. (geno_start, geno_end), if provided, together will define the range of genotypes that will be used to run the association tests.
`pheno_start`	Optional. Default to the 1.See argument pheno_end
`pheno_end`	Optional. Default to the maxinum number of phenotypes. (pheno_start, pheno_end), if provided, together will define the range of phenotypes that will be used to run the association tests.

`ignoreConfounder`	Optional, default to FALSE. If setting to TRUE, the confounders will not be used in analysis even if it was provided.
`pheno_pheno`	Optional Default to FALSE. If setting to TRUE, the input genotype is actually of phenotype format. Might want to change this later to create a new function. Just a hack for now to run phenotype-phenotype association tests.
`verbose`	Optional Default to 0. It currently takes values 0, 1 or 2 and the default value is 0. If the value is 0, no progress information will be printed. If the value is 1, it will print progress at the genotype levele. If the value is set to 2, i will print progress at the (genotype, phenotype) level.

Details

Give more details about how this function works and the reason to allow geno/pheno blocks to be defined.

Value

Return association test results for all or given genotype/phenotype blocks. More details about the output are needed here.

Author(s)

Quanli Wang

`run.logistic` *Fitting logistic regression with confounders.*

Description

Given input phenotype and genotype vectors as well as coresponding covariates this function fits a logistic regression model phenotype ~ genotype + confounders and returns beta, standard error and p value associated with the genotype. To be consitent in output format with other similar testing functions in this package, it also computes some sumary statistics when applicable or NA otherwise.

Usage

```
run.logistic(pheno, geno, confounders)
```

Arguments

`pheno`	a binary vector with 0 for cases, 1 for controls and NA for misisng values.
`geno`	a numeric vector, typically binary vector in this package.
`confounders`	A dataframe with rows for samples and columns for confounder covariates.

Details

This function simply fits a logistic regresion phenotype ~ genotype + confounders and return test statistics associated with the genotype vector. It assumes that the orders of samples from all input are synced.

Value

Return a list of values.

`nSamples`	number of samples that have non misisng phenotype and genotype values
`numCases`	number of samples that have non misisng phenotype and genotype values and have non zero genotypes. Note that in the case of input genotype is nonbinary, the definition here might not be in line with typical difinition for cases.
`numControls`	number of samples that have non misisng phenotype and genotype values and zhave ero genotypes.
`p`	The p value for testing the coefficient for genotype is not zero.
`beta`	the beta coefficient for genotype.
`se`	the standarr error for the beta coefficient for genotype.
`lower`	Not used, always NA. We keep this in the output to be consistent with other similar testing functions.
`upper`	Not used, always NA. We keep this in the output to be consistent with other similar testing functions.
`MedCases`	The median phenotye values for the case samples defined above.
`MedControls`	The median phenotye values for the control samples defined above.

`run.logistic2` *Running logistic regression with confounders too.*

Description

Giving input phenotype and genotype vectors as well as coresponding covariates this function fits a logistic regression model phenotype ~ genotype + confounders and returns beta, standard error and p value associated with the genotype. This implementaiton is almost identical to run.logistic but with less output.

Usage

```
run.logistic2(pheno, geno, confounders)
```

Arguments

`pheno`	a binary vector with 0 for cases, 1 for controls and NA for misisng values.
`geno`	a numeric vector, typically binary vector in this package.
`confounders`	A dataframe with rows for samples and columns for confounder covariates.

Details

This function simply fits a logistic regresion phenotype ~ genotype + confounders and return test statistics associated with the genotype vector. It assumes that the orders of samples from all input are synced.

Value

`nSamples`	number of samples that have non misisng phenotype and genotype values
`numCases`	number of samples that have non misisng phenotype and genotype values and have non zero genotypes. Note that in the case of input genotype is nonbinary, the definition here might not be in line with typical difinition for cases.
`numControls`	number of samples that have non misisng phenotype and genotype values and zhave ero genotypes.
`p`	The p value for testing the coefficient for genotype is not zero.
`beta`	the beta coefficient for genotype.
`se`	the standarr error for the beta coefficient for genotype.

`test.associations` *Association Test Dispacher*

Description

The main function to identify the right kind of association test to perfrom based on phenotype variable information.

Usage

```
test.associations(opt, vl, currentVar, currentVarShort, thisdata, phenoStartIdx)
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list.
`currentVar`	The phenotype/variable to be tested, in the form of FIELDID_INSTANCE. For example 21022_0.
`currentVarShort`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`phenoStartIdx`	The column index of the first phenotype.

Details

For a given variable identified by its FIELDID, it reads from variable list to decide if to run association test and chooses the approriate associtiion test based on variable value type. Currently, recognized variable types are "Integer", "Continuous", "Categorical single", and "Categorical multiple" and all other types are ignored.

Value

Return nothing and all output and lof information are written to output directory specified by user.

`test.categorical.multiple`
Association Test dispacther for "Categorical multiple" value type

Description

The function takes the declared "Categorical multiple" value type of a field and validates/decides on the actual type of association test and perform the test accordingly.

Usage

```
test.categorical.multiple(opt, vl, varName, varType, thisdata, phenoStartIdx)
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list.
`varName`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`varType`	The value type of the variable. Should always be "Categorical multiple" here and used for logging purpose only.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`phenoStartIdx`	The column index of the first phenotype.

Details

Performs variable processing for categorical (multiple) fields, namely

1. Reassign values as specified in data coding file
2. Generate binary variable for each category in field, restricting to correct set of participants as specified
3. Replace missing codes - we assume values < 0 are missing for categorical (single) variables
4. Check derived variable has at least 10 cases in each group
5. Call binary.logistic.regression function for this derived binary variable

Value

Return nothing and all output and lof information are written to output directory specified by user.

`test.categorical.ordered` *Performs ordered logistic regression test*

Description

Performs ordered logistic regression test and saves results in ordered logistic results file.

Usage

```
test.categorical.ordered(opt, vl, varName, varType, thisdata, phenoStartIdx,
                         orderStr = "")
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list.
`varName`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`varType`	The value type of the variable. Should always be "Categorical single" here and used for logging purpose only.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`phenoStartIdx`	The column index of the first phenotype.
`orderStr`	The Order String for the variable.

Details

The variable will be skipped if less than 500 samples have value or there are too many levels.

Value

Return nothing and all output and lof information are written to output directory specified by user.

`test.categorical.single` *Association Test dispacther for "Categorical single" value type*

Description

The function takes the declared "Categorical single" value type of a field and validates/decides on the actual type of association test and perform the test accordingly.

Usage

```
test.categorical.single(opt, vl, varName, varType, thisdata, phenoStartIdx)
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list.
`varName`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`varType`	The value type of the variable. Should always be "Categorical single" here and used for logging purpose only.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`phenoStartIdx`	The column index of the first phenotype.

Details

Performs variable processing for categorical (single) fields, namely

1. Reassign values as specified in data coding information file
2. Reorder categories for ordered fields
3. Remove values with <10 cases
4. Deterimine correct test to perform, either binary, ordered or unordered

Value

Return nothing and all output and lof information are written to output directory specified by user.

`test.categorical.unordered` *Perform unordered categorical phenotype with multinomial regression*

Description

Perform unordered categorical phenotype with multinomial regression and saves this result in the multinomial logistic results file

Usage

```
test.categorical.unordered(opt, vl, varName, varType, thisdata, phenoStartIdx)
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list.
`varName`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`varType`	The value type of the variable. Should always be "Categorical single" here and used for logging purpose only.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`phenoStartIdx`	The column index of the first phenotype.

Details

The variable will be skipped if less than 500 samples have value.

Value

Return nothing and all output and lof information are written to output directory specified by user.

`test.continuous` *Association Test dispacther for Continuous value type*

Description

The function takes the declared "Continuous" value type of a field and validates/decides on the actual type of association test and perform the test accordingly.

Usage

```
test.continuous(opt, vl, varName, varType, thisdata, phenoStartIdx)
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list.
`varName`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`varType`	The value type of the variable. Should always be "Continuous" here and used for logging purpose only.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`phenoStartIdx`	The column index of the first phenotype.

Details

Processing integer fields, namely:

1. Reassigning values as specified in the data code information file
2. Generate a single value if there are several values (arrays) by taking the mean
3. Treating this field as ordinal categorical if >20% examples with same value
4. Otherwise Treat as continuous (with Rank-based Inverse Normal Transformation) if 500 or more samples have values
5. In the case of 3), treat it as binary if only two types of values, or ordinal categorical if its values can be binned approximately into three equl bins.
6. In the case of 5), if more than 3 type of values and can not be binned reasonably into 3 euqal bins, merge them into two bins ans treat it as binary, or give up.

Value

Return nothing and all output and lof information are written to output directory specified by user.

`test.integer` *Association Test dispacther for Integer value type*

Description

The function takes the declared "Integer" value type of a field and validates/decides on the actual type of association test and perform the test accordingly.

Usage

```
test.integer(opt, vl, varName, varType, thisdata, phenoStartIdx)
```

Arguments

`opt`	The list of input options provided by user.
`vl`	The list that holds input phenotype variable list and input data coding list.
`varName`	The phenotype/variable to be tested, in the form of FIELDID. For example 21022.
`varType`	The value type of the variable. Should always be "INTEGER" here and used for logging purpose only.
`thisdata`	The data frame object holds all variables including phenotypes, confounders, depeinding variables and trait of interest.
`phenoStartIdx`	The column index of the first phenotype.

Details

Processing integer fields, namely:

1. Reassigning values as specified in the data code information file
2. Generate a single value if there are several values (arrays) by taking the mean
3. Treating this field as continuous if at least 20 distinct values
4. Otherwise treat as binary or ordered categorical if 2 or more than two values

Value

Return nothing and all output and lof information are written to output directory specified by user.

`validate.phenotype.input.header` *Validate the contents of the phenotype file*

Description

Validate the contents of the phenotype file

Usage

```
validate.phenotype.input.header(opt)
```

Arguments

`opt`	The list of input options provided by user.

Details

This function validates the input phenotype file based on user options.

1. check if user id field exists in pheno file
2. check if phenotype file contains the required age colunn: x21022_0_0
3. check if phenotype file contains the required sex colunn: x31_0_0
4. check if phenotype file contains the required genetic batch colunn: x22000_0_0, when genetic option is used
5. check if phenotype file contains the required genetic principal component colunns(1 to 10): x22009_0_, when sensitivity and genetic options are used
6. check if phenotype file contains the required assessment centre colunn: x54_0_0, when sensitivity option is used

Value

No return values and reports error and stops the program if the validation fails.

Examples

```
#validate.phenotype.input.header(opt)
```

`validate.trait.input.header` *Validate the contents of the trait of interest file*

Description

Validate the contents of the trait of interest file.

Usage

```
validate.trait.input.header(opt)
```

Arguments

`opt`	The list of input options provided by user.

Details

This function validates the contents of the trait of interest file based on user options. A trait of interest ca nbe either from phrenotype file or trait of interest file.

1. check if user id field exists in pheno file or traint of interest file if the trait of interest file is provided.
2. check if phenotype file or trait of interest file contains the required trait of interest variable column.

Value

No return values and reports error and stops the program if the validation fails.

Examples

```
#validate.trait.input.header(opt)
```

Index