Package: CAMeRa (via r-universe)

November 30, 2024

Title CAMeRa (Cross Ancestral Mendelian Randomisation)

Version 0.1.0

Description CAMERA estimates joint causal effect in multiple ancestries and detects pleiotropy via the zero relevance model.

License MIT + file LICENSE

Depends R (>= 4.1.0)

Imports dplyr, ggplot2, ieugwasr (>= 0.1.8), magrittr, tidyr

Suggests RadialMR, winnerscurse, TwoSampleMR, lavaan, susieR, genetics.binaRies, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Remotes amandaforde/winnerscurse, explodecomputer/genetics.binaRies, MRCIEU/ieugwasr, MRCIEU/TwoSampleMR, WSpiller/RadialMR

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Config/pak/sysreqs libicu-dev libssl-dev

Repository https://mrcieu.r-universe.dev

RemoteUrl https://github.com/MRCIEU/CAMERA

RemoteRef HEAD

RemoteSha fc705d5e9c31d1f76dbe4477057784a4f8f8620d

Contents

CAMERA *R6 class for CAMERA*

Description

A simple wrapper function. Using a summary set, identify set of instruments for the traits, and peform SEM MR to test the association across the population.

Methods

Public methods:

- CAMERA$import()
- CAMERA$assign()
- CAMERA$import_from_local()
- CAMERA$new()
- CAMERA$instrument_heterogeneity()
- CAMERA$estimate_instrument_specificity()
- CAMERA$replication_evaluation()
- CAMERA$check_phenotypes()
- CAMERA$cross_estimate()
- CAMERA$plot_cross_estimate()
- CAMERA$extract_instruments()
- CAMERA$extract_instrument_regions()
- CAMERA$scan_regional_instruments()
- CAMERA$plot_regional_instruments_maxz()
- CAMERA$regional_ld_matrices()
- CAMERA$susie_finemap_regions()
- CAMERA$paintor_finemap_regions()
- CAMERA$MsCAVIAR_finemap_regions()
- CAMERA$fema_regional_instruments()
- CAMERA$plot_regional_instruments()
- CAMERA$get_metadata()
- CAMERA$estimate_instrument_heterogeneity_per_variant()
- CAMERA$mrgxe()
- CAMERA$mrgxe_plot()
- CAMERA$mrgxe_plot_variant()
- CAMERA$make_outcome_data()
- CAMERA$make_outcome_local()
- CAMERA$harmonise()
- CAMERA$set_summary()
- CAMERA$pleiotropy()

- CAMERA$plot_pleiotropy()
- CAMERA$plot_pleiotropy_heterogeneity()
- CAMERA$perform_basic_sem()
- CAMERA$runsem()
- CAMERA$standardise_data()
- CAMERA$clone()

Method `import()`**:** Migrate the results from a previous CAMERA

Usage:

```
CAMERA$import(x)
```

Arguments:

`x` R6 Environment created for CAMERA. Default = x

Method `assign()`**:**

Usage:

```
CAMERA$assign(...)
```

Method `import_from_local()`**:**

Usage:

```
CAMERA$import_from_local(
  instrument_raw,
  instrument_outcome,
  instrument_regions,
  instrument_outcome_regions,
  exposure_ids,
  outcome_ids,
  pops,
  ...
)
```

Method `new()`**:** Create a new dataset and initialise an R interface

Usage:

```
CAMERA$new(
  exposure_ids = NULL,
  outcome_ids = NULL,
  pops = NULL,
  bfiles = NULL,
  plink = NULL,
  radius = NULL,
  clump_pop = NULL,
  x = NULL
)
```

Arguments:

`exposure_ids` Exposures IDs obtained from IEU GWAS database (https://gwas.mrcieu.ac.uk/) for each population

`outcome_ids` Outcome IDs obtained from IEU GWAS database (https://gwas.mrcieu.ac.uk/) for each population

`pops` Ancestry information for each population (i.e. AFR, AMR, EUR, EAS, SAS)

`bfiles` Locations of LD reference files for each population (Download from: http://fileserve.mrcieu.ac.uk/ld/1kg.v3.tgz)

`plink` Location of executable plink (ver.1.90 is recommended)

`radius` Genomic window size to extract SNPs

`clump_pop` Reference population for clumping

`x` Import data where available

Method `instrument_heterogeneity()`**:**

Usage:

```
CAMERA$instrument_heterogeneity(
  instrument = self$instrument_raw,
  alpha = "bonferroni",
  method = "ivw",
  outlier_removal = FALSE
)
```

Method `estimate_instrument_specificity()`**:**

Usage:

```
CAMERA$estimate_instrument_specificity(
  instrument,
  alpha = "bonferroni",
  winnerscurse = FALSE
)
```

Method `replication_evaluation()`**:**

Usage:

```
CAMERA$replication_evaluation(
  instrument = self$instrument_raw,
  ld = self$ld_matrices
)
```

Method `check_phenotypes()`**:**

Usage:

```
CAMERA$check_phenotypes(ids = self$exposure_ids)
```

Method `cross_estimate()`**:**

Usage:

```
CAMERA$cross_estimate(dat = self$harmonised_dat)
```

Method `plot_cross_estimate()`**:**

Usage:

```
CAMERA$plot_cross_estimate(est = self$mrres, qj_alpha = 0.05)
```

Method `extract_instruments()`**:**

Usage:

```
CAMERA$extract_instruments(exposure_ids = self$exposure_ids, ...)
```

Method `extract_instrument_regions()`**:**

Usage:

```
CAMERA$extract_instrument_regions(
  radius = self$radius,
  instrument_raw = self$instrument_raw,
  exposure_ids = self$exposure_ids
)
```

Method `scan_regional_instruments()`**:**

Usage:

```
CAMERA$scan_regional_instruments(
  instrument_raw = self$instrument_raw,
  instrument_regions = self$instrument_regions
)
```

Method `plot_regional_instruments_maxz()`**:**

Usage:

```
CAMERA$plot_regional_instruments_maxz(
  instrument_region_zscores = self$instrument_region_zscores,
  instruments = self$instrument_raw,
  region = 1:min(10, nrow(instruments)),
  comparison = FALSE
)
```

Method `regional_ld_matrices()`**:**

Usage:

```
CAMERA$regional_ld_matrices(
  instrument_regions = self$instrument_regions,
  bfiles = self$bfiles,
  pops = self$pops,
  plink = self$plink
)
```

Method `susie_finemap_regions()`**:**

Usage:

```
CAMERA$susie_finemap_regions(
  dat = self$instrument_regions,
  ld = self$ld_matrices
)
```

Method `paintor_finemap_regions()`**:**

Usage:

```
CAMERA$paintor_finemap_regions(
  region = self$instrument_regions,
  ld = self$ld_matrices,
  PAINTOR = "PAINTOR",
  workdir = tempdir()
)
```

Method `MsCAVIAR_finemap_regions()`**:**

Usage:

```
CAMERA$MsCAVIAR_finemap_regions(
  region = self$instrument_regions,
  ld = self$ld_matrices,
  MsCAVIAR = "MsCAVIAR",
  workdir = tempdir()
)
```

Method `fema_regional_instruments()`**:**

Usage:

```
CAMERA$fema_regional_instruments(
  method = "fema",
  instrument_regions = self$instrument_regions,
  instrument_raw = self$instrument_raw,
  n = self$exposure_metadata$sample_size
)
```

Method `plot_regional_instruments()`**:**

Usage:

```
CAMERA$plot_regional_instruments(
  region,
  instrument_regions = self$instrument_regions,
  meta_analysis_regions = self$instrument_fema_regions
)
```

Method `get_metadata()`**:**

Usage:

```
CAMERA$get_metadata(
  exposure_ids = self$exposure_ids,
  outcome_ids = self$outcome_ids
)
```

Method `estimate_instrument_heterogeneity_per_variant()`**:**

Usage:

```
CAMERA$estimate_instrument_heterogeneity_per_variant(dat = self$harmonised_dat)
```

Method `mrgxe()`**:**

Usage:

```
CAMERA$mrgxe(
  dat = self$harmonised_dat,
 variant_list = subset(self$instrument_heterogeneity_per_variant, Qfdr < 0.05)$SNP,
  nboot = 100
)
```

Method `mrgxe_plot()`**:**

Usage:

```
CAMERA$mrgxe_plot(mrgxe_res = self$mrgxe_res)
```

Method `mrgxe_plot_variant()`**:**

Usage:

```
CAMERA$mrgxe_plot_variant(
 variant = self$mrgxe_res %>% dplyr::filter(p.adjust(a_pval, "fdr") < 0.05) %>% {

      .$SNP
 },
  dat = self$harmonised_dat
)
```

Method `make_outcome_data()`**:**

Usage:

```
CAMERA$make_outcome_data(exp = self$instrument_raw, p_exp = 0.05/nrow(exp))
```

Method `make_outcome_local()`**:**

Usage:

```
CAMERA$make_outcome_local(
  exp = self$instrument_raw,
  out = self$instrument_outcome_regions,
  p_exp = 0.05/nreow(exp)
)
```

Method `harmonise()`**:**

Usage:

```
CAMERA$harmonise(exp = self$instrument_raw, out = self$instrument_outcome)
```

Method `set_summary()`**:**

Usage:

```
CAMERA$set_summary()
```

Method `pleiotropy()`**:**

Usage:

```
CAMERA$pleiotropy(harmonised_dat = self$harmonised_dat, mrres = self$mrres)
```

Method `plot_pleiotropy()`**:**

Usage:

```
CAMERA$plot_pleiotropy(dat = self$pleiotropy_outliers)
```

Method `plot_pleiotropy_heterogeneity()`**:**

Usage:

```
CAMERA$plot_pleiotropy_heterogeneity(
  dat = self$pleiotropy_Q_outliers,
  pthresh = 0.05
)
```

Method `perform_basic_sem()`**:**

Usage:

```
CAMERA$perform_basic_sem(harmonised_dat = self$harmonised_dat_sem)
```

Method `runsem()`**:**

Usage:

```
CAMERA$runsem(model, data, modname)
```

Method `standardise_data()`**:**

Usage:

```
CAMERA$standardise_data(
  dat = self$instrument_raw,
  standardise_unit = FALSE,
  standardise_scale = FALSE,
  scaling_method = "simple_mode"
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CAMERA$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CAMERA_local` *CAMERA_local class*

Description

A simple wrapper function for importing data from local files for use with the CAMERA class.

Methods

Public methods:

- CAMERA_local$new()
- CAMERA_local$standardise()
- CAMERA_local$read_file()
- CAMERA_local$pool_tophits()
- CAMERA_local$organise_data()
- CAMERA_local$fixed_effects_meta_analysis_fast()
- CAMERA_local$organise()
- CAMERA_local$clone()

Method `new()`**:** Create a new dataset and initialise an R interface

Usage:

```
CAMERA_local$new(
  metadata,
  ld_ref,
  plink_bin,
  mc.cores = 1,
  radius = 25000,
  pthresh = 5e-08,
  minmaf = 0.01
)
```

Arguments:

`metadata` Data frame with information about the data. One row per dataset. See details for info on columns

`ld_ref` Data frame with two columns - pop = population (referencing the pop values in metadata), bfile = path to plink file for that reference

`plink_bin` Location of executable plink (ver.1.90 is recommended)

`radius` Genomic window size to extract SNPs

`pthresh` P-value threshold for instrument inclusion

`minmaf` Minimum allelel frequency per dataset

`clump_pop` Reference population for clumping

Method `standardise()`**:**

Usage:

```
CAMERA_local$standardise(
  d,
  ea_col = "ea",
  oa_col = "oa",
  beta_col = "beta",
  eaf_col = "eaf",
  chr_col = "chr",
  pos_col = "pos",
  vid_col = "vid"
)
```

Method `read_file()`**:**

Usage:

```
CAMERA_local$read_file(m, minmaf = 0.01)
```

Method `pool_tophits()`**:**

Usage:

```
CAMERA_local$pool_tophits(
  rawdat,
  tophits,
  metadata,
  radius = 250000,
  pthresh = 5e-08,
  mc.cores = 10
)
```

Method `organise_data()`**:**

Usage:

```
CAMERA_local$organise_data(
  metadata = self$metadata,
  plink_bin = self$plink_bin,
  ld_ref = self$ld_ref,
  pthresh = self$pthresh,
  minmaf = self$minmaf,
  radius = self$radius,
  mc.cores = self$mc.cores
)
```

Method `fixed_effects_meta_analysis_fast()`**:**

Usage:

```
CAMERA_local$fixed_effects_meta_analysis_fast(beta_mat, se_mat)
```

Method `organise()`**:**

Usage:

```
CAMERA_local$organise()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CAMERA_local$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`egger_bootstrap` *Perform MR GxE*

Description

For a single variant estiamted in different sub groups.

Usage

```
egger_bootstrap(b_gx, se_gx, b_gy, se_gy, nboot = 1000)
```

Arguments

`b_gx`	Vector of instrument-exposure associations, one for each sub group
`se_gx`	Vector of standard errors to b_gx
`b_gy`	Vector of instrument-outcome associations, one for each sub group
`se_gy`	Vector of standard errors for b_gy
`nboot`	Number of bootstraps. Default=1000

Details

Estimate the degree of pleiotropy using MR GxE. This method uses a negative control type approach based on an assumption that the instrument-exposure association is uncorrelated with the pleiotropic effect. Therefore, as the instrument-exposure association reduces in magnitude, the effect on the outcome will reduce towards an intercept term which represents the pleiotropic effect.

Standard errors are obtained from parametric bootstrap

Value

List

- a = intercept estimate (pleiotropy)
- b = slope estimate (b_iv effect)
- a_se = standard error of intercept
- b_se = standard error of slope
- a_pval = p-value of intercept estimate
- b_pval = p-value of slope estimate
- a_mean = mean value of intercept from bootstraps
- b_mean = mean value of slope estimates from bootstraps

`fixed_effects_meta_analysis` *Perform fixed effects meta analysis for one association*

Description

Perform fixed effects meta analysis for one association

Usage

```
fixed_effects_meta_analysis(beta_vec, se_vec, infl = 10000)
```

Arguments

`beta_vec`	Vector of betas
`se_vec`	Vector of ses
`infl`	Inflation factor - how much larger is the estimate than the estimate of the tightest SE - for use in removing unreliable estimates

Value

list of results

`fixed_effects_meta_analysis_fast` *Fixed effects meta analysis vectorised across multiple SNPs*

Description

Assumes effects across studies are all on the same scale

Usage

```
fixed_effects_meta_analysis_fast(beta_mat, se_mat)
```

Arguments

`beta_mat`	Matrix of betas - rows are SNPs, columns are studies
`se_mat`	Matrix of SEs - rows are SNPs, columns are studies

Value

list of meta analysis betas and SEs

identify_blownup_estimates *Identify blown up estimates*

Description

Sometimes estimates appear unstable. They are likely unreliable and best to not use for heterogeneity analyses etc.

Usage

```
identify_blownup_estimates(b, se, infl)
```

Arguments

b	Vector of betas
se	Vector of SEs
infl	Inflation factor - how much larger is the estimate than the estimate of the tightest SE

Value

index of betas to remove

prop_overlap *Estimate expected vs observed replication of effects between discovery and replication datasets*

Description

Taken from Okbay et al 2016. Under the assumption that all discovery effects are unbiased, what fraction of associations would replicate in the replication dataset, given the differential power of the discovery and replication datasets. Uses standard error of the replication dataset to account for differences in sample size and distribution of independent variable

Usage

```
prop_overlap(b_disc, b_rep, se_disc, se_rep, alpha)
```

Arguments

b_disc	Vector of discovery betas
b_rep	Vector of replication betas
se_disc	Vector of discovery standard errors
se_rep	Vector of replication standard errors
alpha	Nominal replication significance threshold

Value

List of results

- res: aggregate expected replication rate vs observed replication rate
- variants: per variant expected replication rates

`z_meta_analysis` *P-value based meta analysis*

Description

Uses weighted Z scores following advice from https://onlinelibrary.wiley.com/doi/full/10.1111/j.1420-9101.2005.00917.x Suggested weights are 1/se^2 However se is scale dependent and it would be ideal to avoid scale issues at this stage So using instead calculate expected se based on n and af. Assumes continuous traits in the way it uses n (i.e. not case control aware at the moment)

Usage

```
z_meta_analysis(beta_mat, se_mat, n, eaf_mat)
```

Arguments

`beta_mat`	Matrix of betas - rows are SNPs, columns are studies
`se_mat`	Matrix of SEs - rows are SNPs, columns are studies
`n`	Vector of sample sizes for each
`eaf_mat`	Matrix of allele frequencies - rows are SNPs, columns are studies

Index