Package: ApplyPolygenicScore (via r-universe)

March 6, 2025

Type Package

Title Utilities for the Application of a Polygenic Score to a VCF

Version 3.0.1

Description Simple and transparent parsing of genotype/dosage data from an input Variant Call Format (VCF) file, matching of genotype coordinates to the component Single Nucleotide Polymorphisms (SNPs) of an existing polygenic score (PGS), and application of SNP weights to dosages for the calculation of a polygenic score for each individual in accordance with the additive weighted sum of dosages model. Methods are designed in reference to best practices described by Collister, Liu, and Clifton (2022) <doi:10.3389/fgene.2022.818574>.

Depends R (>= 4.2.0)

Imports vcfR, pROC, data.table, reshape2, BoutrosLab.plotting.general, lattice

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

License GPL-2

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Paul Boutros [cre], Nicole Zeltser [aut] (<https://orcid.org/0000-0001-7246-2771>), Rachel Dang [ctb]

Maintainer Paul Boutros <PBoutros@mednet.ucla.edu>

Date/Publication 2025-03-05 15:50:15 UTC

Config/pak/sysreqs libicu-dev libjpeg-dev libpng-dev

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

RemoteUrl https://github.com/cran/ApplyPolygenicScore

RemoteRef HEAD

RemoteSha af9179b56e63ac8e69e8da3390748378c2d70f72

Contents

`apply.polygenic.score` *Apply polygenic score to VCF data*

Description

Apply a polygenic score to VCF data.

Usage

```
apply.polygenic.score(
  vcf.data,
  pgs.weight.data,
  phenotype.data = NULL,
  phenotype.analysis.columns = NULL,
  correct.strand.flips = TRUE,
  remove.ambiguous.allele.matches = FALSE,
  remove.mismatched.indels = FALSE,
  output.dir = NULL,
  file.prefix = NULL,
  missing.genotype.method = "mean.dosage",
```

```
  use.external.effect.allele.frequency = FALSE,
  n.percentiles = NULL,
  analysis.source.pgs = NULL,
  validate.inputs.only = FALSE
)
```

Arguments

`vcf.data` A data.frame containing VCF genotype data as formatted by `import.vcf()`.

`pgs.weight.data` A data.frame containing PGS weight data as formatted by `import.pgs.weight.file()`.

`phenotype.data` A data.frame containing phenotype data. Must have an Indiv column matching vcf.data. Default is `NULL`.

`phenotype.analysis.columns` A character vector of phenotype columns from phenotype.data to analyze in a regression analsyis. Default is `NULL`. Phenotype variables are automatically classified as continuous, binary, or neither based on data type and number of unique values. The calculated PGS is associated with each phenotype variable using linear or logistic regression for continuous or binary phenotypes, respectively. See `run.pgs.regression` for more details. If no phenotype.analysis.columns are provided, no regression analysis is performed.

`correct.strand.flips` A logical indicating whether to check PGS weight data/VCF genotype data matches for strand flips and correct them. Default is `TRUE`. The PGS catalog standard column `other_allele` in `pgs.weight.data` is required for this check.

`remove.ambiguous.allele.matches` A logical indicating whether to remove PGS variants with ambiguous allele matches between PGS weight data and VCF genotype data. Default is `FALSE`. The PGS catalog standard column `other_allele` in `pgs.weight.data` is required for this check.

`remove.mismatched.indels` A logical indicating whether to remove indel variants that are mismatched between PGS weight data and VCF genotype data. Default is `FALSE`. The PGS catalog standard column `other_allele` in `pgs.weight.data` is required for this check.

`output.dir` A character string indicating the directory to write output files. Separate files are written for per-sample pgs results and optional regression results. Files are tab-separate .txt files. Default is NULL in which case no files are written.

`file.prefix` A character string to prepend to the output file names. Default is `NULL`.

`missing.genotype.method` A character string indicating the method to handle missing genotypes. Options are "mean.dosage", "normalize", or "none". Default is "mean.dosage".

`use.external.effect.allele.frequency` A logical indicating whether to use an external effect allele frequency for calculating mean dosage when handling missing genotypes. Default is `FALSE`. Provide allele frequency as a column is `pgs.weight.data` named `allelefrequency_effect`.

`n.percentiles` An integer indicating the number of percentiles to calculate for the PGS. Default is `NULL`.

`analysis.source.pgs`
A character string indicating the source PGS for percentile calculation and regression analyses. Options are "mean.dosage", "normalize", or "none". When not specified, defaults to `missing.genotype.method` choice and if more than one PGS missing genotype method is chosen, calculation defaults to the first selection.

`validate.inputs.only`
A logical indicating whether to only perform input data validation checks without running PGS application. If no errors are triggered, a message is printed and TRUE is returned. Default is `FALSE`.

Value

A list containing per-sample PGS output and per-phenotype regression output if phenotype analysis columns are provided.

Output Structure

The outputed list contains the following elements:

- pgs.output: A data.frame containing the PGS per sample and optional phenotype data.
- regression.output: A data.frame containing the results of the regression analysis if phenotype.analysis.columns are provided, otherwise `NULL`.

pgs.output columns:

- `Indiv`: A character string indicating the sample ID.
- `PGS`: A numeric vector indicating the PGS per sample. (only if missing.genotype.method includes "none")
- `PGS.with.normalized.missing`: A numeric vector indicating the PGS per sample with missing genotypes normalized. (only if missing.genotype.method includes "normalize")
- `PGS.with.replaced.missing`: A numeric vector indicating the PGS per sample with missing genotypes replaced by mean dosage. (only if missing.genotype.method includes "mean.dosage")
- `percentile`: A numeric vector indicating the percentile rank of the PGS.
- `decile`: A numeric vector indicating the decile rank of the PGS.
- `quartile`: A numeric vector indicating the quartile rank of the PGS.
- `percentile.X`: A numeric vector indicating the user-specified percentile rank of the PGS where "X" is substituted by `n.percentiles`. (only if `n.percentiles` is specified)
- `n.missing.genotypes`: A numeric vector indicating the number of missing genotypes per sample.
- `percent.missing.genotypes`: A numeric vector indicating the percentage of missing genotypes per sample.
- All columns in `phenotype.data` if provided.

regression.output columns:

- phenotype: A character vector of phenotype names.

- `model`: A character vector indicating the regression model used. One of "logistic.regression" or "linear.regression".
- `beta`: A numeric vector indicating the beta coefficient of the regression analysis.
- `se`: A numeric vector indicating the standard error of the beta coefficient.
- `p.value`: A numeric vector indicating the p-value of the beta coefficient.
- `r.squared`: A numeric vector indicating the r-squared value of linear regression analysis. NA for logistic regression.
- `AUC`: A numeric vector indicating the area under the curve of logistic regression analysis. NA for linear regression.

PGS Calculation

PGS for each individual *i* is calculated as the sum of the product of the dosage and beta coefficient for each variant in the PGS:

$$PGS_i = \sum_{m=1}^{M} (\beta_m \times dosage_{im})$$

Where *m* is a PGS component variant out of a total *M* variants.

Missing Genotype Handling

VCF genotype data are matched to PGS data by chromosome and position. If a SNP cannot be matched by genomic coordinate, an attempt is made to match by rsID (if available). If a SNP from the PGS weight data is not found in the VCF data after these two matching attempts, it is considered a cohort-wide missing variant.

Missing genotypes (in individual samples) among successfully matched variants are handled by three methods:

`none`: Missing genotype dosages are excluded from the PGS calculation. This is equivalent to assuming that all missing genotypes are homozygous for the non-effect allele, resulting in a dosage of 0.

`normalize`: Missing genotypes are excluded from score calculation but the final score is normalized by the number of non-missing alleles. The calculation assumes a diploid genome:

$$PGS_i = \frac{\sum (\beta_m \times dosage_{im})}{P_i * M_{non-missing}}$$

Where *P* is the ploidy and has the value `2` and $M_{non-missing}$ is the number of non-missing genotypes.

`mean.dosage`: Missing genotype dosages are replaced by the mean population dosage of the variant which is calculated as the product of the effect allele frequency *EAF* and the ploidy of a diploid genome:

$$\overline{dosage_k} = EAF_k * P$$

where *k* is a PGS component variant that is missing in between 1 and n-1 individuals in the cohort and *P* = ploidy = 2 This dosage calculation holds under assumptions of Hardy-Weinberg equilibrium. By default, the effect allele frequency is calculated from the provided VCF data. For variants that are missing in all individuals (cohort-wide), dosage is assumed to be zero (homozygous non-reference) for all individuals. An external allele frequency can be provided in the `pgs.weight.data` as a column named `allelefrequency_effect` and by setting `use.external.effect.allele.frequency` to `TRUE`.

Multiallelic Site Handling

If a PGS weight file provides weights for multiple effect alleles, the appropriate dosage is calculated for the alleles that each individual carries. It is assumed that multiallelic variants are encoded in the same row in the VCF data. This is known as "merged" format. Split multiallelic sites are not accepted. VCF data can be formatted to merged format using external tools for VCF file manipulation.

Allele Mismatch Handling Variants from the PGS weight data are merged with records in the VCF data by genetic coordinate. After the merge is complete, there may be cases where the VCF reference (REF) and alternative (ALT) alleles do not match their conventional counterparts in the PGS weight data (other allele and effect allele, respectively). This is usually caused by a strand flip: the variant in question was called against opposite DNA reference strands in the PGS training data and the VCF data. Strand flips can be detected and corrected by flipping the affected allele to its reverse complement. `apply.polygenic.score` uses `assess.pgs.vcf.allele.match` to assess allele concordance, and is controlled through the following arguments:

- `correct.strand.flips`: When `TRUE`, detected strand flips are corrected by flipping the affected value in the `effect_allele` column prior to dosage calling.
- `remove.ambiguous.allele.matches`: Corresponds to the `return.ambiguous.as.missing` argument in `assess.pgs.vcf.allele.match`. When `TRUE`, non-INDEL allele mismatches that cannot be resolved (due to palindromic alleles or causes other than strand flips) are removed by marking the affected value in the `effect_allele` column as missing prior to dosage calling and missing genotype handling. The corresponding dosage is set to NA and the variant is handled according to the chosen missing genotype method.
- `remove.mismatched.indels`: Corresponds to the `return.indels.as.missing` argument in `assess.pgs.vcf.allele.match`. When `TRUE`, INDEL allele mismatches (which cannot be assessed for strand flips) are removed by marking the affected value in the `effect_allele` column as missing prior to dosage calling and missing genotype handling. The corresponding dosage is set to NA and the variant is handled according to the chosen missing genotype method.

Note that an allele match assessment requires the presence of both the `other_allele` and `effect_allele` in the PGS weight data. The `other_allele` column is not required by the PGS Catalog, and so is not always available.

Examples

```
# Example VCF
vcf.path <- system.file(
    'extdata',
    'HG001_GIAB.vcf.gz',
    package = 'ApplyPolygenicScore',
    mustWork = TRUE
    );
vcf.import <- import.vcf(vcf.path);

# Example pgs weight file
pgs.weight.path <- system.file(
    'extdata',
    'PGS000662_hmPOS_GRCh38.txt.gz',
```

```
        package = 'ApplyPolygenicScore',
        mustWork = TRUE
        );
    pgs.import <- import.pgs.weight.file(pgs.weight.path);

    pgs.data <- apply.polygenic.score(
        vcf.data = vcf.import$dat,
        pgs.weight.data = pgs.import$pgs.weight.data,
        missing.genotype.method = 'none'
        );

    # Specify different methods for handling missing genotypes
    pgs.import$pgs.weight.data$allelefrequency_effect <- rep(0.5, nrow(pgs.import$pgs.weight.data));
    pgs.data <- apply.polygenic.score(
        vcf.data = vcf.import$dat,
        pgs.weight.data = pgs.import$pgs.weight.data,
        missing.genotype.method = c('none', 'mean.dosage', 'normalize'),
        use.external.effect.allele.frequency = TRUE
        );

    # Specify allele mismatch handling
    pgs.data <- apply.polygenic.score(
       vcf.data = vcf.import$dat,
       pgs.weight.data = pgs.import$pgs.weight.data,
       correct.strand.flips = TRUE,
       remove.ambiguous.allele.matches = TRUE,
       remove.mismatched.indels = FALSE
       );

    # Provide phenotype data for basic correlation analysis
    phenotype.data <- data.frame(
        Indiv = unique(vcf.import$dat$Indiv),
        continuous.phenotype = rnorm(length(unique(vcf.import$dat$Indiv))),
        binary.phenotype = sample(
            c('a', 'b'),
            length(unique(vcf.import$dat$Indiv)),
            replace = TRUE
            )
        );

    pgs.data <- apply.polygenic.score(
        vcf.data = vcf.import$dat,
        pgs.weight.data = pgs.import$pgs.weight.data,
        phenotype.data = phenotype.data
        );

    # Only run validation checks on input data and report back
    apply.polygenic.score(
        vcf.data = vcf.import$dat,
        pgs.weight.data = pgs.import$pgs.weight.data,
        validate.inputs.only = TRUE
        );
```

assess.pgs.vcf.allele.match *Assess PGS allele match to VCF allele*

Description

Assess whether PGS reference and effect alleles match provided VCF reference and alternative alleles. Mismatches are checked for potential switching of effect and reference PGS alleles (cases where the effect allele is the REF VCF allele) and are evaluated for DNA strand flips (by flipping the PGS alleles). INDEL alleles are not supported for strand flip assessment.

Usage

```
assess.pgs.vcf.allele.match(
  vcf.ref.allele,
  vcf.alt.allele,
  pgs.ref.allele,
  pgs.effect.allele,
  return.indels.as.missing = FALSE,
  return.ambiguous.as.missing = FALSE
)
```

Arguments

`vcf.ref.allele` A character vector of singular VCF reference (REF) alleles.

`vcf.alt.allele` A character vector of VCF alternative (ALT) alleles. Multiple alleles at a multi-allelic site must be separated by commas.

`pgs.ref.allele` A character vector of singular PGS reference alleles aka "non-effect" or "other" alleles.

`pgs.effect.allele` A character vector of singular PGS effect alleles.

`return.indels.as.missing` A logical value indicating whether to return NA for INDEL alleles with detected mismatches. Default is `FALSE`.

`return.ambiguous.as.missing` A logical value indicating whether to return NA for ambiguous cases where both a strand flip and effect switch are possible, or no strand flip is detected and a mismatch cannot be resolved. Default is `FALSE`.

Value

A list containing the match assessment, a new PGS effect allele, and a new PGS other allele.

Output Structure

The outputed list contains the following elements:

- `match.status`: A character vector indicating the match status for each pair of allele pairs. Possible values are `default_match`, `effect_switch`, `strand_flip`, `effect_switch_with_strand_flip`, `ambiguous_flip`, `indel_mismatch`, and `unresolved_mismatch`.
- `new.pgs.effect.allele`: A character vector of new PGS effect alleles based on the match status. If the match status is `default_match`, `effect_switch` or `missing_allele`, the original PGS effect allele is returned. If the match status is `strand_flip` or `effect_switch_with_strand_flip` the flipped PGS effect allele is returned. If the match status is `ambiguous_flip`, `indel_mismatch`, or `unresolved_mismatch`, the return value is either the original allele or NA as dictated by the `return.indels.as.missing` and `return.ambiguous.as.missing` parameters.
- `new.pgs.other.allele`: A character vector of new PGS other alleles based on the match status, following the same logic as `new.pgs.effect.allele`.

The match.status output indicates the following:

- `default_match`: The default PGS reference allele matches the VCF REF allele and the default PGS effect allele matches one of the VCF ALT alleles.
- `effect_switch`: The PGS effect allele matches the VCF REF allele and the PGS reference allele matches one of the VCF ALT alleles.
- `strand_flip`: The PGS reference and effect alleles match their respective VCF pairs when flipped.
- `effect_switch_with_strand_flip`: The PGS effect allele matches the VCF REF allele and the PGS reference allele matches one of the VCF ALT alleles when flipped.
- `ambiguous_flip`: Both an effect switch and a strand flip have been detected. This is an ambiguous case caused by palindromic SNPs.
- `indel_mismatch`: A mismatch was detected between pairs of alleles where at least one was an INDEL. INDEL alleles are not supported for strand flip assessment.
- `unresolved_mismatch`: A mismatch was detected between pairs of non-INDEL alleles that could not be resolved by an effect switch or flipping the PGS alleles.
- `missing_allele`: One of the four alleles is missing, making it impossible to assess the match.

Examples

```
# Example data demonstrating the following cases in each vector element:
# 1. no strand flips
# 2. effect allele switch
# 3. strand flip
# 4. effect allele switch AND strand flip
# 5. palindromic (ambiguous) alleles
# 6. unresolved mismatch
vcf.ref.allele <- c('A', 'A', 'A', 'A', 'A', 'A');
vcf.alt.allele <- c('G', 'G', 'G', 'G', 'T', 'G');
pgs.ref.allele <- c('A', 'G', 'T', 'C', 'T', 'A');
pgs.effect.allele <- c('G', 'A', 'C', 'T', 'A', 'C');
assess.pgs.vcf.allele.match(vcf.ref.allele, vcf.alt.allele, pgs.ref.allele, pgs.effect.allele);
```

`check.pgs.weight.columns` *Check PGS weight file columns*

Description

Check that a PGS weight file contains the required columns for PGS application with `apply.polygenic.score`.

Usage

```
check.pgs.weight.columns(pgs.weight.colnames, harmonized = TRUE)
```

Arguments

`pgs.weight.colnames`
: A character vector of column names.

`harmonized`
: A logical indicating whether the presence of harmonized columns should be checked.

Value

A logical indicating whether the file contains the required columns.

`combine.pgs.bed` *Combine PGS BED files*

Description

Merge overlapping PGS coordinates in multiple BED files.

Usage

```
combine.pgs.bed(
  pgs.bed.list,
  add.annotation.data = FALSE,
  annotation.column.index = 4,
  slop = 0
)
```

Arguments

`pgs.bed.list` A named list of data.frames containing PGS coordinates in BED format.

`add.annotation.data`
: A logical indicating whether an additional annotation data column should be added to the annotation column.

`annotation.column.index`
: An integer indicating the index of the column in the data frames in `pgs.bed.list` that should be added to the annotation column.

`slop` An integer indicating the number of base pairs to add to the BED interval on either side.

Value

A data.frame containing the merged PGS coordinates in BED format with an extra annotation column containing the name of the PGS and data from one additional column optionally selected by the user.

Examples

```
bed1 <- data.frame(
    chr = c(1, 2, 3),
    start = c(1, 2, 3),
    end = c(2, 3, 4),
    annotation = c('a', 'b', 'c')
    );
bed2 <- data.frame(
    chr = c(1, 2, 3),
    start = c(1, 20, 30),
    end = c(2, 21, 31),
    annotation = c('d', 'e', 'f')
    );
bed.input <- list(bed1 = bed1, bed2 = bed2);
combine.pgs.bed(bed.input);
```

---

combine.vcf.with.pgs *Combine VCF with PGS*

Description

Match PGS SNPs to corresponding VCF information by genomic coordinates or rsID using a merge operation.

Usage

```
combine.vcf.with.pgs(vcf.data, pgs.weight.data)
```

Arguments

`vcf.data` A data.frame containing VCF data. Required columns: `CHROM`, `POS`.

`pgs.weight.data` A data.frame containing PGS data. Required columns: `CHROM`, `POS`.

Value

A list containing a data.frame of merged VCF and PGS data and a data.frame of PGS SNPs missing from the VCF.

A primary merge is first performed on chromosome and base pair coordinates. For SNPs that could not be matched in the first mergs, a second merge is attempted by rsID if available. This action can account for short INDELs that can have coordinate mismatches between the PGS and VCF data. The merge is a left outer join: all PGS SNPs are kept as rows even if they are missing from the VCF, and all VCF SNPs that are not a component of the PGS are dropped. If no PGS SNPs are present in the VCF, the function will terminate with an error.

Examples

```
# Example VCF
vcf.path <- system.file(
    'extdata',
    'HG001_GIAB.vcf.gz',
    package = 'ApplyPolygenicScore',
    mustWork = TRUE
    );
vcf.import <- import.vcf(vcf.path);

# Example pgs weight file
pgs.weight.path <- system.file(
    'extdata',
    'PGS000662_hmPOS_GRCh38.txt.gz',
    package = 'ApplyPolygenicScore',
    mustWork = TRUE
    );
pgs.import <- import.pgs.weight.file(pgs.weight.path);

merge.data <- combine.vcf.with.pgs(
    vcf.data = vcf.import$dat,
    pgs.weight.data = pgs.import$pgs.weight.data
    );
```

`convert.allele.frequency.to.dosage` *Convert allele frequency to mean dosage*

Description

Convert a population allele frequency to a mean dosage for that allele.

Usage

```
convert.allele.frequency.to.dosage(allele.frequency)
```

Arguments

`allele.frequency`
A numeric vector of allele frequencies.

Value

A numeric vector of mean dosages for the allele frequencies.

Examples

```
allele.frequency <- seq(0.1, 0.9, 0.1);
convert.allele.frequency.to.dosage(allele.frequency);
```

`convert.alleles.to.pgs.dosage`
Convert alleles to dosage

Description

Convert genotype calls in the form of witten out alleles (e.g. 'A/T') to dosages (0, 1, 2) based on provided risk alleles from a PGS.

Usage

```
convert.alleles.to.pgs.dosage(called.alleles, risk.alleles)
```

Arguments

`called.alleles`	A vector of genotypes in allelic notation separated by a slash or pipe.
`risk.alleles`	A vector of risk alleles from a polygenic score corresponding to each genotype (by locus) in called.alleles.

Value

A vector of dosages corresponding to each genotype in called.alleles. Hemizygous genotypes (one allele e.g. 'A') are counted as 1.

Examples

```
called.alleles <- c('A/A', 'A/T', 'T/T');
risk.alleles <- c('T', 'T', 'T');
convert.alleles.to.pgs.dosage(called.alleles, risk.alleles);
```

convert.pgs.to.bed *Convert PGS data to BED format*

Description

Convert imported and formatted PGS compnent SNP coordinate data to BED format.

Usage

```
convert.pgs.to.bed(
  pgs.weight.data,
  chr.prefix = TRUE,
  numeric.sex.chr = FALSE,
  slop = 0
)
```

Arguments

pgs.weight.data	A data.frame containing SNP coordinate data with standardized CHROM and POS columns.
chr.prefix	A logical indicating whether the 'chr' prefix should be used when formatting chromosome name.
numeric.sex.chr	A logical indicating whether the sex chromosomes should be formatted numerically, as opposed to alphabetically.
slop	An integer indicating the number of base pairs to add to the BED interval on either side.

Value

A data.frame containing the PGS component SNP coordinate data in BED format and any other columns provided in pgs.weight.data.

Examples

```
pgs.weight.data <- data.frame(
    CHROM = c('chr1', 'chrX'),
    POS = c(10, 20)
    );
convert.pgs.to.bed(pgs.weight.data);

# Switch between different chromosome notations
convert.pgs.to.bed(pgs.weight.data, chr.prefix = FALSE, numeric.sex.chr = TRUE);

# Add slop to BED intervals
convert.pgs.to.bed(pgs.weight.data, slop = 10);
```

create.pgs.density.plot
Plot PGS Density

Description

Plot density curves of PGS data outputted by apply.polygenic.score. If phenotype columns are provided, multiple density curves are plotted for automatically detected categories for each categorical variable.

Usage

```
create.pgs.density.plot(
  pgs.data,
  phenotype.columns = NULL,
  output.dir = NULL,
  filename.prefix = NULL,
  file.extension = "png",
  tidy.titles = FALSE,
  width = 10,
  height = 10,
  xaxes.cex = 1.5,
  yaxes.cex = 1.5,
  titles.cex = 1.5,
  key.cex = 1,
  border.padding = 1
)
```

Arguments

pgs.data	data.frame PGS data as formatted by apply.polygenic.score(). Required columns are at least one of PGS, PGS.with.replaced.missing, or PGS.with.normalized.missing. This function is designed to work with the output of apply.polygenic.score().
phenotype.columns	character vector of phenotype columns in pgs.data to plot (optional)
output.dir	character directory to save output plots
filename.prefix	character prefix for output filenames
file.extension	character file extension for output plots
tidy.titles	logical whether to reformat PGS plot titles to remove periods
width	numeric width of output plot in inches
height	numeric height of output plot in inches
xaxes.cex	numeric size for all x-axis labels
yaxes.cex	numeric size for all y-axis labels

titles.cex	numeric size for all plot titles
key.cex	numeric size of color key legend
border.padding	numeric padding for plot borders

Value

If no output directory is provided, a multipanel lattice plot object is returned, otherwise a plot is written to the indicated path and NULL is returned.

Examples

```
set.seed(100);
pgs.data <- data.frame(
    PGS = rnorm(100, 0, 1)
    );
 temp.dir <- tempdir();

# Basic Plot
create.pgs.density.plot(
    pgs.data,
    output.dir = temp.dir,
    filename.prefix = 'basic-plot',
    width = 6,
    height = 6
    );

# Plot multiple PGS outputs
pgs.data$PGS.with.normalized.missing <- rnorm(100, 1, 1);
create.pgs.density.plot(pgs.data, output.dir = temp.dir);

# Plot phenotype categories

pgs.data$sex <- sample(c('male', 'female', 100, replace = TRUE));
create.pgs.density.plot(
    pgs.data,
    output.dir = temp.dir,
    filename.prefix = 'multiple-pgs',
    phenotype.columns = 'sex'
    );

# Plot multiple phenotypes

pgs.data$letters <- sample(letters[1:5], 100, replace = TRUE);
create.pgs.density.plot(
    pgs.data,
    output.dir = temp.dir,
    filename.prefix = 'multiple-phenotypes',
    phenotype.columns = c('sex', 'letters')
    );
```

create.pgs.rank.plot *Plot PGS Rank*

Description

Plot PGS percentile rank of each sample outputted by `apply.polygenic.score()` as a barplot, plot missing genotypes if any are present, plot corresponding decile and quartile markers as a heatmap, optionally plot phenotype covariates as color bars.

Usage

```
create.pgs.rank.plot(
  pgs.data,
  phenotype.columns = NULL,
  missing.genotype.style = "count",
  categorical.palette = NULL,
  binary.palette = NULL,
  output.dir = NULL,
  filename.prefix = NULL,
  file.extension = "png",
  width = 8,
  height = 8,
  xaxis.cex = 1.2,
  yaxis.cex = 1,
  titles.cex = 1.2,
  border.padding = 1
)
```

Arguments

`pgs.data` data.frame PGS data as formatted by `apply.polygenic.score()` Required columns: `Indiv`, `percentile`, `decile`, `quartile`, `n.missing.genotypes`, `percent.missing.genotypes`, and optionally user-defined percentiles and phenotype covariates. This function is designed to work with the output of the function apply.polygenic.score().

`phenotype.columns` character vector of column names in pgs.data containing phenotype covariates to plot as color bars. Default is `NULL`.

`missing.genotype.style` character style of missing genotype barplot. Default is "count". Options are "count" or "percent".

`categorical.palette` character vector of colors to use for categorical phenotype covariates. Default is `NULL` in which case the default palette is used, which contains 12 unique colors. If the number of unique categories exceeds the number of colors in the color palette, an error will be thrown.

binary.palette character vector of colors to use for binary and continuous phenotype covariates. Each color is contrasted with white to create a color ramp or binary categories. Default is NULL in which case the default palette is used, which contains 9 unique colors paired with white. If the number of binary and continuous phenotype covariates exceeds the number of colors in the color palette, an error will be thrown.

output.dir character directory path to write plot to file. Default is NULL in which case the plot is returned as lattice multipanel object.

filename.prefix character prefix for plot filename.

file.extension character file extension for plot file. Default is "png".

width numeric width of plot in inches.

height numeric height of plot in inches.

xaxis.cex numeric size of x-axis labels.

yaxis.cex numeric size of y-axis labels.

titles.cex numeric size of plot titles.

border.padding numeric padding around plot border.

Value

If no output directory is provided, a multipanel lattice plot object is returned, otherwise a plot is written to the indicated path and NULL is returned.

For clarity, certain plot aspects change when sample size exceeds 50:

- x-axis labels are no longer displayed
- missing (NA) values are not labeled with text in heatmaps but are color-coded with a legend

Colors for continuous and binary phenotypes are chosen from the binary color palettes in BoutrosLab.plotting.general::
Colors for categorical phenotypes are chosen by default from the qualitative color palette in BoutrosLab.plotting.general

Examples

```
set.seed(200);
percentiles <- get.pgs.percentiles(rnorm(200, 0, 1));
pgs.data <- data.frame(
    Indiv = paste0('sample', 1:200),
    percentile = percentiles$percentile,
    decile = percentiles$decile,
    quartile = percentiles$quartile,
    n.missing.genotypes = sample(1:10, 200, replace = TRUE),
    percent.missing.genotypes = sample(1:10, 200, replace = TRUE) / 100,
    continuous.pheno = rnorm(200, 1, 1),
    categorical.pheno = sample(letters[1:5], 200, replace = TRUE),
    binary.pheno = sample(c(0,1), 200, replace = TRUE)
    );

temp.dir <- tempdir();
```

```
create.pgs.rank.plot(
    pgs.data,
    phenotype.columns = c('continuous.pheno', 'categorical.pheno', 'binary.pheno'),
    missing.genotype.style = 'percent',
    output.dir = temp.dir,
    filename.prefix = 'example-rank-plot'
    );
```

---

create.pgs.with.continuous.phenotype.plot
Plot PGS Scatterplots

Description

Create scatterplots for PGS data outputed by `apply.polygenic.score()` with continuous phenotype variables

Usage

```
create.pgs.with.continuous.phenotype.plot(
  pgs.data,
  phenotype.columns,
  hexbin.threshold = 1000,
  hexbin.colour.scheme = NULL,
  hexbin.colourkey = TRUE,
  hexbin.colourcut = seq(0, 1, length = 11),
  hexbin.mincnt = 1,
  hexbin.maxcnt = NULL,
  hexbin.xbins = 30,
  hexbin.aspect = 1,
  output.dir = NULL,
  filename.prefix = NULL,
  file.extension = "png",
  tidy.titles = FALSE,
  compute.correlation = TRUE,
  corr.legend.corner = c(0, 1),
  corr.legend.cex = 1.5,
  include.origin = FALSE,
  width = 10,
  height = 10,
  xaxes.cex = 1.5,
  yaxes.cex = 1.5,
  titles.cex = 1.5,
  point.cex = 0.75,
  border.padding = 1
)
```

Arguments

`pgs.data` data.frame PGS data as formatted by `apply.polygenic.score()`. Required columns are at least one of PGS, PGS.with.replaced.missing, or PGS.with.normalized.missing, and at least one continuous phenotype column. This function is designed to work with the output of `apply.polygenic.score()`.

`phenotype.columns` character vector of continuous phenotype column names in pgs.data to plot

`hexbin.threshold` numeric threshold (exclusive) for cohort size at which to switch from scatterplot to hexbin plot.

`hexbin.colour.scheme` character vector of colors for hexbin plot bins. Default is `NULL` which uses gray/black.

`hexbin.colourkey` logical whether a legend should be drawn for a hexbinplot, defaults to `TRUE`.

`hexbin.colourcut` numeric vector of values covering [0, 1] that determine hexagon colour class boundaries and hexagon legend size boundaries. Alternatively, an integer (<= hexbin.maxcnt) specifying the number of equispaced colourcut values in [0,1].

`hexbin.mincnt` integer, minimum count for a hexagon to be plotted. Default is 1.

`hexbin.maxcnt` integer, maximum count for a hexagon to be plotted. Cells with more counts are not plotted. Default is `NULL`.

`hexbin.xbins` integer, number of bins in the x direction for hexbin plot. Default is 30.

`hexbin.aspect` numeric, aspect ratio of hexbin plot to control plot dimensions. Default is 1.

`output.dir` character directory to save output plots

`filename.prefix` character prefix for output filenames

`file.extension` character file extension for output plots

`tidy.titles` logical whether to reformat PGS plot titles to remove periods

`compute.correlation` logical whether to compute correlation between PGS and phenotype and display in plot

`corr.legend.corner` numeric vector indicating the corner of the correlation legend e.g. `c(0,1)` for top left

`corr.legend.cex` numeric cex for correlation legend

`include.origin` logical whether to include the origin (zero) in plot axes

`width` numeric width of output plot in inches

`height` numeric height of output plot in inches

`xaxes.cex` numeric size for x-axis labels

`yaxes.cex` numeric size for y-axis labels

`titles.cex` numeric size for plot titles

`point.cex` numeric size for plot points

`border.padding` numeric padding for plot borders

Value

If no output directory is provided, a multipanel lattice plot object is returned, otherwise a plot is written to the indicated path and `NULL` is returned. If no continuous phenotype variables are detected, a warning is issued and `NULL` is returned.

Examples

```
set.seed(100);

pgs.data <- data.frame(
    PGS = rnorm(100, 0, 1),
    continuous.phenotype = rnorm(100, 2, 1)
    );
 temp.dir <- tempdir();

# Basic Plot
create.pgs.with.continuous.phenotype.plot(
    pgs.data,
    output.dir = temp.dir,
    filename.prefix = 'basic-plot',
    phenotype.columns = 'continuous.phenotype',
    width = 6,
    height = 6
    );

# Plot multiple PGS outputs

pgs.data$PGS.with.normalized.missing <- rnorm(100, 1, 1);
create.pgs.with.continuous.phenotype.plot(
    pgs.data,
    output.dir = temp.dir,
    filename.prefix = 'multiple-pgs',
    phenotype.columns = 'continuous.phenotype'
    );


# Plot multiple phenotypes

pgs.data$continuous.phenotype2 <- rnorm(100, 10, 1);
create.pgs.with.continuous.phenotype.plot(
    pgs.data,
    output.dir = temp.dir,
    filename.prefix = 'multiple-phenotypes',
    phenotype.columns = c('continuous.phenotype', 'continuous.phenotype2')
    );
```

`flip.DNA.allele` *Flip DNA allele*

Description

Flip single base pair DNA alleles to their reverse complement. INDEL flipping is not supported.

Usage

```
flip.DNA.allele(alleles, return.indels.as.missing = FALSE)
```

Arguments

`alleles` A character vector of DNA alleles.

`return.indels.as.missing`
A logical value indicating whether to return NA for INDEL alleles. Default is `FALSE`.

Value

A character vector of flipped DNA alleles. INDEL alleles are returned as is unless `return.indels.as.missing` is `TRUE`.

Examples

```
alleles <- c('A', 'T', 'C', 'G', 'ATG', NA);
flip.DNA.allele(alleles);
```

`format.chromosome.notation`
Format chromosome names

Description

Format chromosome names according to user specifications.

Usage

```
## S3 method for class 'chromosome.notation'
format(chromosome, chr.prefix, numeric.sex.chr)
```

Arguments

`chromosome` A character vector of chromosome names.

`chr.prefix` A logical indicating whether the 'chr' prefix should be used when formatting chromosome name.

`numeric.sex.chr`
A logical indicating whether the sex chromosomes should be formatted numerically, as opposed to alphabetically.

Value

A character vector of chromosome names formatted according to user specifications.

Examples

```
numeric.chr <- c(1,2,23,24);
chr.with.prefix <- c('chr1', 'chr2', 'chrX', 'chrY');
format.chromosome.notation(numeric.chr, chr.prefix = TRUE, numeric.sex.chr = FALSE);
format.chromosome.notation(chr.with.prefix, chr.prefix = FALSE, numeric.sex.chr = TRUE);
```

---

get.pgs.percentiles *get.pgs.percentiles*

Description

Calculate percentiles and report decile and quartile ranks for a vector of polygenic scores

Usage

```
get.pgs.percentiles(pgs, n.percentiles = NULL)
```

Arguments

pgs	numeric vector of polygenic scores
n.percentiles	integer number of percentiles to calculate (optional)

Value

data frame with columns for percentile, decile, quartile, and optional n.percentiles

Examples

```
x <- rnorm(100);
get.pgs.percentiles(x, n.percentiles = 20);
```

`import.pgs.weight.file` *Import PGS weight file*

Description

Import a PGS weight file formatted according to PGS catalog guidelines, and prepare for PGS application with `apply.polygenic.score()`.

Usage

```
import.pgs.weight.file(pgs.weight.path, use.harmonized.data = TRUE)
```

Arguments

`pgs.weight.path`
: A character string indicating the path to the pgs weight file.

`use.harmonized.data`
: A logical indicating whether the file should be formatted to indicate harmonized data columns for use in future PGS application.

Value

A list containing the file metadata and the weight data.

Examples

```
# Example pgs weight file
pgs.weight.path <- system.file(
    'extdata',
    'PGS000662_hmPOS_GRCh38.txt.gz',
    package = 'ApplyPolygenicScore',
    mustWork = TRUE
    );
import.pgs.weight.file(pgs.weight.path);

# Note, harmonized data is used by default. To disable set `use.harmonized.data = FALSE`
import.pgs.weight.file(pgs.weight.path, use.harmonized.data = FALSE);
```

`import.vcf` *Import VCF file*

Description

A wrapper for the VCF import function in the vcfR package that formats VCF data for PGS application with `apply.polygenic.score()`.

Usage

```
import.vcf(vcf.path, info.fields = NULL, format.fields = NULL, verbose = FALSE)
```

Arguments

`vcf.path`	A character string indicating the path to the VCF file to be imported.
`info.fields`	A character vector indicating the INFO fields to be imported.
`format.fields`	A character vector indicating the FORMAT fields to be imported.
`verbose`	A logical indicating whether verbose output should be printed by vcfR.

Value

A list containing a tibble of VCF meta data and a tibble data.frame containing the parsed VCF data in long form.

Examples

```
# Example VCF
vcf <- system.file(
    'extdata',
    'HG001_GIAB.vcf.gz',
    package = 'ApplyPolygenicScore',
    mustWork = TRUE
    );
vcf.data <- import.vcf(vcf.path = vcf);
```

`parse.pgs.input.header` *Parse PGS input file header*

Description

Parse metadata from a PGS input file header.

Usage

```
parse.pgs.input.header(pgs.weight.path)
```

Arguments

`pgs.weight.path`
A character string indicating the path to the pgs weight file.

Value

A data frame containing the metadata from the file header.

Examples

```
# Example pgs weight file
pgs.weight.path <- system.file(
    'extdata',
    'PGS000662_hmPOS_GRCh38.txt.gz',
    package = 'ApplyPolygenicScore',
    mustWork = TRUE
    );
parse.pgs.input.header(pgs.weight.path);
```

---

run.pgs.regression   *Run linear and logistic regression on a polygenic score and a set of phenotypes*

Description

Phenotype data variables are automatically classified as continuous or binary and a simple linear regression or logistic regression, respectively, is run between the polygenic score and each phenotype. Categorical phenotypes with more than two category are ignored. If a binary variable is not formatted as a factor, it is converted to a factor using `as.factor()` defaults. For logistic regression, the first level is classified as "failure" and the second "success" by `glm()` defaults.

Usage

```
run.pgs.regression(pgs, phenotype.data)
```

Arguments

`pgs`	numeric vector of polygenic scores
`phenotype.data`	data.frame of phenotypes

Value

data frame with columns for phenotype, model, beta, se, p.value, r.squared, and AUC

Examples

```
set.seed(200);
pgs <- rnorm(200, 0, 1);
phenotype.data <- data.frame(
    continuous.pheno = rnorm(200, 1, 1),
    binary.pheno = sample(c(0, 1), 200, replace = TRUE)
    );
run.pgs.regression(pgs, phenotype.data);
```

---

write.apply.polygenic.score.output.to.file

Write apply.polygenic.score output to file

Description

A utility function that writes the two data frames outputted by apply.polygenic.score to two tab-delimited text files.

Usage

```
write.apply.polygenic.score.output.to.file(
  apply.polygenic.score.output,
  output.dir,
  file.prefix = NULL
)
```

Arguments

`apply.polygenic.score.output`
: list of two data frames: `pgs.output` and `regression.output`

`output.dir`
: character string of the path to write both output files

`file.prefix`
: character string of the file prefix to use for both output files

Index