Package: plmmr (via r-universe)

October 22, 2024

Title Penalized Linear Mixed Models for Correlated Data

Version 4.1.0.0

Date 2024-10-05

Description Fits penalized linear mixed models that correct for unobserved confounding factors. 'plmmr' infers and corrects for the presence of unobserved confounding effects such as population stratification and environmental heterogeneity. It then fits a linear model via penalized maximum likelihood. Originally designed for the multivariate analysis of single nucleotide polymorphisms (SNPs) measured in a genome-wide association study (GWAS), 'plmmr' eliminates the need for subpopulation-specific analyses and post-analysis p-value adjustments. Functions for the appropriate processing of 'PLINK' files are also supplied. For examples, see the package homepage. <`https://pbreheny.github.io/plmmr/`>.

License GPL-3

URL `https://pbreheny.github.io/plmmr/`, `https://github.com/pbreheny/plmmr/`

Depends bigalgebra, bigmemory, R (>= 4.4.0)

Imports biglasso (>= 1.6.0), data.table, glmnet, Matrix, ncvreg, parallel, utils

Suggests bigsnpr, bigstatsr, graphics, grDevices, knitr, MASS, rmarkdown, tinytest

LinkingTo BH, bigmemory, Rcpp, RcppArmadillo (>= 0.8.600)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

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

RemoteUrl https://github.com/pbreheny/plmmr

RemoteRef HEAD

RemoteSha 37adf53812b5b9a3069d4143ee278ad21aea2f59

Contents

`admix` *Admix: Semi-simulated SNP data*

Description

A dataset containing the 100 SNPs, a demographic variable representing race, and a simulated outcome

Usage

```
admix
```

Format

A list with 3 components

X SNP matrix (197 observations of 100 SNPs)

y vector of simulated (continuous) outcomes

race vector with racial group categorization: # 0 = African, 1 = African American, 2 = European, 3 = Japanese

Source

https://hastie.su.domains/CASI/

`coef.cv_plmm` *Coef method for "cv_plmm" class*

Description

Coef method for "cv_plmm" class

Usage

```
## S3 method for class 'cv_plmm'
coef(object, lambda, which = object$min, ...)
```

Arguments

`object`	An object of class "cv_plmm."
`lambda`	A numeric vector of lambda values.
`which`	Vector of lambda indices for which coefficients to return. Defaults to lambda index with minimum CVE.
`...`	Additional arguments (not used).

Value

Returns a named numeric vector. Values are the coefficients of the model at the specified value of either `lambda` or `which`. Names are the values of `lambda`.

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
cv_fit <- cv_plmm(design = admix_design, return_fit = TRUE)
head(coef(cv_fit))
```

coef.plmm *Coef method for "plmm" class*

Description

Coef method for "plmm" class

Usage

```
## S3 method for class 'plmm'
coef(object, lambda, which = 1:length(object$lambda), drop = TRUE, ...)
```

Arguments

`object`	An object of class "plmm."
`lambda`	A numeric vector of lambda values.
`which`	Vector of lambda indices for which coefficients to return.
`drop`	Logical.
`...`	Additional arguments.

Value

Either a numeric matrix (if model was fit on data stored in memory) or a sparse matrix (if model was fit on data stored filebacked). Rownames are feature names, columns are values of `lambda`.

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
fit <- plmm(design = admix_design)
coef(fit)[1:10, 41:45]
```

create_design *a function to create a design for PLMM modeling*

Description

a function to create a design for PLMM modeling

Usage

```
create_design(data_file = NULL, rds_dir = NULL, X = NULL, y = NULL, ...)
```

Arguments

`data_file`	For **filebacked data** (data from `process_plink()` or `process_delim()`), this is the filepath to the processed data. Defaults to NULL (this argument does not apply for in-memory data).
`rds_dir`	For **filebacked data**, this is the filepath to the directory/folder where you want the design to be saved. **Note**: do not include/append the name you want for the to-be-created file – the name is the argument `new_file`, passed to `create_design_filebacked()`. Defaults to NULL (this argument does not apply for in-memory data).
`X`	For **in-memory data (data in a matrix or data frame)**, this is the design matrix. Defaults to NULL (this argument does not apply for filebacked data).
`y`	For **in-memory data**, this is the numeric vector representing the outcome. Defaults to NULL (this argument does not apply for filebacked data). **Note**: it is the responsibility of the user to ensure that the rows in X and the corresponding elements of y have the same row order, i.e., observations must be in the same order in both the design matrix and in the outcome vector.
`...`	Additional arguments to pass to `create_design_filebacked()` or `create_design_in_memory()`. See the documentation for those helper functions for details.

Details

This function is a wrapper for the other `create_design...()` inner functions; all arguments included here are passed along to the `create_design...()` inner function that matches the type of the data being supplied. Note which arguments are optional and which ones are not.

Additional arguments for **all filebacked** data:

- **new_file** User-specified filename (*without .bk/.rds extension*) for the to-be-created .rds/.bk files. Must be different from any existing .rds/.bk files in the same folder.
- **feature_id** Optional: A string specifying the column in the data X (the feature data) with the row IDs (e.g., identifiers for each row/sample/participant/, etc.). No duplicates allowed. - for PLINK data: a string specifying an ID column of the PLINK `.fam` file. Options are "IID" (default) and "FID" - for all other filebacked data: a character vector of unique identifiers (IDs) for each row of the feature data (i.e., the data processed with `process_delim()`) - if left NULL (default), X is assumed to have the same row-order as add_outcome. **Note**: if this assumption is made in error, calculations downstream will be incorrect. Pay close attention here.
- **add_outcome** A data frame or matrix with two columns: and ID column and a column with the outcome value (to be used as 'y' in the final design). IDs must be characters, outcome must be numeric.
- **outcome_id** A string specifying the name of the ID column in 'add_outcome'
- **outcome_col** A string specifying the name of the phenotype column in 'add_outcome'
- **na_outcome_vals** Optional: a vector of numeric values used to code NA values in the outcome. Defaults to `c(-9, NA_integer)` (the -9 matches PLINK conventions).
- **overwrite** Optional: logical - should existing .rds files be overwritten? Defaults to FALSE.
- **logfile** Optional: name of the '.log' file to be written – **Note:** do not append a `.log` to the filename; this is done automatically.

- **quiet** Optional: logical - should messages to be printed to the console be silenced? Defaults to FALSE

Additional arguments specific to **PLINK** data:

- **add_predictor** Optional (for PLINK data only): a matrix or data frame to be used for adding additional **unpenalized** covariates/predictors/features from an external file (i.e., not a PLINK file). This matrix must have one column that is an ID column; all other columns aside the ID will be used as covariates in the design matrix. Columns must be named.
- **predictor_id** Optional (for PLINK data only): A string specifying the name of the column in 'add_predictor' with sample IDs. Required if 'add_predictor' is supplied. The names will be used to subset and align this external covariate with the supplied PLINK data.

Additional arguments specific to **delimited file** data:

- **unpen** Optional: an character vector with the names of columns to mark as unpenalized (i.e., these features would always be included in a model). **Note**: if you choose to use this option, your delimited file **must** have column names.

Additional arguments for **in-memory** data:

- **unpen** Optional: an character vector with the names of columns to mark as unpenalized (i.e., these features would always be included in a model). **Note**: if you choose to use this option, X must have column names.

Value

A filepath to an object of class `plmm_design`, which is a named list with the design matrix, outcome, penalty factor vector, and other details needed for fitting a model. This list is stored as an .rds file for filebacked data, so in the filebacked case a string with the path to that file is returned. For in-memory data, the list itself is returned.

Examples

```
## Example 1: matrix data in-memory ##
admix_design <- create_design(X = admix$X, y = admix$y, unpen = "Snp1")

## Example 2: delimited data ##
# process delimited data
temp_dir <- tempdir()
colon_dat <- process_delim(data_file = "colon2.txt",
 data_dir = find_example_data(parent = TRUE), overwrite = TRUE,
 rds_dir = temp_dir, rds_prefix = "processed_colon2", sep = "\t", header = TRUE)

# prepare outcome data
colon_outcome <- read.delim(find_example_data(path = "colon2_outcome.txt"))

# create a design
colon_design <- create_design(data_file = colon_dat, rds_dir = temp_dir, new_file = "std_colon2",
add_outcome = colon_outcome, outcome_id = "ID", outcome_col = "y", unpen = "sex",
overwrite = TRUE, logfile = "test.log")
```

```
# look at the results
colon_rds <- readRDS(colon_design)
str(colon_rds)

## Example 3: PLINK data ##

# process PLINK data
temp_dir <- tempdir()
unzip_example_data(outdir = temp_dir)

plink_data <- process_plink(data_dir = temp_dir,
  data_prefix = "penncath_lite",
  rds_dir = temp_dir,
  rds_prefix = "imputed_penncath_lite",
  # imputing the mode to address missing values
  impute_method = "mode",
  # overwrite existing files in temp_dir
  # (you can turn this feature off if you need to)
  overwrite = TRUE,
  # turning off parallelization - leaving this on causes problems knitting this vignette
  parallel = FALSE)

# get outcome data
penncath_pheno <- read.csv(find_example_data(path = 'penncath_clinical.csv'))

outcome <- data.frame(FamID = as.character(penncath_pheno$FamID),
                  CAD = penncath_pheno$CAD)

unpen_predictors <- data.frame(FamID = as.character(penncath_pheno$FamID),
                               sex = penncath_pheno$sex,
                               age = penncath_pheno$age)


# create design where sex and age are always included in the model
pen_design <- create_design(data_file = plink_data,
  feature_id = "FID",
  rds_dir = temp_dir,
  new_file = "std_penncath_lite",
  add_outcome = outcome,
  outcome_id = "FamID",
  outcome_col = "CAD",
  add_predictor = unpen_predictors,
  predictor_id = "FamID",
  logfile = "design",
  # again, overwrite if needed; use with caution
  overwrite = TRUE)

# examine the design - notice the components of this object
pen_design_rds <- readRDS(pen_design)
```

cv_plmm *Cross-validation for plmm*

Description

Performs k-fold cross validation for lasso-, MCP-, or SCAD-penalized linear mixed models over a grid of values for the regularization parameter `lambda`.

Usage

```
cv_plmm(
  design,
  y = NULL,
  K = NULL,
  diag_K = NULL,
  eta_star = NULL,
  penalty = "lasso",
  type = "blup",
  gamma,
  alpha = 1,
  lambda_min,
  nlambda = 100,
  lambda,
  eps = 1e-04,
  max_iter = 10000,
  convex = TRUE,
  dfmax = NULL,
  warn = TRUE,
  init = NULL,
  cluster,
  nfolds = 5,
  seed,
  fold = NULL,
  returnY = FALSE,
  returnBiasDetails = FALSE,
  trace = FALSE,
  save_rds = NULL,
  save_fold_res = FALSE,
  return_fit = TRUE,
  compact_save = FALSE,
  ...
)
```

Arguments

`design`	The first argument must be one of three things: (1) `plmm_design` object (as created by `create_design()`) (2) a string with the file path to a design object

	(the file path must end in '.rds') (3) a `matrix` or `data.frame` object representing the design matrix of interest
`y`	Optional: In the case where `design` is a `matrix` or `data.frame`, the user must also supply a numeric outcome vector as the `y` argument. In this case, `design` and `y` will be passed internally to `create_design(X = design, y = y)`.
`K`	Similarity matrix used to rotate the data. This should either be (1) a known matrix that reflects the covariance of y, (2) an estimate (Default is $\frac{1}{p}(XX^T)$), or (3) a list with components 'd' and 'u', as returned by choose_k().
`diag_K`	Logical: should K be a diagonal matrix? This would reflect observations that are unrelated, or that can be treated as unrelated. Defaults to FALSE. Note: plmm() does not check to see if a matrix is diagonal. If you want to use a diagonal K matrix, you must set diag_K = TRUE.
`eta_star`	Optional argument to input a specific eta term rather than estimate it from the data. If K is a known covariance matrix that is full rank, this should be 1.
`penalty`	The penalty to be applied to the model. Either "lasso" (the default), "SCAD", or "MCP".
`type`	A character argument indicating what should be returned from predict.plmm(). If type == 'lp', predictions are based on the linear predictor, X beta. If type == 'blup', predictions are based on the sum of the linear predictor and the estimated random effect (BLUP). Defaults to 'blup', as this has shown to be a superior prediction method in many applications.
`gamma`	The tuning parameter of the MCP/SCAD penalty (see details). Default is 3 for MCP and 3.7 for SCAD.
`alpha`	Tuning parameter for the Mnet estimator which controls the relative contributions from the MCP/SCAD penalty and the ridge, or L2 penalty. alpha=1 is equivalent to MCP/SCAD penalty, while alpha=0 would be equivalent to ridge regression. However, alpha=0 is not supported; alpha may be arbitrarily small, but not exactly 0.
`lambda_min`	The smallest value for lambda, as a fraction of lambda.max. Default is .001 if the number of observations is larger than the number of covariates and .05 otherwise.
`nlambda`	Length of the sequence of lambda. Default is 100.
`lambda`	A user-specified sequence of lambda values. By default, a sequence of values of length nlambda is computed, equally spaced on the log scale.
`eps`	Convergence threshold. The algorithm iterates until the RMSD for the change in linear predictors for each coefficient is less than eps. Default is `1e-4`.
`max_iter`	Maximum number of iterations (total across entire path). Default is 10000.
`convex`	(future idea; not yet incorporated) Calculate index for which objective function ceases to be locally convex? Default is TRUE.
`dfmax`	(future idea; not yet incorporated) Upper bound for the number of nonzero coefficients. Default is no upper bound. However, for large data sets, computational burden may be heavy for models with a large number of nonzero coefficients.
`warn`	Return warning messages for failures to converge and model saturation? Default is TRUE.

`init`	Initial values for coefficients. Default is 0 for all columns of X.
`cluster`	cv_plmm() can be run in parallel across a cluster using the parallel package. The cluster must be set up in advance using parallel::makeCluster(). The cluster must then be passed to cv_plmm().
`nfolds`	The number of cross-validation folds. Default is 5.
`seed`	You may set the seed of the random number generator in order to obtain reproducible results.
`fold`	Which fold each observation belongs to. By default, the observations are randomly assigned.
`returnY`	Should cv_plmm() return the linear predictors from the cross-validation folds? Default is FALSE; if TRUE, this will return a matrix in which the element for row i, column j is the fitted value for observation i from the fold in which observation i was excluded from the fit, at the jth value of lambda.
`returnBiasDetails`	Logical: should the cross-validation bias (numeric value) and loss (n x p matrix) be returned? Defaults to FALSE.
`trace`	If set to TRUE, inform the user of progress by announcing the beginning of each CV fold. Default is FALSE.
`save_rds`	Optional: if a filepath and name *without* the '.rds' suffix is specified (e.g., `save_rds = "~/dir/my_results"`), then the model results are saved to the provided location (e.g., "~/dir/my_results.rds"). Defaults to NULL, which does not save the result.
`save_fold_res`	Optional: a logical value indicating whether the results (loss and predicted values) from each CV fold should be saved? If TRUE, then two '.rds' files will be saved ('loss' and 'yhat') will be created in the same directory as 'save_rds'. Both files will be updated after each fold is done. Defaults to FALSE.
`return_fit`	Optional: a logical value indicating whether the fitted model should be returned as a `plmm` object in the current (assumed interactive) session. Defaults to TRUE.
`compact_save`	Optional: if TRUE, three separate .rds files will saved: one with the 'beta_vals', one with 'K', and one with everything else (see below). Defaults to FALSE. **Note**: you must specify `save_rds` for this argument to be called.
`...`	Additional arguments to `plmm_fit`

Value

a list with 12 items:

- type: the type of prediction used ('lp' or 'blup')
- cve: numeric vector with the cross validation error (CVE) at each value of `lambda`
- cvse: numeric vector with the estimated standard error associated with each value of for `cve`
- fold: numeric `n` length vector of integers indicating the fold to which each observation was assigned
- lambda: numeric vector of `lambda` values
- fit: the overall fit of the object, including all predictors; this is a list as returned by `plmm()`

- min: The index corresponding to the value of lambda that minimizes cve
- lambda_min: The lambda value at which cve is minmized
- min1se: The index corresponding to the value of lambda within standard error of that which minimizes cve
- lambda1se: largest value of lambda such that error is within 1 standard error of the minimum.
- null.dev: numeric value representing the deviance for the intercept-only model. If you have supplied your own lambda sequence, this quantity may not be meaningful.
- estimated_Sigma: an n x n matrix representing the estimated covariance matrix.

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
cv_fit <- cv_plmm(design = admix_design, return_fit = TRUE)
print(summary(cv_fit))
plot(cv_fit)

# Note: for examples with filebacked data, see the filebacking vignette
# https://pbreheny.github.io/plmmr/articles/filebacking.html
```

`find_example_data` *A function to help with accessing example PLINK files*

Description

A function to help with accessing example PLINK files

Usage

```
find_example_data(path, parent = FALSE)
```

Arguments

path	Argument (string) specifying a path (filename) for an external data file in extdata/
parent	If path=TRUE and the user wants the name of the parent directory where that file is located, set parent=TRUE. Defaults to FALSE.

Value

If path=NULL, a character vector of file names is returned. If path is given, then a character string with the full file path

Examples

```
find_example_data(parent = TRUE)
```

`lasso` *helper function to implement lasso penalty*

Description

helper function to implement lasso penalty

Usage

```
lasso(z, l1, l2, v)
```

Arguments

z	solution over active set at each feature
l1	upper bound
l2	lower bound
v	the 'xtx' term

Value

numeric vector of the lasso-penalized coefficient estimates within the given bounds

`plmm` *Fit a linear mixed model via non-convex penalized maximum likelihood.*

Description

Fit a linear mixed model via non-convex penalized maximum likelihood.

Usage

```
plmm(
  design,
  y = NULL,
  K = NULL,
  diag_K = NULL,
  eta_star = NULL,
  penalty = "lasso",
  init = NULL,
  gamma,
  alpha = 1,
  dfmax = NULL,
  lambda_min,
  nlambda = 100,
  lambda,
  eps = 1e-04,
  max_iter = 10000,
  convex = TRUE,
  warn = TRUE,
  trace = FALSE,
  save_rds = NULL,
  compact_save = FALSE,
  return_fit = NULL,
  ...
)
```

Arguments

`design`	The first argument must be one of three things: (1) `plmm_design` object (as created by `create_design()`) (2) a string with the file path to a design object (the file path must end in '.rds') (3) a `matrix` or `data.frame` object representing the design matrix of interest
`y`	Optional: In the case where `design` is a `matrix` or `data.frame`, the user must also supply a numeric outcome vector as the `y` argument. In this case, `design` and `y` will be passed internally to `create_design(X = design, y = y)`.
`K`	Similarity matrix used to rotate the data. This should either be: (1) a known matrix that reflects the covariance of y, (2) an estimate (Default is $\frac{1}{p}(XX^T)$), or (3) a list with components 'd' and 'U', as returned by a previous `plmm()` model fit on the same data.
`diag_K`	Logical: should K be a diagonal matrix? This would reflect observations that are unrelated, or that can be treated as unrelated. Defaults to FALSE. Note: plmm() does not check to see if a matrix is diagonal. If you want to use a diagonal K matrix, you must set diag_K = TRUE.
`eta_star`	Optional argument to input a specific eta term rather than estimate it from the data. If K is a known covariance matrix that is full rank, this should be 1.
`penalty`	The penalty to be applied to the model. Either "lasso" (the default), "SCAD", or "MCP".
`init`	Initial values for coefficients. Default is 0 for all columns of X.
`gamma`	The tuning parameter of the MCP/SCAD penalty (see details). Default is 3 for MCP and 3.7 for SCAD.
`alpha`	Tuning parameter for the Mnet estimator which controls the relative contributions from the MCP/SCAD penalty and the ridge, or L2 penalty. alpha=1 is equivalent to MCP/SCAD penalty, while alpha=0 would be equivalent to ridge regression. However, alpha=0 is not supported; alpha may be arbitrarily small, but not exactly 0.
`dfmax`	(**Future idea; not yet incorporated**): Upper bound for the number of nonzero coefficients. Default is no upper bound. However, for large data sets, computational burden may be heavy for models with a large number of nonzero coefficients.

`lambda_min`	The smallest value for lambda, as a fraction of lambda.max. Default is .001 if the number of observations is larger than the number of covariates and .05 otherwise.
`nlambda`	Length of the sequence of lambda. Default is 100.
`lambda`	A user-specified sequence of lambda values. By default, a sequence of values of length nlambda is computed, equally spaced on the log scale.
`eps`	Convergence threshold. The algorithm iterates until the RMSD for the change in linear predictors for each coefficient is less than eps. Default is `1e-4`.
`max_iter`	Maximum number of iterations (total across entire path). Default is 10000.
`convex`	(**Future idea; not yet incorporated**): Calculate index for which objective function ceases to be locally convex? Default is TRUE.
`warn`	Return warning messages for failures to converge and model saturation? Default is TRUE.
`trace`	If set to TRUE, inform the user of progress by announcing the beginning of each step of the modeling process. Default is FALSE.
`save_rds`	Optional: if a filepath and name *without* the '.rds' suffix is specified (e.g., `save_rds = "~/dir/my_results"`), then the model results are saved to the provided location (e.g., "~/dir/my_results.rds"). Defaults to NULL, which does not save the result.
`compact_save`	Optional: if TRUE, three separate .rds files will saved: one with the 'beta_vals', one with 'K', and one with everything else (see below). Defaults to FALSE. **Note**: you must specify `save_rds` for this argument to be called.
`return_fit`	Optional: a logical value indicating whether the fitted model should be returned as a `plmm` object in the current (assumed interactive) session. Defaults to TRUE for in-memory data, and defaults to FALSE for filebacked data.
`...`	Additional optional arguments to `plmm_checks()`

Value

A list which includes:

- beta_vals: the matrix of estimated coefficients on the original scale. Rows are predictors, columns are values of `lambda`
- rotated_scale_beta_vals: the matrix of estimated coefficients on the ~rotated~ scale. This is the scale on which the model was fit.
- lambda: a numeric vector of the lasso tuning parameter values used in model fitting.
- eta: a number (double) between 0 and 1 representing the estimated proportion of the variance in the outcome attributable to population/correlation structure
- linear_predictors: the matrix resulting from the product of `stdrot_X` and the estimated coefficients on the ~rotated~ scale.
- penalty: character string indicating the penalty with which the model was fit (e.g., 'MCP')
- gamma: numeric value indicating the tuning parameter used for the SCAD or lasso penalties was used. Not relevant for lasso models.
- alpha: numeric value indicating the elastic net tuning parameter.

- loss: vector with the numeric values of the loss at each value of `lambda` (calculated on the ~rotated~ scale)
- penalty_factor: vector of indicators corresponding to each predictor, where 1 = predictor was penalized.
- ns_idx: vector with the indices of predictors which were non-singular features (i.e., features which had variation).
- iter: numeric vector with the number of iterations needed in model fitting for each value of `lambda`
- converged: vector of logical values indicating whether the model fitting converged at each value of `lambda`
- K: a list with 2 elements, `s` and `U` —
  - s: a vector of the eigenvalues of the relatedness matrix; see `relatedness_mat()` for details.
  - U: a matrix of the eigenvectors of the relatedness matrix

Examples

```
# using admix data
admix_design <- create_design(X = admix$X, y = admix$y)
fit_admix1 <- plmm(design = admix_design)
s1 <- summary(fit_admix1, idx = 50)
print(s1)
plot(fit_admix1)

# Note: for examples with large data that are too big to fit in memory,
# see the article "PLINK files/file-backed matrices" on our website
# https://pbreheny.github.io/plmmr/articles/filebacking.html
```

`plmm_loss` *Loss method for "plmm" class*

Description

Loss method for "plmm" class

Usage

```
plmm_loss(y, yhat)
```

Arguments

`y`	Observed outcomes (response) vector
`yhat`	Predicted outcomes (response) vector

Value

A numeric vector of the squared-error loss values for the given observed and predicted outcomes

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
fit <- plmm(design = admix_design, K = relatedness_mat(admix$X))
yhat <- predict(object = fit, newX = admix$X, type = 'lp', lambda = 0.05)
head(plmm_loss(yhat = yhat, y = admix$y))
```

---

plot.cv_plmm *Plot method for cv_plmm class*

Description

Plot method for cv_plmm class

Usage

```
## S3 method for class 'cv_plmm'
plot(
  x,
  log.l = TRUE,
  type = c("cve", "rsq", "scale", "snr", "all"),
  selected = TRUE,
  vertical.line = TRUE,
  col = "red",
  ...
)
```

Arguments

`x`	An object of class cv_plmm
`log.l`	Logical to indicate the plot should be returned on the natural log scale. Defaults to `log.l = FALSE`.
`type`	Type of plot to return. Defaults to "cve."
`selected`	Logical to indicate which variables should be plotted. Defaults to TRUE.
`vertical.line`	Logical to indicate whether vertical line should be plotted at the minimum/maximum value. Defaults to TRUE.
`col`	Color for vertical line, if plotted. Defaults to "red."
`...`	Additional arguments.

Value

Nothing is returned; instead, a plot is drawn representing the relationship between the tuning parameter 'lambda' value (x-axis) and the cross validation error (y-axis).

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
cvfit <- cv_plmm(design = admix_design)
plot(cvfit)
```

`plot.plmm` *Plot method for plmm class*

Description

Plot method for plmm class

Usage

```
## S3 method for class 'plmm'
plot(x, alpha = 1, log.l = FALSE, shade = TRUE, col, ...)
```

Arguments

x	An object of class `plmm`
alpha	Tuning parameter for the Mnet estimator which controls the relative contributions from the MCP/SCAD penalty and the ridge, or L2 penalty. `alpha=1` is equivalent to MCP/SCAD penalty, while `alpha=0` would be equivalent to ridge regression. However, `alpha=0` is not supported; alpha may be arbitrarily small, but not exactly 0.
log.l	Logical to indicate the plot should be returned on the natural log scale. Defaults to `log.l = FALSE`.
shade	Logical to indicate whether a local nonconvex region should be shaded. Defaults to TRUE.
col	Vector of colors for coefficient lines.
...	Additional arguments.

Value

Nothing is returned; instead, a plot of the coefficient paths is drawn at each value of lambda (one 'path' for each coefficient).

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
fit <- plmm(design = admix_design)
plot(fit)
plot(fit, log.l = TRUE)
```

`predict.plmm` *Predict method for plmm class*

Description

Predict method for plmm class

Usage

```
## S3 method for class 'plmm'
predict(
  object,
  newX,
  type = c("lp", "coefficients", "vars", "nvars", "blup"),
  lambda,
  idx = 1:length(object$lambda),
  X,
  y,
  ...
)
```

Arguments

`object`	An object of class `plmm`.
`newX`	Matrix of values at which predictions are to be made (not used for `type="coefficients"` or for some of the `type` settings in `predict`). This can be either a FBM object or a 'matrix' object. **Note**: Columns of this argument must be named!
`type`	A character argument indicating what type of prediction should be returned. Options are "lp," "coefficients," "vars," "nvars," and "blup." See details.
`lambda`	A numeric vector of regularization parameter `lambda` values at which predictions are requested.
`idx`	Vector of indices of the penalty parameter `lambda` at which predictions are required. By default, all indices are returned.
`X`	Original design matrix (not including intercept column) from object. Required only if `type == 'blup'` and object is too large to be returned in `plmm` object.
`y`	Original continuous outcome vector from object. Required only if `type == 'blup'`.
`...`	Additional optional arguments

Details

Define beta-hat as the coefficients estimated at the value of lambda that minimizes cross-validation error (CVE). Then options for `type` are as follows:

- ’response’ (default): uses the product of newX and beta-hat to predict new values of the outcome. This does not incorporate the correlation structure of the data. For the stats folks out there, this is simply the linear predictor.
- ’blup’ (acronym for Best Linear Unbiased Predictor): adds to the ’response’ a value that represents the esetimated random effect. This addition is a way of incorporating the estimated correlation structure of data into our prediction of the outcome.
- ’coefficients’: returns the estimated beta-hat
- ’vars’: returns the *indices* of variables (e.g., SNPs) with nonzero coefficients at each value of lambda. EXCLUDES intercept.
- ’nvars’: returns the *number* of variables (e.g., SNPs) with nonzero coefficients at each value of lambda. EXCLUDES intercept.

Value

Depends on the `type` - see Details

Examples

```
set.seed(123)
train_idx <- sample(1:nrow(admix$X), 100)
# Note: ^ shuffling is important here! Keeps test and train groups comparable.
train <- list(X = admix$X[train_idx,], y = admix$y[train_idx])
train_design <- create_design(X = train$X, y = train$y)

test <- list(X = admix$X[-train_idx,], y = admix$y[-train_idx])
fit <- plmm(design = train_design)

# make predictions for all lambda values
 pred1 <- predict(object = fit, newX = test$X, type = "lp")

# look at mean squared prediction error
mspe <- apply(pred1, 2, function(c){crossprod(test$y - c)/length(c)})
min(mspe)

# compare the MSPE of our model to a null model, for reference
# null model = intercept only -> y_hat is always mean(y)
crossprod(mean(test$y) - test$y)/length(test$y)
```

`print.summary.cv_plmm` *Print method for summary.cv_plmm objects*

Description

Print method for summary.cv_plmm objects

Usage

```
## S3 method for class 'summary.cv_plmm'
print(x, digits, ...)
```

Arguments

`x`	An object of class `summary.cv_plmm`
`digits`	The number of digits to use in formatting output
`...`	Not used

Value

Nothing is returned; instead, a message is printed to the console summarizing the results of the cross-validated model fit.

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
cv_fit <- cv_plmm(design = admix_design)
print(summary(cv_fit))
```

`print.summary.plmm` *A function to print the summary of a* `plmm` *model*

Description

A function to print the summary of a `plmm` model

Usage

```
## S3 method for class 'summary.plmm'
print(x, ...)
```

Arguments

`x`	A `summary.plmm` object
`...`	Not used

Value

Nothing is returned; instead, a message is printed to the console summarizing the results of the model fit.

Examples

```
lam <- rev(seq(0.01, 1, length.out=20)) |> round(2) # for sake of example
admix_design <- create_design(X = admix$X, y = admix$y)
fit <- plmm(design = admix_design, lambda = lam)
fit2 <- plmm(design = admix_design, penalty = "SCAD", lambda = lam)
print(summary(fit, idx = 18))
print(summary(fit2, idx = 18))
```

process_delim *A function to read in large data files as an FBM*

Description

A function to read in large data files as an FBM

Usage

```
process_delim(
  data_dir,
  data_file,
  feature_id,
  rds_dir = data_dir,
  rds_prefix,
  logfile = NULL,
  overwrite = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

`data_dir`	The directory to the file.
`data_file`	The file to be read in, without the filepath. This should be a file of numeric values. Example: use `data_file = "myfile.txt"`, not `data_file = "~/mydirectory/myfile.txt"` Note: if your file has headers/column names, set 'header = TRUE' – this will be passed into `bigmemory::read.big.matrix()`.
`feature_id`	A string specifying the column in the data X (the feature data) with the row IDs (e.g., identifiers for each row/sample/participant/, etc.). No duplicates allowed.
`rds_dir`	The directory where the user wants to create the '.rds' and '.bk' files Defaults to `data_dir`
`rds_prefix`	String specifying the user's preferred filename for the to-be-created .rds file (will be create insie `rds_dir` folder) Note: 'rds_prefix' cannot be the same as 'data_prefix'
`logfile`	Optional: the name (character string) of the prefix of the logfile to be written. Defaults to 'process_delim', i.e. you will get 'process_delim.log' as the outfile.

`overwrite`	Optional: the name (character string) of the prefix of the logfile to be written. Defaults to 'process_plink', i.e. you will get 'process_plink.log' as the outfile. **Note**: If there are multiple `.rds` files with names that start with "std_prefix_...", **this will error out**. To protect users from accidentally deleting files with saved results, only one `.rds` file can be removed with this option.
`quiet`	Logical: should the messages printed to the console be silenced? Defaults to FALSE.
`...`	Optional: other arguments to be passed to `bigmemory::read.big.matrix()`. Note: 'sep' is an option to pass here, as is 'header'.

Value

The file path to the newly created '.rds' file

Examples

```
temp_dir <- tempdir()
colon_dat <- process_delim(data_file = "colon2.txt",
 data_dir = find_example_data(parent = TRUE), overwrite = TRUE,
 rds_dir = temp_dir, rds_prefix = "processed_colon2", sep = "\t", header = TRUE)

colon2 <- readRDS(colon_dat)
str(colon2)
```

`process_plink` *Preprocess PLINK files using the* `bigsnpr` *package*

Description

Preprocess PLINK files using the `bigsnpr` package

Usage

```
process_plink(
  data_dir,
  data_prefix,
  rds_dir = data_dir,
  rds_prefix,
  logfile = NULL,
  impute = TRUE,
  impute_method = "mode",
  id_var = "IID",
  parallel = TRUE,
  quiet = FALSE,
  overwrite = FALSE,
  ...
)
```

Arguments

`data_dir`	The path to the bed/bim/fam data files, *without* a trailing "/" (e.g., use `data_dir = '~/my_dir'`, **not** `data_dir = '~/my_dir/'`)
`data_prefix`	The prefix (as a character string) of the bed/fam data files (e.g., `data_prefix = 'mydata'`)
`rds_dir`	The path to the directory in which you want to create the new '.rds' and '.bk' files. Defaults to `data_dir`
`rds_prefix`	String specifying the user's preferred filename for the to-be-created .rds file (will be create insie `rds_dir` folder) Note: 'rds_prefix' cannot be the same as 'data_prefix'
`logfile`	Optional: the name (character string) of the prefix of the logfile to be written in 'rds_dir'. Default to NULL (no log file written). Note: if you supply a file path in this argument, it will error out with a "file not found" error. Only supply the string; e.g., if you want my_log.log, supply 'my_log', the my_log.log file will appear in rds_dir.
`impute`	Logical: should data be imputed? Default to TRUE.
`impute_method`	If 'impute' = TRUE, this argument will specify the kind of imputation desired. Options are: * mode (default): Imputes the most frequent call. See `bigsnpr::snp_fastImputeSimple()` for details. * random: Imputes sampling according to allele frequencies. * mean0: Imputes the rounded mean. * mean2: Imputes the mean rounded to 2 decimal places. * xgboost: Imputes using an algorithm based on local XGBoost models. See `bigsnpr::snp_fastImpute()` for details. Note: this can take several minutes, even for a relatively small data set.
`id_var`	String specifying which column of the PLINK `.fam` file has the unique sample identifiers. Options are "IID" (default) and "FID"
`parallel`	Logical: should the computations within this function be run in parallel? Defaults to TRUE. See `count_cores()` and `?bigparallelr::assert_cores` for more details. In particular, the user should be aware that too much parallelization can make computations *slower.*
`quiet`	Logical: should messages to be printed to the console be silenced? Defaults to FALSE
`overwrite`	Logical: if existing `.bk`/`.rds` files exist for the specified directory/prefix, should these be overwritten? Defaults to FALSE. Set to TRUE if you want to change the imputation method you're using, etc. **Note**: If there are multiple `.rds` files with names that start with "std_prefix_...", **this will error out**. To protect users from accidentally deleting files with saved results, only one `.rds` file can be removed with this option.
`...`	Optional: additional arguments to `bigsnpr::snp_fastImpute()` (relevant only if impute_method = "xgboost")

Details

Three files are created in the location specified by `rds_dir`:

- 'rds_prefix.rds': This is a list with three items: (1) `X`: the filebacked `bigmemory::big.matrix` object pointing to the imputed genotype data. This matrix has type 'double', which is important for downstream operations in `create_design()` (2) `map`: a data.frame with the PLINK 'bim' data (i.e., the variant information) (3) `fam`: a data.frame with the PLINK 'fam' data (i.e., the pedigree information)
- 'prefix.bk': This is the backingfile that stores the numeric data of the genotype matrix
- 'rds_prefix.desc'" This is the description file, as needed by the

Note that `process_plink()` need only be run once for a given set of PLINK files; in subsequent data analysis/scripts, `get_data()` will access the '.rds' file.

For an example, see vignette on processing PLINK files

Value

The filepath to the '.rds' object created; see details for explanation.

`relatedness_mat` *Calculate a relatedness matrix*

Description

Given a matrix of genotypes, this function estimates the genetic relatedness matrix (GRM, also known as the RRM, see Hayes et al. 2009, doi:10.1017/S0016672308009981) among the subjects: XX'/p, where X is standardized.

Usage

```
relatedness_mat(X, std = TRUE, fbm = FALSE, ns = NULL, ...)
```

Arguments

`X`	An n x p numeric matrix of genotypes (from *fully-imputed* data). Note: This matrix should *not* include non-genetic features.
`std`	Logical: should X be standardized? If you set this to FALSE (which can only be done if data are stored in memory), you should have a good reason for doing so, as standardization is a best practice.
`fbm`	Logical: is X stored as an FBM? Defaults to FALSE
`ns`	Optional vector of values indicating the indices of nonsingular features
`...`	Other optional arguments to `bigstatsr::bigapply()` (like `ncores = ...`)

Value

An n x n numeric matrix capturing the genomic relatedness of the samples represented in `X`. In our notation, we call this matrix K for 'kinship'; this is also known as the GRM or RRM.

Examples

```
RRM <- relatedness_mat(X = admix$X)
RRM[1:5, 1:5]
```

`summary.cv_plmm` *A summary function for cv_plmm objects*

Description

A summary function for cv_plmm objects

Usage

```
## S3 method for class 'cv_plmm'
summary(object, lambda = "min", ...)
```

Arguments

`object`	A `cv_plmm` object
`lambda`	The regularization parameter value at which inference should be reported. Can choose a numeric value, 'min', or '1se'. Defaults to 'min.'
`...`	Not used

Value

The return value is an object with S3 class `summary.cv_plmm`. The class has its own print method and contains the following list elements:

- `lambda_min`: The lambda value at the minimum cross validation error
- `lambda.1se`: The maximum lambda value within 1 standard error of the minimum cross validation error
- `penalty`: The penalty applied to the fitted model
- `nvars`: The number of non-zero coefficients at the selected lambda value
- `cve`: The cross validation error at all folds
- `min`: The minimum cross validation error
- `fit`: The `plmm` fit used in the cross validation

if `returnBiasDetails` = `TRUE`, two more items are returned:

- `bias`: The mean bias of the cross validation
- `loss`: The loss at each value of `lambda`

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
cv_fit <- cv_plmm(design = admix_design)
summary(cv_fit)
```

`summary.plmm` *A summary method for the plmm objects*

Description

A summary method for the plmm objects

Usage

```
## S3 method for class 'plmm'
summary(object, lambda, idx, eps = 1e-05, ...)
```

Arguments

`object`	An object of class `plmm`
`lambda`	The regularization parameter value at which inference should be reported.
`idx`	Alternatively, `lambda` may be specified by an index; `idx=10` means: report inference for the 10th value of `lambda` along the regularization path. If both `lambda` and `idx` are specified, `lambda` takes precedence.
`eps`	If lambda is given, eps is the tolerance for difference between the given lambda value and a lambda value from the object. Defaults to 0.0001 (1e-5)
`...`	Not used

Value

The return value is an object with S3 class `summary.plmm`. The class has its own print method and contains the following list elements:

- `penalty`: The penalty used by `plmm` (e.g. SCAD, MCP, lasso)
- `n`: Number of instances/observations
- `std_X_n`: the number of observations in the standardized data; the only time this would differ from 'n' is if data are from PLINK and the external data does not include all the same samples
- `p`: Number of regression coefficients (not including the intercept)
- `converged`: Logical indicator for whether the model converged
- `lambda`: The `lambda` value at which inference is being reported
- `lambda_char`: A formatted character string indicating the lambda value
- `nvars`: The number of nonzero coefficients (again, not including the intercept) at that value of `lambda`
- `nonzero`: The column names indicating the nonzero coefficients in the model at the specified value of `lambda`

Examples

```
admix_design <- create_design(X = admix$X, y = admix$y)
fit <- plmm(design = admix_design)
summary(fit, idx = 97)
```

`unzip_example_data` *For Linux/Unix and MacOS only, here is a companion function to unzip the .gz files that ship with the `plmmr` package*

Description

For Linux/Unix and MacOS only, here is a companion function to unzip the .gz files that ship with the `plmmr` package

Usage

```
unzip_example_data(outdir)
```

Arguments

`outdir`	The file path to the directory to which the .gz files should be written

Details

For an example of this function, look at `vignette('plink_files', package = "plmmr")`. Note again: this function will not work on Windows systems - only for Linux/Unix and MacOS.

Value

Nothing is returned; the PLINK files that ship with the `plmmr` package are stored in the directory specified by 'outdir'

Index