Package: LEGIT (via r-universe)

August 22, 2024

Title Latent Environmental & Genetic InTeraction (LEGIT) Model

Version 1.4.1

Date 2024-01-23

Author Alexia Jolicoeur-Martineau

`<alexia.jolicoeur-martineau@mail.mcgill.ca>`

Maintainer Alexia Jolicoeur-Martineau

`<alexia.jolicoeur-martineau@mail.mcgill.ca>`

Description Constructs genotype x environment interaction (GxE) models where G is a weighted sum of genetic variants (genetic score) and E is a weighted sum of environments (environmental score) using the alternating optimization algorithm by Jolicoeur-Martineau et al. (2017) <arXiv:1703.08111>. This approach has greatly enhanced predictive power over traditional GxE models which include only a single genetic variant and a single environmental exposure. Although this approach was originally made for GxE modelling, it is flexible and does not require the use of genetic and environmental variables. It can also handle more than 2 latent variables (rather than just G and E) and 3-way interactions or more. The LEGIT model produces highly interpretable results and is very parameter-efficient thus it can even be used with small sample sizes (n < 250). Tools to determine the type of interaction (vantage sensitivity, diathesis-stress or differential susceptibility), with any number of genetic variants or environments, are available <arXiv:1712.04058>. The software can now produce mixed-effects LEGIT models through the lme4 package.

License GPL-3

Imports pROC, foreach, snow, doSNOW, utils, iterators, Hmisc, grDevices, boot, RColorBrewer, glmnet, lme4, methods

Depends formula.tools, stats, graphics

Encoding UTF-8

RoxygenNote 7.3.0

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Date/Publication 2024-01-24 15:12:49 UTC

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

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

RemoteRef HEAD

RemoteSha d19facb6fce429a20c4e22def4e020bf83c0e391

Contents

best_model *Best model*

Description

Best model

Usage

```
best_model(object, ...)
```

Arguments

`object`	An object
`...`	Further arguments passed to or from other methods.

Value

Best model

best_model.elastic_net_var_select *Best model from elastic net variable selection*

Description

Best model from elastic net variable selection (based on selected criteria)

Usage

```
## S3 method for class 'elastic_net_var_select'
best_model(object, criterion, ...)
```

Arguments

`object`	An object of class "elastic_net_var_select", usually, a result of a call to elastic_net_var_select.
`criterion`	Criteria used to determine which model is the best. If `search_criterion="AIC"`, uses the AIC, if `search_criterion="AICc"`, uses the AICc, if `search_criterion="BIC"`, uses the BIC, if `search_criterion="cv_R2"`, uses the cross-validation R-squared, if `search_criterion="cv_AUC"`, uses the cross-validated AUC, if `search_criterion="cv_Huber"`, uses the Huber cross-validation error, if `search_criterion="cv_L1"`, uses the L1-norm cross-validation error (Default = "AIC"). The Huber and L1-norm

	cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers. For all criterion, lower is better, with the exception of `search_criterion="cv_R2"` and `search_criterion="cv_AUC"`.
`...`	Further arguments passed to or from other methods.

Value

Returns the best IMLEGIT model resulting from the glmnet path with associated information.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
## Not run:
N = 1000
train = example_3way(N, sigma=1, logit=FALSE, seed=7)
g1_bad = rbinom(N,1,.30)
g2_bad = rbinom(N,1,.30)
g3_bad = rbinom(N,1,.30)
g4_bad = rbinom(N,1,.30)
g5_bad = rbinom(N,1,.30)
train$G = cbind(train$G, g1_bad, g2_bad, g3_bad, g4_bad, g5_bad)
lv = list(G=train$G, E=train$E)
fit = elastic_net_var_select(train$data, lv, y ~ G*E)
summary(fit)
best_model(fit, criterion="BIC")
 # Instead of taking the best, if you want the model with "Model index"=17 from summary, do
plot(fit)
# With Cross-validation
fit = elastic_net_var_select(train$data, lv, y ~ G*E, cross_validation=TRUE, cv_iter=1, cv_folds=5)
best_model(fit, criterion="cv_R2")
# Elastic net only applied on G
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(1))
# Elastic net only applied on E
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2))
# Most E variables not removed, use lambda_mult > 1 to remove more
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2), lambda_mult=5)
# Lasso (only L1 regularization)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, alpha=1)
# Want more lambdas (useful if # of variables is large)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, n_lambda = 200)

## End(Not run)
```

`bootstrap_var_select` *Bootstrap variable selection (for IMLEGIT)*

Description

[Very slow, not recommended] Creates bootstrap samples, runs a stepwise search on all of them and then reports the percentage of times that each variable was selected. This is very computationally demanding. With small sample sizes, variable selection can be unstable and bootstrap can be used to give us an idea of the degree of certitude that a variable should be included or not.

Usage

```
bootstrap_var_select(
  data,
  formula,
  boot_iter = 1000,
  boot_size = NULL,
  boot_group = NULL,
  latent_var_original = NULL,
  latent_var_extra = NULL,
  search_type = "bidirectional-forward",
  search = 0,
  search_criterion = "AIC",
  forward_exclude_p_bigger = 0.2,
  backward_exclude_p_smaller = 0.01,
  exclude_worse_AIC = TRUE,
  max_steps = 100,
  start_latent_var = NULL,
  eps = 0.01,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  progress = TRUE,
  n_cluster = 1,
  best_subsets = 5,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).

`boot_iter`	number of bootstrap samples (Default = 1000).
`boot_size`	Optional size of the bootstrapped samples (Default = number of observations).
`boot_group`	Optional vector which represents the group associated with each observation. Sampling will be done by group instead of by observations (very important if you have longitudinal data). The sample sizes of the bootstrap samples might differ by up to "`boot_size` - maximum group size" observations.
`latent_var_original`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ...
`latent_var_extra`	list of data.frame (with the same structure as latent_var_original) containing the additional elements to try including inside the latent variables. Set to NULL if using a backward search.
`search_type`	If `search_type="forward"`, uses a forward search. If `search_type="backward"`, uses backward search. If `search_type="bidirectional-forward"`, uses bidirectional search (that starts as a forward search). If `search_type="bidirectional-backward"`, uses bidirectional search (that starts as a backward search).
`search`	If `search=0`, uses a stepwise search for all latent variables. Otherwise, if search = i, uses a stepwise search on the i-th latent variable (Default = 0).
`search_criterion`	Criterion used to determine which variable is the best to add or worst to drop. If `search_criterion="AIC"`, uses the AIC, if `search_criterion="AICc"`, uses the AICc, if `search_criterion="BIC"`, uses the BIC (Default = "AIC").
`forward_exclude_p_bigger`	If p-value > `forward_exclude_p_bigger`, we do not consider the variable for inclusion in the forward steps (Default = .20). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to 1 to prevent any exclusion here.
`backward_exclude_p_smaller`	If p-value < `backward_exclude_p_smaller`, we do not consider the variable for removal in the backward steps (Default = .01). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to 0 to prevent any exclusion here.
`exclude_worse_AIC`	If AIC with variable > AIC without variable, we ignore the variable (Default = TRUE). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to FALSE to prevent any exclusion here.
`max_steps`	Maximum number of steps taken (Default = 50).
`start_latent_var`	Optional list of starting points for each latent variable (The list must have the same length as the number of latent variables and each element of the list must

	have the same length as the number of variables of the corresponding latent variable).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`seed`	Optional seed for bootstrap.
`progress`	If TRUE, shows the progress done (Default=TRUE).
`n_cluster`	Number of parallel clusters, I recommend using the number of CPU cores - 1 (Default = 1).
`best_subsets`	If `best_subsets` = k, the output will show the k most frequently chosen subsets of variables (Default = 5)
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns a list of vectors containing the percentage of times that each variable was selected within each latent variable.

References

Peter C Austin and Jack V Tu. *Bootstrap Methods for Developing Predictive Models* (2012). dx.doi.org/10.1198/0003130043277.

Mark Reiser, Lanlan Yao, Xiao Wang, Jeanne Wilcox and Shelley Gray. *A Comparison of Bootstrap Confidence Intervals for Multi-level Longitudinal Data Using Monte-Carlo Simulation* (2017). 10.1007/978-981-10-3307-0_17.

Examples

```
## Not run:
## Example
train = example_3way_3latent(250, 2, seed=777)
# Bootstrap with Bidirectional-backward search for everything based on AIC
# Normally you should use a lot more than 10 iterations and extra CPUs (n_cluster)
boot = bootstrap_var_select(train$data, latent_var_extra=NULL,
latent_var_original=train$latent_var,
formula=y ~ E*G*Z,search_type="bidirectional-backward", search=0,
search_criterion="AIC", boot_iter=10, n_cluster=1)
# Assuming it's longitudinal with 5 timepoints, even though it's not
id = factor(rep(1:50,each=5))
```

```
boot_longitudinal = bootstrap_var_select(train$data, latent_var_extra=NULL,
latent_var_original=train$latent_var,
formula=y ~ E*G*Z,search_type="bidirectional-backward", search=0,
search_criterion="AIC", boot_iter=10, n_cluster=1, boot_group=id)

## End(Not run)
```

---

elastic_net_var_select

Elastic net for variable selection in IMLEGIT model

Description

[Fast and accurate, highly recommended] Apply Elastic Net (from the glmnet package) with IMLEGIT to obtain the order of variable removal that makes the most sense. The output shows the information criterion at every step, so you can decide which variable to retain. It is significantly faster (seconds/minutes instead of hours) than all other variable selection approaches (except for stepwise) and it is very accurate. Note that, as opposed to LEGIT/IMLEGIT, the parameters of variables inside the latent variables are not L1-normalized; instead, its the main model parameters which are L1-normalized. This is needed to make elastic net works. It doesn't matter in the end, because we only care about which variables were removed and we only output the IMLEGIT models without elastic net penalization.

Usage

```
elastic_net_var_select(
  data,
  latent_var,
  formula,
  latent_var_searched = NULL,
  cross_validation = FALSE,
  alpha = 0.75,
  standardize = TRUE,
  lambda_path = NULL,
  lambda_mult = 1,
  lambda_min = 1e-04,
  n_lambda = 100,
  start_latent_var = NULL,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  cv_iter = 5,
  cv_folds = 10,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  print = TRUE,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ... (See examples below for more details)
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`latent_var_searched`	Optional If not null, you must specify a vector containing all indexes of the latent variables you want to use elastic net on. Ex: If latent_var=list(G=genes, E=env), specifying latent_var_search=c(1,2) will use both, latent_var_search=1 will only do it for G, and latent_var_search=2 will only do it for E.
`cross_validation`	(Optional) If TRUE, will return cross-validation criterion (slower, but very good criterion).
`alpha`	The elasticnet mixing parameter (between 0 and 1). 1 leads to lasso, 0 leads to ridge. See glmnet package manual for more information. We recommend somewhere betwen .50 and 1.
`standardize`	If TRUE, standardize all variables inside every latent_var component. Note that if FALSE, glmnet will still standardize and unstandardize, but it will do so for each model (i.e., when at the step of estimating the parameters of latent variable G it standardize them, apply glmnet, then unstandarize them). This means that fixed parameters in the alternating steps are not standardized when standardize=FALSE. In practice, we found that standardize=FALSE leads to weird paths that do not always make sense. In the end, we only care about the order of the variable removal from the glmnet. We highly recommend standardize=TRUE for best results.
`lambda_path`	Optional vector of all lambda (penalty term for elastic net, see glmnet package manual). By default, we automatically determine it.
`lambda_mult`	scalar which multiplies the maximum lambda (penalty term for elastic net, see glmnet package manual) from the lambda path determined automatically. Sometimes, the maximum lambda found automatically is too big or too small and you may not want to spend the effort to manually set your own lambda path. This is where this comes in, you can simply scale lambda max up or down. (Default = 1)
`lambda_min`	minimum lambda (penalty term for elastic net, see glmnet package manual) from the lambda path. (Default = .0001)

`n_lambda`	Number of lambda (penalty term for elastic net, see glmnet package manual) in lambda path. Make lower for faster training, or higher for more precision. If you have many variables, make it bigger than 100 (Default = 100).
`start_latent_var`	Optional list of starting points for each latent variable (The list must have the same length as the number of latent variables and each element of the list must have the same length as the number of variables of the corresponding latent variable).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification (binary outcome).
`print`	If FALSE, nothing except warnings will be printed. (Default = TRUE).
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns an object of the class "elastic_net_var_select" which is list containing, in the following order: the criterion at each lambda, the coefficients of the latent variables at each lambda, the fits of each IMLEGIT models for each variable retained at each lambda, and the vector of lambda used.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
## Not run:
N = 1000
train = example_3way(N, sigma=1, logit=FALSE, seed=7)
g1_bad = rbinom(N,1,.30)
g2_bad = rbinom(N,1,.30)
g3_bad = rbinom(N,1,.30)
g4_bad = rbinom(N,1,.30)
g5_bad = rbinom(N,1,.30)
train$G = cbind(train$G, g1_bad, g2_bad, g3_bad, g4_bad, g5_bad)
lv = list(G=train$G, E=train$E)
fit = elastic_net_var_select(train$data, lv, y ~ G*E)
summary(fit)
best_model(fit, criterion="BIC")
 # Instead of taking the best, if you want the model with "Model index"=17 from summary, do
plot(fit)
# With Cross-validation
fit = elastic_net_var_select(train$data, lv, y ~ G*E, cross_validation=TRUE, cv_iter=1, cv_folds=5)
best_model(fit, criterion="cv_R2")
# Elastic net only applied on G
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(1))
# Elastic net only applied on E
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2))
# Most E variables not removed, use lambda_mult > 1 to remove more
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2), lambda_mult=5)
# Lasso (only L1 regularization)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, alpha=1)
# Want more lambdas (useful if # of variables is large)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, n_lambda = 200)

## End(Not run)
```

`example_2way` *Simulated example of a 2 way interaction GxE model.*

Description

Simulated example of a 2 way interaction GxE model (where G and E are latent variables).

$$g_j \sim Binomial(n = 1, p = .30)$$

$$j = 1, 2, 3, 4$$

$$e_l \sim Normal(\mu = 0, \sigma = 1.5)$$

$$l = 1, 2, 3$$

$$g = .2g_1 + .15g_2 - .3g_3 + .1g_4 + .05g_1g_3 + .2g_2g_3$$

$$e = -.45e_1 + .35e_2 + .2e_3$$

$$\mu = -1 + 2g + 3e + 4ge$$

$$y \sim Normal(\mu = \mu, \sigma = \texttt{sigma}) \text{ if } \texttt{logit}\text{=FALSE}$$
$$y \sim Binomial(n = 1, p = logit(\mu)) \text{ if } \texttt{logit}\text{=TRUE}$$

Usage

```
example_2way(N, sigma = 1, logit = FALSE, seed = NULL)
```

Arguments

`N`	Sample size.
`sigma`	Standard deviation of the gaussian noise (if `logit`=FALSE).
`logit`	If TRUE, the outcome is transformed to binary with a logit link.
`seed`	RNG seed.

Value

Returns a list containing, in the following order: data.frame with the observed outcome (with noise) and the true outcome (without noise), data.frame of the genetic variants (G), data.frame of the environments (E), vector of the true genetic coefficients, vector of the true environmental coefficients, vector of the true main model coefficients

Examples

```
example_2way(5,1,logit=FALSE)
example_2way(5,0,logit=TRUE)
```

`example_2way_lme4` *Simulated example of a 3 way interaction GxExZ model*

Description

Simulated example of a 3 way interaction GxExZ model (where G, E and Z are latent variables).

$$g_j \sim Binomial(n = 1, p = .30)$$
$$j = 1, 2, 3, 4$$
$$e_k \sim Normal(\mu = 0, \sigma = 1.5)$$
$$k = 1, 2, 3$$
$$z_l \sim Normal(\mu = 3, \sigma = 1)$$
$$l = 1, 2, 3$$
$$g = .2g_1 + .15g_2 - .3g_3 + .1g_4 + .05g_1g_3 + .2g_2g_3$$
$$e = -.45e_1 + .35e_2 + .2e_3$$
$$z = .15z_1 + .60z_2 + .25z_3$$

$$\mu = -2 + 2g + 3e + z + 5ge - 1.5ez + 2gz + 2gez$$

$$y \sim Normal(\mu = \mu, \sigma = \texttt{sigma}) \text{ if } \texttt{logit}\text{=FALSE}$$
$$y \sim Binomial(n = 1, p = logit(\mu)) \text{ if } \texttt{logit}\text{=TRUE}$$

Usage

```
example_2way_lme4(N, sigma = 1, logit = FALSE, seed = NULL)
```

Arguments

`N`	Sample size.
`sigma`	Standard deviation of the gaussian noise (if `logit`=FALSE).
`logit`	If TRUE, the outcome is transformed to binary with a logit link.
`seed`	RNG seed.

Value

Returns a list containing, in the following order: data.frame with the observed outcome (with noise) and the true outcome (without noise), list containing the data.frame of the genetic variants (G), the data.frame of the e environments (E) and the data.frame of the z environments (Z), vector of the true genetic coefficients, vector of the true e environmental coefficients, vector of the true z environmental coefficients, vector of the true main model coefficients

Examples

```
# Doing only one iteration so its faster
train = example_2way_lme4(250, 1, seed=777)
D = train$data
G = train$G
E = train$E

F = y ~ G*E
fit = LEGIT(D, G, E, F, lme4=FALSE, maxiter=1)
summary(fit)
F = y ~ 1
fit_test = GxE_interaction_test(D, G, E, F, criterion="AIC", lme4=FALSE, maxiter=1)
fit_test
#fit_test = GxE_interaction_test(D, G, E, F, criterion="cv", lme4=FALSE, maxiter=1, cv_iter=1)
#fit_test

F = y ~ G*E + (1|subject)
fit = LEGIT(D, G, E, F, lme4=TRUE, maxiter=1)
summary(fit)
F = y ~ (1|subject)
fit_test = GxE_interaction_test(D, G, E, F, criterion="AIC", lme4=TRUE, maxiter=1)
fit_test
#fit_test = GxE_interaction_test(D, G, E, F, criterion="cv", lme4=TRUE, maxiter=1, cv_iter=1)
#fit_test
```

example_3way *Simulated example of a 3 way interaction GxExz model*

Description

Simulated example of a 3 way interaction GxExz model (where G and E are latent variables).

$$g_j \sim Binomial(n = 1, p = .30)$$
$$j = 1, 2, 3, 4$$
$$e_l \sim Normal(\mu = 0, \sigma = 1.5)$$
$$l = 1, 2, 3$$
$$z \sim Normal(\mu = 3, \sigma = 1)$$
$$g = .2g_1 + .15g_2 - .3g_3 + .1g_4 + .05g_1g_3 + .2g_2g_3$$
$$e = -.45e_1 + .35e_2 + .2e_3$$
$$\mu = -2 + 2g + 3e + z + 5ge - 1.5ez + 2gz + 2gez$$

$y \sim Normal(\mu = \mu, \sigma =$ `sigma`$)$ if `logit`=FALSE
$y \sim Binomial(n = 1, p = logit(\mu))$ if `logit`=TRUE

Usage

```
example_3way(N, sigma = 2.5, logit = FALSE, seed = NULL)
```

Arguments

`N`	Sample size.
`sigma`	Standard deviation of the gaussian noise (if `logit`=FALSE).
`logit`	If TRUE, the outcome is transformed to binary with a logit link.
`seed`	RNG seed.

Value

Returns a list containing, in the following order: data.frame with the observed outcome (with noise), the true outcome (without noise) and z, data.frame of the genetic variants (G), data.frame of the environments (E), vector of the true genetic coefficients, vector of the true environmental coefficients, vector of the true main model coefficients

Examples

```
example_3way(5,2.5,logit=FALSE)
example_3way(5,0,logit=TRUE)
```

`example_3way_3latent` *Simulated example of a 3 way interaction GxExZ model*

Description

Simulated example of a 3 way interaction GxExZ model (where G, E and Z are latent variables).

$$g_j \sim Binomial(n = 1, p = .30)$$
$$j = 1, 2, 3, 4$$
$$e_k \sim Normal(\mu = 0, \sigma = 1.5)$$
$$k = 1, 2, 3$$
$$z_l \sim Normal(\mu = 3, \sigma = 1)$$
$$l = 1, 2, 3$$
$$g = .2g_1 + .15g_2 - .3g_3 + .1g_4 + .05g_1g_3 + .2g_2g_3$$
$$e = -.45e_1 + .35e_2 + .2e_3$$
$$z = .15z_1 + .60z_2 + .25z_3$$
$$\mu = -2 + 2g + 3e + z + 5ge - 1.5ez + 2gz + 2gez$$

$y \sim Normal(\mu = \mu, \sigma =$ `sigma`$)$ if `logit`=FALSE
$y \sim Binomial(n = 1, p = logit(\mu))$ if `logit`=TRUE

Usage

```
example_3way_3latent(N, sigma = 1, logit = FALSE, seed = NULL)
```

Arguments

`N`	Sample size.
`sigma`	Standard deviation of the gaussian noise (if `logit`=FALSE).
`logit`	If TRUE, the outcome is transformed to binary with a logit link.
`seed`	RNG seed.

Value

Returns a list containing, in the following order: data.frame with the observed outcome (with noise) and the true outcome (without noise), list containing the data.frame of the genetic variants (G), the data.frame of the e environments (E) and the data.frame of the z environments (Z), vector of the true genetic coefficients, vector of the true e environmental coefficients, vector of the true z environmental coefficients, vector of the true main model coefficients

Examples

```
example_3way_3latent(5,1,logit=FALSE)
example_3way_3latent(5,0,logit=TRUE)
```

`example_with_crossover`

Simulated example of a 2 way interaction GxE model with crossover point.

Description

Simulated example of a 2 way interaction GxE model with crossover point (where G and E are latent variables).

$$g_j \sim Binomial(n = 1, p = .30)$$
$$j = 1, 2, 3, 4$$
$$e_l \sim 10Beta(\alpha, \beta))$$
$$l = 1, 2, 3$$
$$g = .30g_1 + .10g_2 + .20g_3 + .40g_4$$
$$e = .45e_1 + .35e_2 + .2e_3$$
$$\mu = coef[1] + coef[2]e + coef[3]ge$$

$y \sim Normal(\mu = \mu, \sigma =$ `sigma`$)$ if `logit`=FALSE
$y \sim Binomial(n = 1, p = logit(\mu))$ if `logit`=TRUE

Usage

```
example_with_crossover(
  N,
  sigma = 1,
  c = 0,
  coef_main = c(0, 1, 2),
  coef_G = c(0.3, 0.1, 0.2, 0.4),
  coef_E = c(0.45, 0.35, 0.2),
  logit = FALSE,
  seed = NULL,
  beta_param = c(2, 2)
)
```

Arguments

`N`	Sample size.
`sigma`	Standard deviation of the gaussian noise (if `logit`=FALSE).
`c`	crossover point
`coef_main`	Coefficients of the main model, must be a vector of size 3 for intercept, E main effect and GxE effect (Default = c(0,1,2)).
`coef_G`	Coefficients of the 4 genes, must be a vector of size 4 (Default = c(.30, .10, .20, .40)).

`coef_E`	Coefficients of the 3 environments, must be a vector of size 3 (Default = c(.45, .35, .2)).
`logit`	If TRUE, the outcome is transformed to binary with a logit link.
`seed`	RNG seed.
`beta_param`	Vector of size two for the parameters of the beta distribution of the environmental variables (Default = c(2,2)).

Value

Returns a list containing, in the following order: data.frame with the observed outcome (with noise) and the true outcome (without noise), data.frame of the genetic variants (G), data.frame of the environments (E), vector of the true genetic coefficients, vector of the true environmental coefficients, vector of the true main model coefficients, the crossover point.

Examples

```
## Examples
# Diathesis Stress WEAK
ex_dia = example_with_crossover(250, c=10, coef_main = c(3,1,2), sigma=1)
# Diathesis Stress STRONG
ex_dia_s = example_with_crossover(250, c=10, coef_main = c(3,0,2), sigma=1)
# Differential Susceptibility WEAK
ex_ds = example_with_crossover(250, c=5, coef_main = c(3+5,1,2), sigma=1)
# Differential Susceptibility STRONG
ex_ds_s = example_with_crossover(250, c=5, coef_main = c(3+5,0,2), sigma=1)
```

`genetic_var_select` *Parallel genetic algorithm variable selection (for IMLEGIT)*

Description

[Very slow, recommended when the number of variables is large] Use a standard genetic algorithm with single-point crossover and a single mutation ran in parallel to find the best subset of variables. The percentage of times that each variable is included the final populations is also given. This is very computationally demanding but this finds much better solutions than either stepwise search or bootstrap variable selection.

Usage

```
genetic_var_select(
  data,
  formula,
  parallel_iter = 10,
  entropy_threshold = 0.1,
  popsize = 25,
  mutation_prob = 0.5,
  first_pop = NULL,
  latent_var = NULL,
  search_criterion = "AIC",
  maxgen = 100,
  eps = 0.01,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  progress = TRUE,
  n_cluster = 1,
  best_subsets = 5,
  cv_iter = 5,
  cv_folds = 5,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`parallel_iter`	number of parallel genetic algorithms (Default = 10). I recommend using 2-4 times the number of CPU cores used.
`entropy_threshold`	Entropy threshold for convergence of the population (Default = .10). Note that not reaching the entropy threshold just means that the population has some diversity, this is not necessarily a bad thing. Reaching the threshold is not necessary but if a population reach the threshold, we want it to stop reproducing (rather than continuing until `maxgen`) since the future generations won't change much.
`popsize`	Size of the population (Default = 25). Between 25 and 100 is generally adequate.
`mutation_prob`	Probability of mutation (Default = .50). A single variable is selected for mutation and it is mutated with probability `mutation_prob`. If the mutation causes a latent variable to become empty, no mutation is done. Using a small value (close to .05) will lead to getting more stuck in suboptimal solutions but using a large value (close to 1) will greatly increase the computing time because it will have a hard time reaching the entropy threshold.
`first_pop`	optional Starting initial population which is used instead of a fully random one. Mutation is also done on the initial population to increase variability.
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ...

`search_criterion`	Criterion used to determine which variable is the best to add or worst to drop. If `search_criterion="AIC"`, uses the AIC, if `search_criterion="AICc"`, uses the AICc, if `search_criterion="BIC"`, uses the BIC, if `search_criterion="cv"`, uses the cross-validation error, if `search_criterion="cv_AUC"`, uses the cross-validated AUC, if `search_criterion="cv_Huber"`, uses the Huber cross-validation error, if `search_criterion="cv_L1"`, uses the L1-norm cross-validation error (Default = "AIC"). The Huber and L1-norm cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers, the lower the values the better.
`maxgen`	Maximum number of generations (iterations) of the genetic algorithm (Default = 100). Between 50 and 200 generations is generally adequate.
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results). Note that using .001 rather than .01 (default) can more than double or triple the computing time of genetic_var_select.
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`seed`	Optional seed.
`progress`	If TRUE, shows the progress done (Default=TRUE).
`n_cluster`	Number of parallel clusters, I recommend using the number of CPU cores - 1 (Default = 1).
`best_subsets`	If `best_subsets = k`, the output will show the k best subsets of variables (Default = 5)
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification and cross-validation (binary outcome).
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns a list of vectors containing the percentage of times that each variable was included in the final populations, the criterion of the best k models, the starting points of the best k models (with the names of the best variables) and the entropy of the populations.

References

Mu Zhu, & Hugh Chipman. *Darwinian evolution in parallel universes: A parallel genetic algorithm for variable selection* (2006). Technometrics, 48(4), 491-502.

Examples

```
## Not run:
## Example
train = example_3way_3latent(250, 2, seed=777)
# Genetic algorithm based on BIC
# Normally you should use a lot more than 2 populations with 10 generations
ga = genetic_var_select(train$data, latent_var=train$latent_var,
formula=y ~ E*G*Z, search_criterion="AIC", parallel_iter=2, maxgen = 10)

## End(Not run)
```

`GxE_interaction_RoS` *Regions of significance using Johnson-Neyman technique*

Description

Constructs a LEGIT model and returns the regions of significance (RoS) with the predicted type of interaction (diathesis-stress, vantage-sensitivity, or differential susceptibility). RoS is not recommended due to poor accuracy with small samples and small effect sizes, GxE_interaction_test has much better accuracy overall. Only implemented for family=gaussian.

Usage

```
GxE_interaction_RoS(
  data,
  genes,
  env,
  formula_noGxE,
  t_alpha = 0.05,
  start_genes = NULL,
  start_env = NULL,
  eps = 0.001,
  maxiter = 100,
  ylim = NULL,
  reverse_code = FALSE,
  rescale = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`genes`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).
`env`	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
`formula_noGxE`	formula WITHOUT *G* or *E* (y ~ covariates). *G* and *E* will automatically be added.
`t_alpha`	Alpha level of the student-t distribution for the regions of significance (Default = .05)
`start_genes`	Optional starting points for genetic score (must be the same length as the number of columns of `genes`).
`start_env`	Optional starting points for environmental score (must be the same length as the number of columns of `env`).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`reverse_code`	If TRUE, after fitting the model, the genes with negative weights are reverse coded (ex: g_rev = 1 - g). It assumes that the original coding is in [0,1]. The purpose of this option is to prevent genes with negative weights which cause interpretation problems (ex: depression normally decreases attention but with a negative genetic score, it increases attention). Warning, using this option with GxG interactions could cause nonsensical results since GxG could be inverted. Also note that this may fail with certain models (Default=FALSE).
`rescale`	If TRUE, the environmental variables are automatically rescaled to the range [-1,1]. This improves interpretability (Default=FALSE).

Value

Returns a list containing the RoS and the predicted type of interaction.

References

Alexia Jolicoeur-Martineau, Jay Belsky, Eszter Szekely, Keith F. Widaman, Michael Pluess, Celia Greenwood and Ashley Wazana. *Distinguishing differential susceptibility, diathesis-stress and vantage sensitivity: beyond the single gene and environment model* (2017). https://osf.io/preprints/psyarxiv/27uw8. 10.17605/OSF.IO/27UW8.

Daniel J. Bauer & Patrick J. Curran. *Probing Interactions in Fixed and Multilevel Regression: Inferential and Graphical Techniques* (2005). Multivariate Behavioral Research, 40:3, 373-400, DOI: 10.1207/s15327906mbr4003_5.

Examples

```
train = example_2way(500, 1, seed=777)
ros = GxE_interaction_RoS(train$data, train$G, train$E, y ~ 1)
ros
```

`GxE_interaction_test` *Testing of the GxE interaction*

Description

Testing of the GxE interaction using the competitive-confirmatory approach adapted from Belsky, Pluess et Widaman (2013). Reports the different hypotheses (diathesis-stress, vantage-sensitivity, or differential susceptibility), assuming or not assuming a main effect for E (WEAK vs STRONG) using the LEGIT model.

Usage

```
GxE_interaction_test(
  data,
  genes,
  env,
  formula_noGxE,
  crossover = c("min", "max"),
  include_noGxE_models = TRUE,
  reverse_code = FALSE,
  rescale = FALSE,
  boot = NULL,
  criterion = "BIC",
  start_genes = NULL,
  start_env = NULL,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  cv_iter = 5,
  cv_folds = 10,
  folds = NULL,
  Huber_p = 1.345,
  id = NULL,
  classification = FALSE,
  seed = NULL,
  test_only = FALSE,
  lme4 = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`genes`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).
`env`	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
`formula_noGxE`	formula WITHOUT G or E (y ~ covariates). G and E will automatically be added properly based on the hypotheses tested.
`crossover`	A tuple containting the minimum and maximum of the environment used as crossover point of *E* used in the vantage sensitivity and diathesis-stress models. Instead of providing two number, you can also write c("min","max") to automatically choose the expected minimum or maximum of the environmental score which is calculated based on the min/max of the environments and the current weights.
`include_noGxE_models`	If True, we test for models with only G, only E, both G and E, neither G and E (four models without a GxE). This is to verify for false positives, if one of those models has the best fit, then it is possible that there is no GxE, thus no type of GxE. With a single gene and environment, simply looking at the p-value of the GxE is good enough to get around 5-10 percent false positive rate, but with multiple genes and environments, we need to compare model fits to get a low false positive rate. Use your own judgment when using this because if you have multiple genes and environments and small/moderate N, a model without GxE could have a lower BIC but still not be the actual best model. However, if you see little difference in BIC between all 4 GxE models and the non-GxE models have much lower BIC, than it is likely that there is no GxE. Note that this is only implemented for AIC, AICc and BIC. (Default = True)
`reverse_code`	If TRUE, after fitting the model, the genes with negative weights are reverse coded (ex: g_{rev} = 1 - g). It assumes that the original coding is in [0,1]. The purpose of this option is to prevent genes with negative weights which cause interpretation problems (ex: depression normally decreases attention but with a negative genetic score, it increases attention). Warning, using this option with GxG interactions could cause nonsensical results since GxG could be inverted. Also note that this may fail with certain models (Default=FALSE).
`rescale`	If TRUE, the environmental variables are automatically rescaled to the range [-1,1]. This improves interpretability (Default=FALSE).
`boot`	Optional number of bootstrap samples. If not NULL, we use bootstrap to find the confidence interval of the crossover point. This provides more realistic confidence intervals. Make sure to use a bigger number (>= 1000) to get good precision; also note that a too small number could return an error ("estimated adjustment 'a' is NA").
`criterion`	Criterion used to assess which model is the best. It can be set to "AIC", "AICc", "BIC", "cv", "cv_AUC", "cv_Huber" (Default="BIC").
`start_genes`	Optional starting points for genetic score (must be the same length as the number of columns of `genes`).

`start_env`	Optional starting points for environmental score (must be the same length as the number of columns of `env`).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`id`	Optional id of observations, can be a vector or data.frame (only used when returning list of possible outliers).
`classification`	Set to TRUE if you are doing classification (binary outcome).
`seed`	Seed for cross-validation folds.
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.
`lme4`	If TRUE, uses lme4::lmer or lme4::glmer; Note that is an experimental feature, bugs may arise and certain functions may fail. Currently only summary(), plot(), GxE_interaction_test(), LEGIT(), LEGIT_cv() work. Also note that the AIC and certain elements ignore the existence of the genes and environment variables, thus the AIC may not be used for variable selection of the genes and the environment. However, the AIC can still be used to compare models with the same genes and environments. (Default=FALSE).

Value

Returns a list containing 1) the six models ordered from best to worse (vantage sensitivity WEAK/STRONG, diathesis-stress WEAK/STRONG, differential susceptibility WEAK/STRONG) and 2) a data frame with the criterion, the crossover, 95% coverage of the crossover, whether the crossover 95% interval is within the observable range and the percentage of observations below the crossover point in order from best to worst based on the selected criterion. Models not within the observable range should be rejected even if the criterion is slightly better. An extremely low percentage of observations below the crossover point is also evidence toward diathesis-stress. Note that we assume that the environmental score is from bad to good but if this is not the case, then the models labelled as "diathesis-stress" could actually reflect vantage sensitivity and vice-versa. If outcome is Good-to-Bad: C=min(E) is diathesis-stress, C=max(E) is vantage sensitivity. If outcome is Bad-to-Good: C=max(E) is diathesis-stress, C=min(E) is vantage sensitivity.

References

Alexia Jolicoeur-Martineau, Jay Belsky, Eszter Szekely, Keith F. Widaman, Michael Pluess, Celia Greenwood and Ashley Wazana. *Distinguishing differential susceptibility, diathesis-stress and vantage sensitivity: beyond the single gene and environment model* (2017). https://osf.io/preprints/psyarxiv/27uw8. 10.17605/OSF.IO/27UW8.

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Jay Belsky, Michael Pluess and Keith F. Widaman. *Confirmatory and competitive evaluation of alternative gene-environment interaction hypotheses* (2013). Journal of Child Psychology and Psychiatry, 54(10), 1135-1143.

Examples

```
## Not run:
## Examples where x is in [0, 10]
# Diathesis Stress WEAK
ex_dia = example_with_crossover(250, c=10, coef_main = c(3,1,2), sigma=1)
# Diathesis Stress STRONG
ex_dia_s = example_with_crossover(250, c=10, coef_main = c(3,0,2), sigma=1)
## Assuming there is a crossover point at x=5
# Differential Susceptibility WEAK
ex_ds = example_with_crossover(250, c=5, coef_main = c(3+5,1,2), sigma=1)
# Differential Susceptibility STRONG
ex_ds_s = example_with_crossover(250, c=5, coef_main = c(3+5,0,2), sigma=1)

## If true model is "Diathesis Stress WEAK"
GxE_test_BIC = GxE_interaction_test(ex_dia$data, ex_dia$G, ex_dia$E,
formula_noGxE = y ~ 1, start_genes = ex_dia$coef_G, start_env = ex_dia$coef_E,
criterion="BIC")
GxE_test_BIC$results

## If true model is "Diathesis Stress STRONG"
GxE_test_BIC = GxE_interaction_test(ex_dia_s$data, ex_dia_s$G, ex_dia_s$E,
formula_noGxE = y ~ 1, start_genes = ex_dia_s$coef_G, start_env = ex_dia_s$coef_E,
criterion="BIC")
GxE_test_BIC$results

## If true model is "Differential susceptibility WEAK"
GxE_test_BIC = GxE_interaction_test(ex_ds$data, ex_ds$G, ex_ds$E,
formula_noGxE = y ~ 1, start_genes = ex_ds$coef_G, start_env = ex_ds$coef_E,
criterion="BIC")
GxE_test_BIC$results

## If true model is "Differential susceptibility STRONG"
GxE_test_BIC = GxE_interaction_test(ex_ds_s$data, ex_ds_s$G, ex_ds_s$E,
formula_noGxE = y ~ 1, start_genes = ex_ds_s$coef_G, start_env = ex_ds_s$coef_E,
criterion="BIC")
GxE_test_BIC$results
```

```
# Example of plots
plot(GxE_test_BIC$fits$diff_suscept_STRONG, xlim=c(0,10), ylim=c(3,13))
plot(GxE_test_BIC$fits$diff_suscept_WEAK, xlim=c(0,10), ylim=c(3,13))
plot(GxE_test_BIC$fits$diathesis_stress_STRONG, xlim=c(0,10), ylim=c(3,13))
plot(GxE_test_BIC$fits$diathesis_stress_WEAK, xlim=c(0,10), ylim=c(3,13))

## End(Not run)
```

IMLEGIT — *Independent Multiple Latent Environmental & Genetic InTeraction (IMLEGIT) model*

Description

Constructs a generalized linear model (glm) with latent variables using alternating optimization. This is an extension of the LEGIT model to accommodate more than 2 latent variables.

Usage

```
IMLEGIT(
  data,
  latent_var,
  formula,
  start_latent_var = NULL,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  print = TRUE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ... (See examples below for more details)
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`start_latent_var`	Optional list of starting points for each latent variable (The list must have the same length as the number of latent variables and each element of the list must

	have the same length as the number of variables of the corresponding latent variable).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`print`	If FALSE, nothing except warnings will be printed. (Default = TRUE).

Value

Returns an object of the class "IMLEGIT" which is list containing, in the following order: a glm fit of the main model, a list of the glm fits of the latent variables and a list of the true model parameters (AIC, BIC, rank, df.residual, null.deviance) for which the individual model parts (main, genetic, environmental) don't estimate properly.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
train = example_2way(500, 1, seed=777)
fit_best = IMLEGIT(train$data, list(G=train$G, E=train$E), y ~ G*E,
list(train$coef_G, train$coef_E))
fit_default = IMLEGIT(train$data, list(G=train$G, E=train$E), y ~ G*E)
summary(fit_default)
summary(fit_best)
train = example_3way_3latent(500, 1, seed=777)
fit_best = IMLEGIT(train$data, train$latent_var, y ~ G*E*Z,
list(train$coef_G, train$coef_E, train$coef_Z))
fit_default = IMLEGIT(train$data, train$latent_var, y ~ G*E*Z)
summary(fit_default)
summary(fit_best)
```

IMLEGIT_cv *Cross-validation for the IMLEGIT model*

Description

Uses cross-validation on the IMLEGIT model. Note that this is not a very fast implementation since it was written in R.

Usage

```
IMLEGIT_cv(
  data,
  latent_var,
  formula,
  cv_iter = 5,
  cv_folds = 10,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  start_latent_var = NULL,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  id = NULL,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ...
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.

`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification (binary outcome).
`start_latent_var`	Optional list of starting points for each latent variable (The list must have the same length as the number of latent variables and each element of the list must have the same length as the number of variables of the corresponding latent variable).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`seed`	Seed for cross-validation folds.
`id`	Optional id of observations, can be a vector or data.frame (only used when returning list of possible outliers).
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

If `classification` = FALSE, returns a list containing, in the following order: a vector of the cross-validated R^2 at each iteration, a vector of the Huber cross-validation error at each iteration, a vector of the L1-norm cross-validation error at each iteration, a matrix of the possible outliers (standardized residuals > 2.5 or < -2.5) and their corresponding standardized residuals and standardized pearson residuals. If `classification` = TRUE, returns a list containing, in the following order: a vector of the cross-validated R^2 at each iteration, a vector of the Huber cross-validation error at each iteration, a vector of the L1-norm cross-validation error at each iteration, a vector of the AUC at each iteration, a matrix of the best choice of threshold (based on Youden index) and the corresponding specificity and sensitivity at each iteration, and a list of objects of class "roc" (to be able to make roc curve plots) at each iteration. The Huber and L1-norm cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers, the lower the values the better.

References

Denis Heng-Yan Leung. *Cross-validation in nonparametric regression with outliers.* Annals of Statistics (2005): 2291-2310.

Examples

```
## Not run:
train = example_3way_3latent(250, 1, seed=777)
# Cross-validation 4 times with 5 Folds
cv_5folds = IMLEGIT_cv(train$data, train$latent_var, y ~ G*E*Z, cv_iter=4, cv_folds=5)
cv_5folds
# Leave-one-out cross-validation (Note: very slow)
cv_loo = IMLEGIT_cv(train$data, train$latent_var, y ~ G*E*Z, cv_iter=1, cv_folds=250)
cv_loo
# Cross-validation 4 times with 5 Folds (binary outcome)
train_bin = example_2way(500, 2.5, logit=TRUE, seed=777)
cv_5folds_bin = IMLEGIT_cv(train_bin$data, list(G=train_bin$G, E=train_bin$E), y ~ G*E,
cv_iter=4, cv_folds=5, classification=TRUE, family=binomial)
cv_5folds_bin
par(mfrow=c(2,2))
pROC::plot.roc(cv_5folds_bin$roc_curve[[1]])
pROC::plot.roc(cv_5folds_bin$roc_curve[[2]])
pROC::plot.roc(cv_5folds_bin$roc_curve[[3]])
pROC::plot.roc(cv_5folds_bin$roc_curve[[4]])

## End(Not run)
```

`IMLEGIT_net` *Independent Multiple Latent Environmental & Genetic InTeraction (IMLEGIT) model with Elastic Net on the latent variables. Do not use on it's own, use elastic_net_var_select instead.*

Description

Constructs a generalized linear model (glm) with latent variables using alternating optimization. This is an extension of the LEGIT model to accommodate more than 2 latent variables. Note that, as opposed to LEGIT/IMLEGIT, the parameters of variables inside the latent variables are not L1-normalized; instead, its the main model parameters which are L1-normalized. This is needed to make elastic net works. It doesn't matter in the end, because we only care about which variables were removed and we only give the IMLEGIT models without elastic net penalization.

Usage

```
IMLEGIT_net(
  data,
  latent_var,
  formula,
  latent_var_searched = NULL,
  cross_validation = FALSE,
  alpha = 1,
  lambda = 1e-04,
  start_latent_var = NULL,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  cv_iter = 5,
  cv_folds = 10,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  print = TRUE,
  warn = TRUE,
  family_string = NULL,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ... (See examples below for more details)
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`latent_var_searched`	Optional If not null, you must specify a vector containing all indexes of the latent variables you want to use elastic net on. Ex: If latent_var=list(G=genes, E=env), specifying latent_var_search=c(1,2) will use both, latent_var_search=1 will only do it for G, and latent_var_search=2 will only do it for E.
`cross_validation`	If TRUE, will return cross-validation criterion (slower)
`alpha`	The elasticnet mixing parameter (between 0 and 1). 1 leads to lasso, 0 leads to ridge. See glmnet package manual for more information. We recommend somewhere betwen .50 and 1.
`lambda`	Lambda (penalty term for elastic net, see glmnet package manual) (Default = .0001)
`start_latent_var`	Optional list of starting points for each latent variable (The list must have the same length as the number of latent variables and each element of the list must have the same length as the number of variables of the corresponding latent variable).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.

`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification (binary outcome).
`print`	If FALSE, nothing except warnings will be printed. (Default = TRUE).
`warn`	If FALSE, it will not show warnings when all variables inside a latent variable are removed. This serves to prevent lots of warning when running elastic_net_var_select (Default = TRUE).
`family_string`	Optional String version of the family (gaussian leads to "gaussian"). This is only needed when using elastic_net_var_select. Please ignore this.
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns a list containing, in the following order: a IMLEGIT model, the coefficients of the variables in the latent variables from glmnet models, and the cross-validation results (if asked).

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

`IMLEGIT_to_LEGIT` *IMLEGIT to LEGIT*

Description

Transforms a IMLEGIT model into a LEGIT model

Usage

```
IMLEGIT_to_LEGIT(
  fit,
  data,
  genes,
  env,
  formula,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  print = TRUE
)
```

Arguments

`fit`	IMLEGIT model
`data`	data.frame of the dataset to be used.
`genes`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).
`env`	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
`formula`	Model formula. Use *E* for the environmental score and *G* for the genetic score. Do not manually code interactions, write them in the formula instead (ex: G*E*z or G:E:z).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`print`	If FALSE, nothing except warnings will be printed (Default = TRUE).

Value

Returns an object of the class "LEGIT" which is list containing, in the following order: a glm fit of the main model, a glm fit of the genetic score, a glm fit of the environmental score, a list of the true model parameters (AIC, BIC, rank, df.residual, null.deviance) for which the individual model parts (main, genetic, environmental) don't estimate properly and the formula.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
train = example_2way(500, 1, seed=777)
fit = LEGIT(train$data, train$G, train$E, y ~ G*E, train$coef_G, train$coef_E)
fit_IMLEGIT = LEGIT_to_IMLEGIT(fit,train$data, train$G, train$E, y ~ G*E)
fit_LEGIT = IMLEGIT_to_LEGIT(fit_IMLEGIT,train$data, train$G, train$E, y ~ G*E)
```

LEGIT *Latent Environmental & Genetic InTeraction (LEGIT) model*

Description

Constructs a generalized linear model (glm) with a weighted latent environmental score and weighted latent genetic score using alternating optimization.

Usage

```
LEGIT(
  data,
  genes,
  env,
  formula,
  start_genes = NULL,
  start_env = NULL,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  print = TRUE,
  print_steps = FALSE,
  crossover = NULL,
  crossover_fixed = FALSE,
  reverse_code = FALSE,
  rescale = FALSE,
  lme4 = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.

`genes`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).
`env`	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
`formula`	Model formula. Use *E* for the environmental score and *G* for the genetic score. Do not manually code interactions, write them in the formula instead (ex: G*E*z or G:E:z).
`start_genes`	Optional starting points for genetic score (must be the same length as the number of columns of `genes`).
`start_env`	Optional starting points for environmental score (must be the same length as the number of columns of `env`).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`print`	If FALSE, nothing except warnings will be printed (Default = TRUE).
`print_steps`	If TRUE, print the parameters at all iterations, good for debugging (Default = FALSE).
`crossover`	If not NULL, estimates the crossover point of *E* using the provided value as starting point (To test for diathesis-stress vs differential susceptibility).
`crossover_fixed`	If TRUE, instead of estimating the crossover point of E, we force/fix it to the value of "crossover". (Used when creating a diathes-stress model) (Default = FALSE).
`reverse_code`	If TRUE, after fitting the model, the genes with negative weights are reverse coded (ex: g_rev = 1 - g). It assumes that the original coding is in [0,1]. The purpose of this option is to prevent genes with negative weights which cause interpretation problems (ex: depression normally decreases attention but with a negative genetic score, it increases attention). Warning, using this option with GxG interactions could cause nonsensical results since GxG could be inverted. Also note that this may fail with certain models (Default=FALSE).
`rescale`	If TRUE, the environmental variables are automatically rescaled to the range [-1,1]. This improves interpretability (Default=FALSE).
`lme4`	If TRUE, uses lme4::lmer or lme4::glmer; Note that is an experimental feature, bugs may arise and certain functions may fail. Currently only summary(), plot(), GxE_interaction_test(), LEGIT(), LEGIT_cv() work. Also note that the AIC and certain elements ignore the existence of the genes and environment variables, thus the AIC may not be used for variable selection of the genes and the environment. However, the AIC can still be used to compare models with the same genes and environments. (Default=FALSE).

Value

Returns an object of the class "LEGIT" which is list containing, in the following order: a glm fit of the main model, a glm fit of the genetic score, a glm fit of the environmental score, a list of the true model parameters (AIC, BIC, rank, df.residual, null.deviance) for which the individual model parts (main, genetic, environmental) don't estimate properly and the formula.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
train = example_2way(500, 1, seed=777)
fit_best = LEGIT(train$data, train$G, train$E, y ~ G*E, train$coef_G, train$coef_E)
fit_default = LEGIT(train$data, train$G, train$E, y ~ G*E)
summary(fit_default)
summary(fit_best)

train = example_3way(500, 2.5, seed=777)
fit_best = LEGIT(train$data, train$G, train$E, y ~ G*E*z, train$coef_G, train$coef_E)
fit_default = LEGIT(train$data, train$G, train$E, y ~ G*E*z)
summary(fit_default)
summary(fit_best)
```

LEGIT_cv *Cross-validation for the LEGIT model*

Description

Uses cross-validation on the LEGIT model. Note that this is not a very fast implementation since it was written in R.

Usage

```
LEGIT_cv(
  data,
  genes,
  env,
  formula,
  cv_iter = 5,
  cv_folds = 10,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  start_genes = NULL,
  start_env = NULL,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  id = NULL,
  crossover = NULL,
  crossover_fixed = FALSE,
  lme4 = FALSE,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`genes`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).
`env`	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
`formula`	Model formula. Use *E* for the environmental score and *G* for the genetic score. Do not manually code interactions, write them in the formula instead (ex: G*E*z or G:E:z).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification (binary outcome).
`start_genes`	Optional starting points for genetic score (must be the same length as the number of columns of `genes`).
`start_env`	Optional starting points for environmental score (must be the same length as the number of columns of `env`).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).

`seed`	Seed for cross-validation folds.
`id`	Optional id of observations, can be a vector or data.frame (only used when returning list of possible outliers).
`crossover`	If not NULL, estimates the crossover point of *E* using the provided value as starting point (To test for diathesis-stress vs differential susceptibility).
`crossover_fixed`	If TRUE, instead of estimating the crossover point of E, we force/fix it to the value of "crossover". (Used when creating a diathes-stress model) (Default = FALSE).
`lme4`	If TRUE, uses lme4::lmer or lme4::glmer; Note that is an experimental feature, bugs may arise and certain functions may fail. Currently only summary(), plot(), GxE_interaction_test(), LEGIT(), LEGIT_cv() work. Also note that the AIC and certain elements ignore the existence of the genes and environment variables, thus the AIC may not be used for variable selection of the genes and the environment. However, the AIC can still be used to compare models with the same genes and environments. (Default=FALSE).
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

If `classification` = FALSE, returns a list containing, in the following order: a vector of the cross-validated R^2 at each iteration, a vector of the Huber cross-validation error at each iteration, a vector of the L1-norm cross-validation error at each iteration, a matrix of the possible outliers (standardized residuals > 2.5 or < -2.5) and their corresponding standardized residuals and standardized pearson residuals. If `classification` = TRUE, returns a list containing, in the following order: a vector of the cross-validated R^2 at each iteration, a vector of the Huber cross-validation error at each iteration, a vector of the L1-norm cross-validation error at each iteration, a vector of the AUC at each iteration, a matrix of the best choice of threshold (based on Youden index) and the corresponding specificity and sensitivity at each iteration, and a list of objects of class "roc" (to be able to make roc curve plots) at each iteration. The Huber and L1-norm cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers, the lower the values the better.

References

Denis Heng-Yan Leung. *Cross-validation in nonparametric regression with outliers.* Annals of Statistics (2005): 2291-2310.

Examples

```
## Not run:
train = example_3way(250, 2.5, seed=777)
# Cross-validation 4 times with 5 Folds
cv_5folds = LEGIT_cv(train$data, train$G, train$E, y ~ G*E*z, cv_iter=4, cv_folds=5)
cv_5folds
# Leave-one-out cross-validation (Note: very slow)
```

```
cv_loo = LEGIT_cv(train$data, train$G, train$E, y ~ G*E*z, cv_iter=1, cv_folds=250)
cv_loo
# Test set only
cv_test = LEGIT_cv(train$data, train$G, train$E, y ~ G*E*z, cv_iter=1, cv_folds=5, test_only=TRUE)
cv_test
# Cross-validation 4 times with 5 Folds (binary outcome)
train_bin = example_2way(500, 2.5, logit=TRUE, seed=777)
cv_5folds_bin = LEGIT_cv(train_bin$data, train_bin$G, train_bin$E, y ~ G*E,
cv_iter=4, cv_folds=5, classification=TRUE, family=binomial)
cv_5folds_bin
par(mfrow=c(2,2))
pROC::plot.roc(cv_5folds_bin$roc_curve[[1]])
pROC::plot.roc(cv_5folds_bin$roc_curve[[2]])
pROC::plot.roc(cv_5folds_bin$roc_curve[[3]])
pROC::plot.roc(cv_5folds_bin$roc_curve[[4]])

## End(Not run)
```

LEGIT_to_IMLEGIT   *LEGIT to IMLEGIT*

Description

Transforms a LEGIT model into a IMLEGIT model (Useful if you want to do plot() or GxE_interaction_test() with a model resulting from a variable selection method which gave a IMLEGIT model)

Usage

```
LEGIT_to_IMLEGIT(
  fit,
  data,
  genes,
  env,
  formula,
  eps = 0.001,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  print = TRUE
)
```

Arguments

`fit`	LEGIT model
`data`	data.frame of the dataset to be used.
`genes`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).

env	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
formula	Model formula. Use *E* for the environmental score and *G* for the genetic score. Do not manually code interactions, write them in the formula instead (ex: G*E*z or G:E:z).
eps	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
maxiter	Maximum number of iterations.
family	Outcome distribution and link function (Default = gaussian).
ylim	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
print	If FALSE, nothing except warnings will be printed (Default = TRUE).

Value

Returns an object of the class "IMLEGIT" which is list containing, in the following order: a glm fit of the main model, a list of the glm fits of the latent variables and a list of the true model parameters (AIC, BIC, rank, df.residual, null.deviance) for which the individual model parts (main, genetic, environmental) don't estimate properly.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
train = example_2way(500, 1, seed=777)
fit = LEGIT(train$data, train$G, train$E, y ~ G*E, train$coef_G, train$coef_E)
fit_IMLEGIT = LEGIT_to_IMLEGIT(fit,train$data, train$G, train$E, y ~ G*E)
fit_LEGIT = IMLEGIT_to_LEGIT(fit_IMLEGIT,train$data, train$G, train$E, y ~ G*E)
```

---

longitudinal_folds *Longitudinal folds*

Description

Function to create folds adequately for longitudinal datasets by forcing every observation with the same id to be in the same fold. Can be used with LEGIT_cv to make sure that the cross-validation folds are appropriate when using longitudinal data.

Usage

```
longitudinal_folds(
  cv_iter = 1,
  cv_folds = 10,
  id,
  formula = NULL,
  data = NULL,
  data_needed = NULL,
  print = TRUE
)
```

Arguments

`cv_iter`	Number of cross-validation iterations (Default = 1).
`cv_folds`	Number of cross-validation folds (Default = 10).
`id`	Factor vector containing the id number of each observation.
`formula`	Optional Model formula. If data and formula are provided, only the non-missing observations will be used when creating the folds (Put "formula" here if you have missing data).
`data`	Optional data.frame used for the formula. If data and formula are provided, only the non-missing observations will be used when creating the folds (Put "data" here if you have missing data).
`data_needed`	Optional data.frame with variables that have to be included (Put "cbind(genes,env)"" or "latent_var" here if you have missing data).
`print`	If FALSE, nothing except warnings will be printed. (Default = TRUE).

Value

Returns a list of vectors containing the fold number for each observation

Examples

```
train = example_2way(500, 1, seed=777)
# Assuming it's longitudinal with 4 timepoints, even though it's not
id = factor(rep(1:125,each=4))
fit_cv = LEGIT_cv(train$data, train$G, train$E, y ~ G*E, folds=longitudinal_folds(1,10, id))
```

`nes_var_select` *Parallel natural evolutionary variable selection assuming bernouilli distribution (for IMLEGIT)*

Description

[Slow, highly recommended when the number of variables is large] Use natural evolution strategy (nes) gradient descent ran in parallel to find the best subset of variables. It is often as good as genetic algorithms but much faster so it is the recommended variable selection function to use as default. Note that this approach assumes that the inclusion of a variable does not depends on whether other variables are included (i.e. it assumes independent bernouilli distributions); this is generally not true but this approach still converge well and running it in parallel increases the probability of reaching the global optimum.

Usage

```
nes_var_select(
  data,
  formula,
  parallel_iter = 3,
  alpha = c(1, 5, 10),
  entropy_threshold = 0.05,
  popsize = 25,
  lr = 0.2,
  prop_ignored = 0.5,
  latent_var = NULL,
  search_criterion = "AICc",
  n_cluster = 3,
  eps = 0.01,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  progress = TRUE,
  cv_iter = 5,
  cv_folds = 5,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  print = FALSE,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`parallel_iter`	number of parallel tries (Default = 3). For speed, I recommend using the number of CPU cores.

`alpha`	vector of the parameter for the Dirichlet distribution of the starting points (Assuming a symmetric Dirichlet distribution with only one parameter). If the vector has size N and parralel_iter=K, we use alpha[1], ..., alpha[N], alpha[1], ... , alpha[N], ... for parallel_iter 1 to K respectively. We assume a dirichlet distribution for the starting points to get a bit more variability and make sure we are not missing on a great subset of variable that doesn't converge to the global optimum with the default starting points. Use bigger values for less variability and lower values for more variability (Default = c(1,5,10)).
`entropy_threshold`	Entropy threshold for convergence of the population (Default = .10). The smaller the entropy is, the less diversity there is in the population, which means convergence.
`popsize`	Size of the population, the number of subsets of variables sampled at each iteration (Default = 25). Between 25 and 100 is generally adequate.
`lr`	learning rate of the gradient descent, higher will converge faster but more likely to get stuck in local optium (Default = .2).
`prop_ignored`	The proportion of the population that are given a fixed fitness value, thus their importance is greatly reduce. The higher it is, the longer it takes to converge. Highers values makes the algorithm focus more on favorizing the good subsets of variables than penalizing the bad subsets (Default = .50).
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ...
`search_criterion`	Criterion used to determine which variable subset is the best. If `search_criterion="AIC"`, uses the AIC, if `search_criterion="AICc"`, uses the AICc, if `search_criterion="BIC"`, uses the BIC, if `search_criterion="cv"`, uses the cross-validation error, if `search_criterion="cv_AUC"`, uses the cross-validated AUC, if `search_criterion="cv_Huber"`, uses the Huber cross-validation error, if `search_criterion="cv_L1"`, uses the L1-norm cross-validation error (Default = "AIC"). The Huber and L1-norm cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers, the lower the values the better.
`n_cluster`	Number of parallel clusters, I recommend using the number of CPU cores (Default = 1).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results). Note that using .001 rather than .01 (default) can more than double or triple the computing time of genetic_var_select.
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures

	that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`seed`	Optional seed.
`progress`	If TRUE, shows the progress done (Default=TRUE).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification and cross-validation (binary outcome).
`print`	If TRUE, print the parameters of the search distribution and the entropy at each iteration. Note: Only works using Rterm.exe in Windows due to parallel clusters. (Default = FALSE).
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns a list containing the best subset's fit, cross-validation output, latent variables and starting points.

Examples

```
## Not run:
## Example
train = example_3way_3latent(250, 2, seed=777)
nes = nes_var_select(train$data, latent_var=train$latent_var,
formula=y ~ E*G*Z)

## End(Not run)
```

`plot.elastic_net_var_select`

Plot function for the output of elastic_net_var_select

Description

Plot of the coefficients of variables inside the latent variables with respect to the log(lambda). This is your typical elastic-net plot.

Usage

```
## S3 method for class 'elastic_net_var_select'
plot(x, lwd = 2, start = 1, ...)
```

Arguments

`x`	An object of class "elastic_net_var_select", usually, a result of a call to elastic_net_var_select.
`lwd`	Thickness of the lines (Default = 2)
`start`	At which lambda to start (from large lambda to small lambda). If start is not 1, we remove some of the large lambda, this can make plot easier to visualize (Default = 1).
`...`	Further arguments passed to or from other methods.

Value

Returns the plot of the coefficients of variables inside the latent variables with respect to the log(lambda).

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
## Not run:
N = 1000
train = example_3way(N, sigma=1, logit=FALSE, seed=7)
g1_bad = rbinom(N,1,.30)
g2_bad = rbinom(N,1,.30)
g3_bad = rbinom(N,1,.30)
g4_bad = rbinom(N,1,.30)
g5_bad = rbinom(N,1,.30)
train$G = cbind(train$G, g1_bad, g2_bad, g3_bad, g4_bad, g5_bad)
lv = list(G=train$G, E=train$E)
fit = elastic_net_var_select(train$data, lv, y ~ G*E)
summary(fit)
best_model(fit, criterion="BIC")
 # Instead of taking the best, if you want the model with "Model index"=17 from summary, do
plot(fit)
# With Cross-validation
fit = elastic_net_var_select(train$data, lv, y ~ G*E, cross_validation=TRUE, cv_iter=1, cv_folds=5)
best_model(fit, criterion="cv_R2")
# Elastic net only applied on G
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(1))
# Elastic net only applied on E
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2))
```

```
# Most E variables not removed, use lambda_mult > 1 to remove more
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2), lambda_mult=5)
# Lasso (only L1 regularization)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, alpha=1)
# Want more lambdas (useful if # of variables is large)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, n_lambda = 200)

## End(Not run)
```

---

plot.LEGIT *Plot*

Description

Plot of LEGIT models. By default, variables that are not in *G* or *E* are fixed to the mean.

Usage

```
## S3 method for class 'LEGIT'
plot(
  x,
  cov_values = NULL,
  gene_quant = c(0.025, 0.5, 0.975),
  env_quant = c(0.025, 0.5, 0.975),
  outcome_quant = c(0.025, 0.5, 0.975),
  cols = c("#3288BD", "#CAB176", "#D53E4F"),
  ylab = "Outcome",
  xlab = "Environment",
  legtitle = "Genetic score",
  leglab = NULL,
  xlim = NULL,
  ylim = NULL,
  x_at = NULL,
  y_at = NULL,
  cex.axis = 1.9,
  cex.lab = 2,
  cex.main = 2.2,
  cex.leg = 2.2,
  legend = "topleft",
  ...
)
```

Arguments

x	An object of class "LEGIT", usually, a result of a call to LEGIT.
cov_values	Vector of the values, for each covariate, that will be used in the plotting, if there are any covariates. It must contain the names of the variables. Covariates are the variables that are not *G* nor *E* but still are adjusted for in the model. By default, covariates are fixed to the mean.

`gene_quant`	Vector of the genes quantiles used to make the plot. We use quantiles instead of fixed values because genetic scores can vary widely depending on the weights, thus looking at quantiles make this simpler. (Default = c(.025,.50,.975))
`env_quant`	Vector of the environments quantiles used to make the plot. We use quantiles instead of fixed values because environmental scores can vary widely depending on the weights, thus looking at quantiles make this simpler. (Default = c(.025,.50,.975))
`outcome_quant`	Vector of the outcome quantiles used to make the plot. We use quantiles instead of fixed values because environmental scores can vary widely depending on the weights, thus looking at quantiles make this simpler. (Default = c(.025,.50,.975))
`cols`	Colors for the slopes with different genetic score. Must be a vector same length as "gene_range". (Default = c("#3288BD", "#CAB176", #D53E4F"))
`ylab`	Y-axis label (Default = "Outcome")
`xlab`	X-axis label (Default = "Environment")
`legtitle`	Title of the Legend for the genes slopes label (Default = "Genetic score")
`leglab`	Optional vector of labels of the Legend for the genes slopes label
`xlim`	X-axis vector of size two with min and max (Default = NULL which leads to min="2.5 percentile" and max="97.5 percentile").
`ylim`	Y-axis vector of size two with min and max (Default = NULL which leads to min="2.5 percentile" and max="97.5 percentile").
`x_at`	specific ticks for the X-axis, first and last will be min and max respectively (Default = NULL which leads to 2.5, 50 and 97.5 percentiles).
`y_at`	specific ticks for the Y-axis, first and last will be min and max respectively (Default = NULL which leads to 2.5, 50 and 97.5 percentiles).
`cex.axis`	relative scale of axis (Default = 1.9)
`cex.lab`	relative scale of labels (Default = 2)
`cex.main`	relative scale overall (Default = 2.2)
`cex.leg`	relative scale of legend (Default = 2.2)
`legend`	The location may of the legend be specified by setting legend to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center" (Default = "topleft").
`...`	Further arguments passed to or from other methods.

Value

Returns a list containing the different models (diathesis-stress, differential susceptibility and vantage sensitivity WEAK or STRONG) in order from best to worst for each selected criterion.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
train = example_2way(500, 1, seed=777)
fit = LEGIT(train$data, train$G, train$E, y ~ G*E, train$coef_G, train$coef_E)
plot(fit)
```

---

predict.IMLEGIT *Predictions of IMLEGIT fits*

Description

Predictions of IMLEGIT fits.

Usage

```
## S3 method for class 'IMLEGIT'
predict(object, data, latent_var, ...)
```

Arguments

object	An object of class "IMLEGIT", usually, a result of a call to IMLEGIT.
data	data.frame of the dataset to be used.
latent_var	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ...
...	Further arguments passed to or from other methods.

Value

Returns a vector with the predicted values.

Examples

```
train = example_2way(250, 1, seed=777)
test = example_2way(100, 1, seed=666)
fit = IMLEGIT(train$data, list(G=train$G, E=train$E), y ~ G*E)
ssres = sum((test$data$y - predict(fit, test$data, list(G=test$G, E=test$E)))^2)
sstotal = sum((test$data$y - mean(test$data$y))^2)
R2 = 1 - ssres/sstotal
R2
```

predict.LEGIT *Predictions of LEGIT fits*

Description

Predictions of LEGIT fits.

Usage

```
## S3 method for class 'LEGIT'
predict(object, data, genes, env, ...)
```

Arguments

`object`	An object of class "LEGIT", usually, a result of a call to LEGIT.
`data`	data.frame of the dataset to be used.
`genes`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).
`env`	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
`...`	Further arguments passed to or from other methods.

Value

Returns a vector with the predicted values.

Examples

```
train = example_2way(250, 1, seed=777)
test = example_2way(100, 1, seed=666)
fit = LEGIT(train$data, train$G, train$E, y ~ G*E)
ssres = sum((test$data$y - predict(fit, test$data, test$G, test$E))^2)
sstotal = sum((test$data$y - mean(test$data$y))^2)
R2 = 1 - ssres/sstotal
```

r1nes_var_select *Parallel natural evolutionary variable selection assuming multivariate normal search distribution with a simple covariance matrix parametrization (for IMLEGIT)*

Description

[Slow, highly recommended when the number of variables is large] Use natural evolution strategy (nes) gradient descent ran in parallel to find the best subset of variables. It is often as good as genetic algorithms but much faster so it is the recommended variable selection function to use as default. This is slower than nes_var_select but much less likely to get stuck into local optimum so the parallelization is not really needed.

Usage

```
r1nes_var_select(
  data,
  formula,
  parallel_iter = 3,
  alpha = c(1, 5, 10),
  entropy_threshold = 0.05,
  popsize = 25,
  lr = 0.2,
  prop_ignored = 0.5,
  latent_var = NULL,
  search_criterion = "AICc",
  n_cluster = 3,
  eps = 0.01,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  progress = TRUE,
  cv_iter = 5,
  cv_folds = 5,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  print = FALSE,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`parallel_iter`	number of parallel tries (Default = 3). For speed, I recommend using the number of CPU cores.
`alpha`	vector of the parameter for the Dirichlet distribution of the starting points (Assuming a symmetric Dirichlet distribution with only one parameter). If the vector has size N and parralel_iter=K, we use alpha[1], ..., alpha[N], alpha[1], ... , alpha[N], ... for parallel_iter 1 to K respectively. We assume a dirichlet distribution for the starting points to get a bit more variability and make sure we are not missing on a great subset of variable that doesn't converge to the global optimum with the default starting points. Use bigger values for less variability and lower values for more variability (Default = c(1,5,10)).
`entropy_threshold`	Entropy threshold for convergence of the population (Default = .10). The smaller

	the entropy is, the less diversity there is in the population, which means convergence.
`popsize`	Size of the population, the number of subsets of variables sampled at each iteration (Default = 25). Between 25 and 100 is generally adequate.
`lr`	learning rate of the gradient descent, higher will converge faster but more likely to get stuck in local optium (Default = .2).
`prop_ignored`	The proportion of the population that are given a fixed fitness value, thus their importance is greatly reduce. The higher it is, the longer it takes to converge. Highers values makes the algorithm focus more on favorizing the good subsets of variables than penalizing the bad subsets (Default = .50).
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ...
`search_criterion`	Criterion used to determine which variable subset is the best. If `search_criterion="AIC"`, uses the AIC, if `search_criterion="AICc"`, uses the AICc, if `search_criterion="BIC"`, uses the BIC, if `search_criterion="cv"`, uses the cross-validation error, if `search_criterion="cv_AUC"`, uses the cross-validated AUC, if `search_criterion="cv_Huber"`, uses the Huber cross-validation error, if `search_criterion="cv_L1"`, uses the L1-norm cross-validation error (Default = "AIC"). The Huber and L1-norm cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers, the lower the values the better.
`n_cluster`	Number of parallel clusters, I recommend using the number of CPU cores (Default = 1).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results). Note that using .001 rather than .01 (default) can more than double or triple the computing time of genetic_var_select.
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`seed`	Optional seed.
`progress`	If TRUE, shows the progress done (Default=TRUE).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.

`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification and cross-validation (binary outcome).
`print`	If TRUE, print the parameters of the search distribution and the entropy at each iteration. Note: Only works using Rterm.exe in Windows due to parallel clusters. (Default = FALSE).
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns a list containing the best subset's fit, cross-validation output, latent variables and starting points.

Examples

```
## Not run:
## Example
train = example_3way_3latent(250, 2, seed=777)
nes = r1nes_var_select(train$data, latent_var=train$latent_var,
formula=y ~ E*G*Z)

## End(Not run)
```

`rGE` *Gene-Environment correlation estimation and testing*

Description

Estimates the gene-environment correlation (rGE) and tests for a GxE using a residual environmental score. If there is an important correlation between G and E, the model is still valid prediction-wise but the interpretation is affected as the question becomes: is it really a GxE or a GxG since E is partially caused by G? To account for this, we remove the influence of G on E (If E = b0 + b1*G + e, we use E_resid = E - b1*G) and refit the model to see if the model parameters changed. The residual environmental score (E_resid) is uncorrelated with G. This does not account for passive rGE but only active rGE.

Usage

```
rGE(object, ...)
```

Arguments

`object`	An object of class "LEGIT" or "IMLEGIT".
`...`	Further arguments passed to or from other methods.

rGE.IMLEGIT *Gene-Environment correlation estimation and testing of IMLEGIT models*

Description

Estimates the gene-environment correlation (rGE) and tests for a GxE using a residual environmental score. If there is an important correlation between G and E, the model is still valid prediction-wise but the interpretation is affected as the question becomes: is it really a GxE or a GxG since E is partially caused by G? To account for this, we remove the influence of G on E (If E = b0 + b1*G + e, we use E_resid = E - b1*G) and refit the model to see if the model parameters changed. The residual environmental score (E_resid) is uncorrelated with G. This does not account for passive rGE but only active rGE.

Usage

```
## S3 method for class 'IMLEGIT'
rGE(object, formula, latent_var, index_E, index_G, ...)
```

Arguments

`object`	An object of class "IMLEGIT", usually, a result of a call to IMLEGIT.
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`latent_var`	list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise the latent variables will be named L1, L2, ... (See examples below for more details)
`index_E`	vector or scalar representing the index of each latent variable that is part of the "environment"
`index_G`	scalar representing the index of the latent variable for the "genetic" part
`...`	Further arguments passed to or from other methods.

Value

Returns a list containing the Pearson correlation and Kendall tau correlation of G and E and a glm fit of the main model part when removing the influence of G on E so that E and G are now uncorrelated.

Examples

```
# Note: These examples don't have G and E correlation so the model fit doesn't change
# but this shows how to use the rGE function
train = example_3way_3latent(500, 1, seed=777)
fit = IMLEGIT(train$data, train$latent_var, y ~ G*E*Z)
# If we assume Z not to be an "environment"
fit_rGE1 = rGE(fit, y ~ G*E, train$latent_var, 2, 1)
fit_rGE1
summary(fit_rGE1$fit_main_resid)
# If we assume Z to be an "environment"
fit_rGE2 = rGE(fit, y ~ G*E, train$latent_var, c(2,3), 1)
fit_rGE2
summary(fit_rGE2$fit_main_resid)
```

rGE.LEGIT *Gene-Environment correlation estimation and testing of LEGIT models*

Description

Estimates the gene-environment correlation (rGE) and tests for a GxE using a residual environmental score. If there is an important correlation between G and E, the model is still valid prediction-wise but the interpretation is affected as the question becomes: is it really a GxE or a GxG since E is partially caused by G? To account for this, we remove the influence of G on E (If E = b0 + b1*G + e, we use E_resid = E - b1*G) and refit the model to see if the model parameters changed. The residual environmental score (E_resid) is uncorrelated with G. This does not account for passive rGE but only active rGE.

Usage

```
## S3 method for class 'LEGIT'
rGE(object, formula, ...)
```

Arguments

`object`	An object of class "LEGIT", usually, a result of a call to LEGIT.
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can be used in formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).
`...`	Further arguments passed to or from other methods.

Value

Returns a list containing the Pearson correlation and Kendall tau correlation of G and E and a glm fit of the main model part when removing the influence of G on E so that E and G are now uncorrelated.

Examples

```
# Note: These examples don't have G and E correlation so the model fit doesn't change
# but this shows how to use the rGE function
train = example_2way(500, 1, seed=777)
fit = LEGIT(train$data, train$G, train$E, y ~ G*E)
fit_rGE = rGE(fit, y ~ G*E)
fit_rGE
summary(fit_rGE$fit_main_resid)
```

`stepwise_search` *Stepwise search for the best subset of genetic variants or environments with the LEGIT model*

Description

[Fast, recommended for small number of variables] Adds the best variable or drops the worst variable one at a time in the genetic (if `search="genes"`) or environmental score (if `search="env"`). You can select the desired search criterion (AIC, BIC, cross-validation error, cross-validation AUC) to determine which variable is the best/worst and should be added/dropped. Note that when the number of variables in *G* and *E* is large, this does not generally converge to the optimal subset, this function is only recommended when you have a small number of variables (e.g. 2 environments, 6 genetic variants). If using cross-validation (`search_criterion="cv"` or `search_criterion="cv_AUC"`), to prevent cross-validating with each variable (extremely slow), we recommend setting a p-value threshold (`p_threshold`) and forcing the algorithm not to look at models with bigger AIC (`exclude_worse_AIC=TRUE`).

Usage

```
stepwise_search(
  data,
  formula,
  interactive_mode = FALSE,
  genes_original = NULL,
  env_original = NULL,
  genes_extra = NULL,
  env_extra = NULL,
  search_type = "bidirectional-forward",
  search = "both",
  search_criterion = "AIC",
  forward_exclude_p_bigger = 0.2,
  backward_exclude_p_smaller = 0.01,
  exclude_worse_AIC = TRUE,
  max_steps = 100,
  cv_iter = 5,
  cv_folds = 10,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  start_genes = NULL,
  start_env = NULL,
  eps = 0.01,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  print = TRUE,
  remove_miss = FALSE,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`formula`	Model formula. Use *E* for the environmental score and *G* for the genetic score. Do not manually code interactions, write them in the formula instead (ex: G*E*z or G:E:z).
`interactive_mode`	If TRUE, uses interactive mode. In interactive mode, at each iteration, the user is shown the AIC, BIC, p-value and also the cross-validation R^2 if `search_criterion="cv"` and the cross-validation AUC if `search_criterion="cv_AUC"` for the best 5 variables. The user must then enter a number between 1 and 5 to select the variable to be added, entering anything else will stop the search.
`genes_original`	data.frame of the variables inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic).
`env_original`	data.frame of the variables inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental).
`genes_extra`	data.frame of the additionnal variables to try including inside the genetic score *G* (can be any sort of variable, doesn't even have to be genetic). Set to NULL if using a backward search.
`env_extra`	data.frame of the variables to try including inside the environmental score *E* (can be any sort of variable, doesn't even have to be environmental). Set to NULL if using a backward search.
`search_type`	If `search_type="forward"`, uses a forward search. If `search_type="backward"`, uses backward search. If `search_type="bidirectional-forward"`, uses bidirectional search (that starts as a forward search). If `search_type="bidirectional-backward"`, uses bidirectional search (that starts as a backward search).
`search`	If `search="genes"`, uses a stepwise search for the genetic score variables. If `search="env"`, uses a stepwise search for the environmental score variables. If `search="both"`, uses a stepwise search for both the gene and environmental score variables (Default = "both").
`search_criterion`	Criterion used to determine which variable is the best to add or worst to drop. If `search_criterion="AIC"`, uses the AIC, if `search_criterion="AICc"`, uses the AICc, if `search_criterion="BIC"`, uses the BIC, if `search_criterion="cv"`,

	uses the cross-validation error, if `search_criterion="cv_AUC"`, uses the cross-validated AUC, if `search_criterion="cv_Huber"`, uses the Huber cross-validation error, if `search_criterion="cv_L1"`, uses the L1-norm cross-validation error (Default = "AIC"). The Huber and L1-norm cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers, the lower the values the better.
`forward_exclude_p_bigger`	If p-value > `forward_exclude_p_bigger`, we do not consider the variable for inclusion in the forward steps (Default = .20). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to 1 to prevent any exclusion here.
`backward_exclude_p_smaller`	If p-value < `backward_exclude_p_smaller`, we do not consider the variable for removal in the backward steps (Default = .01). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to 0 to prevent any exclusion here.
`exclude_worse_AIC`	If AIC with variable > AIC without variable, we ignore the variable (Default = TRUE). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to FALSE to prevent any exclusion here.
`max_steps`	Maximum number of steps taken (Default = 50).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification (binary outcome).
`start_genes`	Optional starting points for genetic score (must be the same length as the number of columns of `genes`).
`start_env`	Optional starting points for environmental score (must be the same length as the number of columns of `env`).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).

seed	Seed for cross-validation folds.
print	If TRUE, print all the steps and notes/warnings. Highly recommended unless you are batch running multiple stepwise searchs. (Default=TRUE).
remove_miss	If TRUE, remove missing data completely, otherwise missing data is only removed when adding or dropping a variable (Default = FALSE).
test_only	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns an object of the class "LEGIT" which is list containing, in the following order: a glm fit of the main model, a glm fit of the genetic score, a glm fit of the environmental score, a list of the true model parameters (AIC, BIC, rank, df.residual, null.deviance) for which the individual model parts (main, genetic, environmental) don't estimate properly.

Examples

```
## Not run:
## Continuous example
train = example_3way(250, 2.5, seed=777)
# Forward search for genes based on BIC (in interactive mode)
forward_genes_BIC = stepwise_search(train$data, genes_extra=train$G, env_original=train$E,
formula=y ~ E*G*z,search_type="forward", search="genes", search_criterion="BIC",
interactive_mode=TRUE)
# Bidirectional-backward search for environments based on cross-validation error
bidir_backward_env_cv = stepwise_search(train$data, genes_original=train$G, env_original=train$E,
formula=y ~ E*G*z,search_type="bidirectional-backward", search="env", search_criterion="cv")
## Binary example
train_bin = example_2way(500, 2.5, logit=TRUE, seed=777)
# Forward search for genes based on cross-validated AUC (in interactive mode)
forward_genes_AUC = stepwise_search(train_bin$data, genes_extra=train_bin$G,
env_original=train_bin$E, formula=y ~ E*G,search_type="forward", search="genes",
search_criterion="cv_AUC", classification=TRUE, family=binomial, interactive_mode=TRUE)
# Forward search for genes based on AIC
bidir_forward_genes_AIC = stepwise_search(train_bin$data, genes_extra=train_bin$G,
env_original=train_bin$E, formula=y ~ E*G,search_type="bidirectional-forward", search="genes",
search_criterion="AIC", classification=TRUE, family=binomial)

## End(Not run)
```

---

stepwise_search_IM *Stepwise search for the best subset of elements in the latent variables with the IMLEGIT model*

Description

[Fast, recommended when the number of variables is small] Adds the best variable or drops the worst variable one at a time in the latent variables. You can select the desired search criterion (AIC, BIC, cross-validation error, cross-validation AUC) to determine which variable is the best/worst and should be added/dropped. Note that when the number of variables in *G* and *E* is large, this does not generally converge to the optimal subset, this function is only recommended when you have a small number of variables (e.g. 2 environments, 6 genetic variants). If using cross-validation (`search_criterion="cv"` or `search_criterion="cv_AUC"`), to prevent cross-validating with each variable (extremely slow), we recommend setting a p-value threshold (`p_threshold`) and forcing the algorithm not to look at models with bigger AIC (`exclude_worse_AIC=TRUE`).

Usage

```
stepwise_search_IM(
  data,
  formula,
  interactive_mode = FALSE,
  latent_var_original = NULL,
  latent_var_extra = NULL,
  search_type = "bidirectional-forward",
  search = 0,
  search_criterion = "AIC",
  forward_exclude_p_bigger = 0.2,
  backward_exclude_p_smaller = 0.01,
  exclude_worse_AIC = TRUE,
  max_steps = 100,
  cv_iter = 5,
  cv_folds = 10,
  folds = NULL,
  Huber_p = 1.345,
  classification = FALSE,
  start_latent_var = NULL,
  eps = 0.01,
  maxiter = 100,
  family = gaussian,
  ylim = NULL,
  seed = NULL,
  print = TRUE,
  remove_miss = FALSE,
  test_only = FALSE
)
```

Arguments

`data`	data.frame of the dataset to be used.
`formula`	Model formula. The names of `latent_var` can be used in the formula to represent the latent variables. If names(`latent_var`) is NULL, then L1, L2, ... can

be used in the formula to represent the latent variables. Do not manually code interactions, write them in the formula instead (ex: G*E1*E2 or G:E1:E2).

`interactive_mode`
: If TRUE, uses interactive mode. In interactive mode, at each iteration, the user is shown the AIC, BIC, p-value and also the cross-validation R^2 if `search_criterion="cv"` and the cross-validation AUC if `search_criterion="cv_AUC"` for the best 5 variables. The user must then enter a number between 1 and 5 to select the variable to be added, entering anything else will stop the search.

`latent_var_original`
: list of data.frame. The elements of the list are the datasets used to construct each latent variable. For interpretability and proper convergence, not using the same variable in more than one latent variable is highly recommended. It is recommended to set names to the list elements to prevent confusion because otherwise, the latent variables will be named L1, L2, ...

`latent_var_extra`
: list of data.frame (with the same structure as latent_var_original) containing the additionnal elements to try including inside the latent variables. Set to NULL if using a backward search.

`search_type`
: If `search_type="forward"`, uses a forward search. If `search_type="backward"`, uses backward search. If `search_type="bidirectional-forward"`, uses bidirectional search (that starts as a forward search). If `search_type="bidirectional-backward"`, uses bidirectional search (that starts as a backward search).

`search`
: If `search=0`, uses a stepwise search for all latent variables. Otherwise, if search = i, uses a stepwise search on the i-th latent variable (Default = 0).

`search_criterion`
: Criterion used to determine which variable is the best to add or worst to drop. If `search_criterion="AIC"`, uses the AIC, if `search_criterion="AICc"`, uses the AICc, if `search_criterion="BIC"`, uses the BIC, if `search_criterion="cv"`, uses the cross-validation error, if `search_criterion="cv_AUC"`, uses the cross-validated AUC, if `search_criterion="cv_Huber"`, uses the Huber cross-validation error, if `search_criterion="cv_L1"`, uses the L1-norm cross-validation error (Default = "AIC"). The Huber and L1-norm cross-validation errors are alternatives to the usual cross-validation L2-norm error (which the R^2 is based on) that are more resistant to outliers, the lower the values the better.

`forward_exclude_p_bigger`
: If p-value > `forward_exclude_p_bigger`, we do not consider the variable for inclusion in the forward steps (Default = .20). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to 1 to prevent any exclusion here.

`backward_exclude_p_smaller`
: If p-value < `backward_exclude_p_smaller`, we do not consider the variable for removal in the backward steps (Default = .01). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to 0 to prevent any exclusion here.

`exclude_worse_AIC`	If AIC with variable > AIC without variable, we ignore the variable (Default = TRUE). This is an exclusion option which purpose is skipping variables that are likely not worth looking to make the algorithm faster, especially with cross-validation. Set to FALSE to prevent any exclusion here.
`max_steps`	Maximum number of steps taken (Default = 50).
`cv_iter`	Number of cross-validation iterations (Default = 5).
`cv_folds`	Number of cross-validation folds (Default = 10). Using `cv_folds=NROW(data)` will lead to leave-one-out cross-validation.
`folds`	Optional list of vectors containing the fold number for each observation. Bypass cv_iter and cv_folds. Setting your own folds could be important for certain data types like time series or longitudinal data.
`Huber_p`	Parameter controlling the Huber cross-validation error (Default = 1.345).
`classification`	Set to TRUE if you are doing classification (binary outcome).
`start_latent_var`	Optional list of starting points for each latent variable (The list must have the same length as the number of latent variables and each element of the list must have the same length as the number of variables of the corresponding latent variable).
`eps`	Threshold for convergence (.01 for quick batch simulations, .0001 for accurate results).
`maxiter`	Maximum number of iterations.
`family`	Outcome distribution and link function (Default = gaussian).
`ylim`	Optional vector containing the known min and max of the outcome variable. Even if your outcome is known to be in [a,b], if you assume a Gaussian distribution, predict() could return values outside this range. This parameter ensures that this never happens. This is not necessary with a distribution that already assumes the proper range (ex: [0,1] with binomial distribution).
`seed`	Seed for cross-validation folds.
`print`	If TRUE, print all the steps and notes/warnings. Highly recommended unless you are batch running multiple stepwise searchs. (Default=TRUE).
`remove_miss`	If TRUE, remove missing data completely, otherwise missing data is only removed when adding or dropping a variable (Default = FALSE).
`test_only`	If TRUE, only uses the first fold for training and predict the others folds; do not train on the other folds. So instead of cross-validation, this gives you train/test and you get the test R-squared as output.

Value

Returns an object of the class "IMLEGIT" which is list containing, in the following order: a glm fit of the main model, a list of the glm fits of the latent variables and a list of the true model parameters (AIC, BIC, rank, df.residual, null.deviance) for which the individual model parts (main, genetic, environmental) don't estimate properly.

Examples

```
## Not run:
## Example
train = example_3way_3latent(250, 1, seed=777)
# Forward search for genes based on BIC (in interactive mode)
forward_genes_BIC = stepwise_search_IM(train$data,
latent_var_original=list(G=NULL, E=train$latent_var$E, Z=train$latent_var$Z),
latent_var_extra=list(G=train$latent_var$G,E=NULL,Z=NULL),
formula=y ~ E*G*Z,search_type="forward", search=1, search_criterion="BIC",
interactive_mode=TRUE)
# Bidirectional-backward search for everything based on AIC
bidir_backward_AIC = stepwise_search_IM(train$data, latent_var_extra=NULL,
latent_var_original=train$latent_var,
formula=y ~ E*G*Z,search_type="bidirectional-backward", search=0, search_criterion="AIC")

## End(Not run)
```

`summary.elastic_net_var_select` *Summary function for the output of elastic_net_var_select*

Description

Summary function for the output of elastic_net_var_select

Usage

```
## S3 method for class 'elastic_net_var_select'
summary(object, ...)
```

Arguments

`object`	An object of class "elastic_net_var_select", usually, a result of a call to elastic_net_var_select.
`...`	Further arguments passed to or from other methods.

Value

Returns the unique IMLEGIT models resulting from the glmnet path with associated information. Also gives the cross-validation information if asked.

References

Alexia Jolicoeur-Martineau, Ashley Wazana, Eszter Szekely, Meir Steiner, Alison S. Fleming, James L. Kennedy, Michael J. Meaney, Celia M.T. Greenwood and the MAVAN team. *Alternating optimization for GxE modelling with weighted genetic and environmental scores: examples from the MAVAN study* (2017). arXiv:1703.08111.

Examples

```
## Not run:
N = 1000
train = example_3way(N, sigma=1, logit=FALSE, seed=7)
g1_bad = rbinom(N,1,.30)
g2_bad = rbinom(N,1,.30)
g3_bad = rbinom(N,1,.30)
g4_bad = rbinom(N,1,.30)
g5_bad = rbinom(N,1,.30)
train$G = cbind(train$G, g1_bad, g2_bad, g3_bad, g4_bad, g5_bad)
lv = list(G=train$G, E=train$E)
fit = elastic_net_var_select(train$data, lv, y ~ G*E)
summary(fit)
best_model(fit, criterion="BIC")
 # Instead of taking the best, if you want the model with "Model index"=17 from summary, do
plot(fit)
# With Cross-validation
fit = elastic_net_var_select(train$data, lv, y ~ G*E, cross_validation=TRUE, cv_iter=1, cv_folds=5)
best_model(fit, criterion="cv_R2")
# Elastic net only applied on G
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(1))
# Elastic net only applied on E
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2))
# Most E variables not removed, use lambda_mult > 1 to remove more
fit = elastic_net_var_select(train$data, lv, y ~ G*E, c(2), lambda_mult=5)
# Lasso (only L1 regularization)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, alpha=1)
# Want more lambdas (useful if # of variables is large)
fit = elastic_net_var_select(train$data, lv, y ~ G*E, n_lambda = 200)

## End(Not run)
```

---

summary.IMLEGIT *Summarizing IMLEGIT fits*

Description

Shows the summary for all parts (main and latent variables) of the LEGIT model.

Usage

```
## S3 method for class 'IMLEGIT'
summary(object, ...)
```

Arguments

`object`	An object of class "IMLEGIT", usually, a result of a call to IMLEGIT.
`...`	Further arguments passed to or from other methods.

Value

Returns a list of objects of class "summary.glm" containing the summary of each parts (main and latent variables) of the model.

Examples

```
 train = example_2way(250, 1, seed=777)
fit_default = IMLEGIT(train$data, list(G=train$G, E=train$E), y ~ G*E)
summary(fit_default)
```

`summary.LEGIT` *Summarizing LEGIT fits*

Description

Shows the summary for all parts (main, genetic, environmental) of the LEGIT model.

Usage

```
## S3 method for class 'LEGIT'
summary(object, ...)
```

Arguments

`object`	An object of class "LEGIT", usually, a result of a call to LEGIT.
`...`	Further arguments passed to or from other methods.

Value

Returns a list of objects of class "summary.glm" containing the summary of each parts (main, genetic, environmental) of the model.

Examples

```
 train = example_2way(250, 1, seed=777)
fit_default = LEGIT(train$data, train$G, train$E, y ~ G*E)
summary(fit_default)
```

Index