Package 'HIP'

July 3, 2024

Type Package

Title Heterogeneity in Integration and Prediction (HIP)

Version 0.1.0

Author Leif Verace (verac008@umn.edu), Jessica Butts (butts029@umn.edu), and Sandra Safo (ssafo@umn.edu)

Maintainer Sandra Safo <ssafo@umn.edu>

Description We present HIP (Heterogeneity in Integration and Prediction) for integration of multiple data views while accounting for subgroup heterogeneity to identify common and subgroup-
specific variables of interest for a particular outcome. HIP accommodates Gaussian, multi-class, Poisson, and zero-inflated Poisson outcomes. In addition to this package, HIP may be accessed through an R Shiny application found at <https://multi-viewlearn.shinyapps.io/HIP_ShinyApp/>.
For details, refer to the paper found on arXiv: <https://arxiv.org/abs/2111.06962>

License GPL (>=2)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Depends R (>= 2.10)

Imports reticulate,
plyr,
dplyr,
ggplot2,
rTorch,
tibble,
ggpubr,
latex2exp,
magrittr,
parallel,
writexl

R topics documented:

`covid_data` *Example COVID-19 data*

Description

Data taken from a multi-omic study of COVID-19 severity. This particular dataset has Sex (male/female) as the subgroup variable and HFD45 (the number of hospital free days out of the 45 days enrolled in the study) as the outcome variable; a zero indicates the patient was either still in the hospital or died. There are two views in the data: genomic and proteomic data. The genomic data consists of RNAseq expression data for 5800 genes in each patient, while the proteomic data consists of mass spectrometry data for 264 proteins in each patient. Each also has an ID variable such as 'COVID_01' which provides a label for each patient across the data views.

Usage

```
data("covid_data")
```

Format

The data is divided up into 6 dataframes: 3 for training data and 3 for testing data. The training datasets consist of 100 observations, while the testing datasets consist of 20 observations. `X_train_genomic` and `X_test_genomic` are dataframes of 5801 variables: the first column is the ID variable, while the remaining columns are genes. `X_train_proteomic` and `X_test_proteomic` are dataframes of 265 variables, the first column being an ID variable and the remaining are proteins. `Y_train` and `Y_test` consist of 3 columns: the first is the ID variable, the second column is the subgroup variable variable Sex, and the third column is the outcome variable HFD45.

Source

Source of data: Overmeyer et al. (2021), https://doi.org/10.1016/j.cels.2020.10.003

Preprocessing of data: Lipman et al. (2022), https://doi.org/10.1371/journal.pone.0267047

create_virtualenv *Create virtual environment*

Description

Creates a Python virtual environment and installs necessary packages

Usage

```
create_virtualenv()
```

Details

This function creates a Python virtual environment called "HIP_env" and installs necessary packages such as torch, numpy, and matplotlib into the environment. Once created, the environment will be used when the package is loaded.

Examples

```
create_virtualenv()
```

fixed_lambda *Fits a HIP model given lambda values*

Description

`fixed_lambda` fits a model from X, Y data and fixed λ_ξ, λ_G supplied by the user.

Usage

```
fixed_lambda(X, Y, gamma, family, topn, lambda_xi=1, lambda_g=1, K=NULL,
k_thresh=0.2, update_thresh=10^-5, epsilon=10^-4, max_iter=50,
standardize='subgroup', std_type='scale_center', std_x=TRUE,
std_y=TRUE, verbose=FALSE)
```

Arguments

`X`	matrix list - list of X^d,s matrices containing covariates
`Y`	matrix list - list of Y^s matrices containing outcomes
`family`	string - family of outcome; options are 'gaussian', 'multiclass', 'poisson' or 'zip'
`topn`	int or list - number of variables to retain; different values may be specified for each view using a list of length D
`lambda_xi`	double - value of lambda_xi in penalty term
`lambda_g`	double - value of lambda_g in penalty term
`K`	int - number of latent components; will use 'select_K_simple' to choose if not provided

k_thresh	double - threshold to use for 'select_K_simple' if K not provided; default = 0.2
update_thresh	double - threshold to use for suboptimization convergence criterion; default = 10**-5
epsilon	double - threshold to use for overall convergence criterion; default = 10**-4
max_iter	int - maximum number of outer loop iterations; default = 50
standardize	string - One of "all", "subgroup", or "none"; default = "subgroup"
std_type	string - One of "scale_center", "center", or "norm"; default = "scale_center"
std_x	boolean - indicates whether to standardize X; default = TRUE
std_y	boolean - indicates whether to standardize Y; default = TRUE
verbose	boolean - whether to print additional information during optimization; default = FALSE

Value

full	list - results returned from optimize_torch on full data
include	list - list of length D with a 1 indicating the variable was included in subset fit and 0 indicating not included in subset fit
subset	list - results returned from optimize_torch on subset of variables

In addition, fixed_lambda returns -99 for best_index and NA for criterion to indicate they are not applicable to this case. topn is also returned.

Examples

```
# Generating data
dat_gen <- generate_data()

# Getting model from fixed_lambda
res <- fixed_lambda(dat_gen$X, dat_gen$Y, c(1,1), 'gaussian', 50)
```

---

format_data *Format data for use with HIP*

Description

User-supplied data must be formatted properly to work with HIP functions such as select_lambda or HIP_test_eval. Use this function to format the data, which will return a list containing training/testing data converted to PyTorch tensors and useful information such as the number of data views and subgroups. Note, either X or Y data must contain the subgroup variable, and each dataframe must have an ID variable. Y data should contain an outcome variable, which may belong to a Gaussian, multiclass, Poisson, or ZIP family. You may exclude the Y data if you simply want to format X data for use in prediction functions such as HIP_pred. See the example below for a demonstration of the function using a COVID-19 dataset included in the HIP package.

Usage

```
format_data(X_train, Y_train=NULL, id_var=NULL,
subgroup_var=NULL, outcome_var=NULL, X_test=NULL, Y_test=NULL,
family="", data_source="")
```

Arguments

X_train	list of dataframes - list of D dataframes containing X training data (where D is the number of data views). NOTE: X_train should also contain the subgroup_var if it is not contained in the Y_train data.
Y_train	list of dataframes - list of D dataframes containing Y training data
id_var	string - name of ID variable in data
subgroup_var	string - name of subgroup variable in data
outcome_var	string - name of outcome variable in data
X_test	list of dataframes - list of D dataframes containing X test data, if available
Y_test	list of datamframes - list of D dataframes containing Y test data, if available
family	string - family of outcome data; one of either "gaussian", "multiclass", "poisson", or "zip"

Details

As HIP uses functions integrated in Python, it is necessary to convert dataframes to PyTorch tensors prior to use. This function also gathers training and testing data together for use in functions such as `HIP_test_eval`. See individual function documentation for examples. Also note that data from the `generate_data` function is already formatted, so they do not need to be fed into this function.

Value

Returns a list with the following:

X	matrix list - Contains training data X^d,s matrices; access as `data$X[[d]][[s]]`
Y	matrix list - Contains training data Y^s matrices; access as `data$Y[[d]][[s]]`
X_test	matrix list - Contains test data X^d,s matrices; access as `data$X_test[[d]][[s]]`
Y_test	matrix list - Contains test data Y^s matrices; access as `data$Y_test[[d]][[s]]`
D	int - Number of data views
S	int - Number of unique subgroups
sub_vec	int vec - Vector with labels for subgroups
var_list	string list - D lists containing names of variables in X data

Examples

```
# Read in COVID Data
data("covid_data")

# Join data and declare variables for IDs, subgroups, and outcomes
X_train <- list(covid_data$X_train_genomic, covid_data$X_train_proteomic)
X_test <- list(covid_data$X_test_genomic, covid_data$X_test_proteomic)

id_var <- 'ID'
subgroup_var <- 'Sex'
S <- length(unique(covid_data$Y_train[[subgroup_var]]))
outcome_var <- 'HFD45'
D <- 2

# Structure data and convert it to Python format
formatted_data <- format_data(X_train, covid_data$Y_train, id_var, subgroup_var, outcome_var,
                              X_test, covid_data$Y_test, family='gaussian')
```

`generate_data` *Generate simulated data to use with HIP*

Description

`generate_data` returns a nested list containing data matrices and values used to generate the data.

Usage

```
generate_data(seed=1, n=c(250,260), p=c(300,350), K=2, D=2, S=2, nonzero=50, offset=25,
sigma_x=1, sigma_y=1, family='gaussian', test_data=FALSE,
q=1, m = 2, theta_init=NULL, B=NULL, Z=NULL, y_mean=0,
y_sd=1, z_mean=25, z_sd=3, beta=0, tau=NULL, t_dist=FALSE, t_df=20)
```

Arguments

`seed`	int or string - set a seed for replicability
`n`	int vector - number of subjects desired in each subgroup; should be length S
`p`	int vector - number of covariates desired in each data set; should be length D
`K`	int - number of latent components to use in generating the data
`D`	int - number of data sets to generate
`S`	int - number of subgroups
`nonzero`	int - number of important variables for each subgroup; same for all subgroups
`offset`	int - how many variables to offset the important variables between subgroups; same for all subgroups
`sigma_x`	double - factor by which the errors added to X are scaled by
`sigma_y`	double - factor by which the errors added to Y are scaled by; not applicable when family = 'poisson' or 'zip'
`family`	string - determines what type of outcome is generated: • 'gaussian' - q should also be specified or q will default to 1 • 'multiclass' - m should also be specified or m will default to 2 • 'poisson' • 'zip'
`test_data`	bool - TRUE to generate test data, FALSE otherwise
`q`	int - number of continuous outcomes; can be >= 1; default = 1
`m`	int - number of classes in multiclass outcome; can be >= 2; default = 2
`theta_init`	double matrix - matrix to use for theta; if None, then a theta matrix will be generated from a U(0,1) distribution
`B`	matrix list - B^d,s matrices used to generate the data; will randomly generate if not provided
`Z`	matrix list - Z^s matrices used to generate the data; if not provided, will generate as N(mu = z_mean, sigma = z_sd)
`y_mean`	double - Value for mean of Y; default = 0
`y_sd`	double - Value for sd of Y; default = 1

z_mean	double - Value for mean of Y; default = 0
z_sd	double - Value for sd of Y; default = 1
beta	double - Value of intercept term; default = 0
t_dist	boolean - if family = "gaussian", setting t_dist = TRUE will generate data with errors drawn from a t-distribution with some specified degrees of freedom
t_df	int - degrees of freedom for t-distribution if t_dist = TRUE; default = 20

Details

generate_data can be used to generate simulated multi-view data across different subgroups. The input parameters specify relevant information such as the number of data views, the number of subgroups, subjects desired for each subgroup, and more. The output of generate_data gives the $X^{d,s}$ data matrices and Y^s outcome matrices, as well as values used to generate the data such as the Z^s and $B^{d,s}$ matrices. See the HIP paper for details.

Value

generate_data returns a nested list object with the following elements:

X	matrix list - Contains X^d,s matrices; access as data$X[[d]][[s]]
Y	matrix list - Contains Y^s matrices; access as data$Y[[d]][[s]]
X_test	matrix list - Contains test data X^d,s matrices; access as data$X_test[[d]][[s]]
Y_test	matrix list - Contains test data Y^s matrices; access as data$Y_test[[d]][[s]]
D	int - Number of data sets generated
S	int - Number of subgroups
sub_vec	int vec - Vector with labels for subgroups (from 1 to S)
Z	matrix list - Contains all Z^s matrices used to generate the data; access as data[[Z]][[d]][[s]]
B	matrix list - Contains all B^d,s matrices used to generate the data; access as data[[B]][[d]][[s]]
theta	matrix - Contains the theta matrix used in generating the data; access as data$theta
beta	double - Value for intercept
tau	double - If family = 'zip', proportion of observations in zero state with default = 0.25; None ow
y_mean	double - Value for mean of Y
y_sd	double - Value for sd of Y
z_mean	double - Value for mean of Z
z_sd	double - Value for sd of Z
seed	int or string - Value of seed used to generate the data
var_list	vec list - Contains D vectors with labels for variables in each dataset (1 to p^d)

Examples

```
# The values below are the function defaults, but can be changed for other simulated datasets
family <- 'gaussian'
seed <- 1
nonzero <- 50
offset <- 25
n <- c(250,260)
K <- 2
S <- 2
D <- 2
p <- c(300,350)
sigma_x <- 1.0
sigma_y <- 1.0
```

```
# Generating data (no test data)
data_gen <- generate_data(seed, n, p, K, D, S, nonzero, offset, sigma_x,
sigma_y, family)

# Generating data (with test data)
data_gen_test <- generate_data(seed, n, p, K, D, S, nonzero, offset,
                               sigma_x, sigma_y, family, test_data=TRUE)
```

`get_best_r` *Helper function to get best result from select_lambda*

Description

'get_best_R' is used within the select_lambda function to get the best result based on model performance. Not generally called by the user.

Usage

```
get_best_r(res_list, criterion="", family)
```

Arguments

`res_list`	list - search_results list from the select_lambda output
`criterion`	string - criterion for model selection: one of 'CV', 'BIC', 'AIC', 'eBIC_0', 'eBIC_5', or 'eBIC_1'
`family`	string - family of outcome data ('gaussian', 'multiclass', etc.)

Details

get_best_R takes in the list of results from the select_lambda tuning function, and returns the index of the best result. It also takes in the criterion used in training such as AIC or BIC, in addition to the family the outcome data belongs to.

Value

`get_best_R` returns the index of the best results from the select_lambda output

Examples

```
# Generate data
dat_gen <- generate_data()

# Get results from select_lambda
res <- select_lambda(dat_gen$X, dat_gen$Y, c(1,1), "gaussian", 50, K=2, num_steps=c(4,4))

best_index <- get_best_r(res$out$search_results, "eBIC_0", "gaussian")
```

HIP_pred *Make predictions from HIP model*

Description

Outputs predicted Y values given X data and results from `select_lambda` or `fixed_lambda`.

Usage

```
HIP_pred(X_dat, res, fix_or_search, standardize="no")
```

Arguments

`X_dat`	matirx list - X^d,s matrices from `generate_data` or `format_data`
`res`	output from `select_lambda` or `fixed_lambda`
`fix_or_search`	string - 'fixed' if `res` comes from `fixed_lambda`, else 'search'
`standardize`	string - 'yes' to standardize Y data if it was standardized in `select_lambda`, else return unstandardized predicted Y values

Value

The function returns a list of S tensors containing predictions for observations in each subgroup.

Examples

```
# Generate data
dat_gen_test <- generate_data(test_data=TRUE)

# Get results from select_lambda
res <- select_lambda(dat_gen_test$X, dat_gen_test$Y, c(1,1), 'gaussian', 50,
                     K = 2, num_steps=c(4,4))

# Make predictions on X test data (and standardize predictions)
Y_pred <- HIP_pred(dat_gen_test$X_test, res, 'search', 'yes')
```

HIP_test_eval *Evaluate model on test data*

Description

`HIP_test_eval` is used to get model performance metrics (e.g. MSE or classification accuracy) on test data. It can be used with models generated by `select_lambda` or `fixed_lambda`.

Usage

```
HIP_test_eval(data, res, fix_or_search, outcome_var=NULL)
```

Arguments

data	data object such as from generate_data or py_data containing both X, Y data and X_test, Y_test data
res	training result such as from select_lambda or fixed_lambda
fix_or_search	string - 'fixed' for results from fixed_lambda, 'search' for results from select_lambda
outcome_var	string - label of outcome variable, if available

Value

HIP_test_eval returns a list with the following:

each	tensor(double) - (for gaussian outcomes) tensor of MSEs for each of the q outcomes
comp_val	tensor(double) - MSE for gaussian outcomes, classification accuracy for multiclass outcomes, fraction of deviance explained for poisson/ZIP outcomes
pred	list<tensor(double) - predicted values from fitted model. For multiclass outcomes, single integer to represent class, starting with 0. For poisson/ZIP, predicted means from fitted model
dev_sum	list(tensor(double)) - (for poisson/ZIP outcomes), list with more detailed deviance information

Also returns a histogram plot for predicted vs. true values in each subgroup.

Examples

```
# Generate data
dat_gen_test <- generate_data(test_data=TRUE)

# Get results from select_lambda
res <- select_lambda(dat_gen_test$X, dat_gen_test$Y, c(1,1), 'gaussian', 50,
                     K = 2, num_steps=c(4,4))

# Evaluating on test data
test_eval <- HIP_test_eval(dat_gen_test, res, 'search')
```

HIP_train_eval *Evaluate model on training data*

Description

HIP_train_eval is used to get training metrics (e.g. MSE or classification accuracy) from a model built using HIP. It can be used with models generated by select_lambda or fixed_lambda.

Usage

```
HIP_train_eval(data, res, fix_or_search)
```

Arguments

data	data object such as from generate_data or py_data
res	training result such as from select_lambda or fixed_lambda
fix_or_search	string - 'fixed' for results from fixed_lambda, 'search' for results from select_lambda

Value

HIP_train_eval returns a list with the following:

each	tensor(double) - (for gaussian outcomes) tensor of MSEs for each of the q outcomes
comp_val	tensor(double) - MSE for gaussian outcomes, classification accuracy for multiclass outcomes, fraction of deviance explained for poisson/ZIP outcomes
pred	list<tensor(double) - predicted values from fitted model. For multiclass outcomes, single integer to represent class, starting with 0. For poisson/ZIP, predicted means from fitted model
dev_sum	list(tensor(double)) - (for poisson/ZIP outcomes), list with more detailed deviance information

Also returns a histogram plot for predicted vs. true values in each subgroup.

Examples

```
# Generate data
dat_gen_test <- generate_data(test_data=TRUE)

# Get results from select_lambda
res <- select_lambda(dat_gen_test$X, dat_gen_test$Y, c(1,1), 'gaussian', 50,
                     K = 2, num_steps=c(4,4))

# Evaluating training data
train_eval <- HIP_train_eval(dat_gen_test, res, 'search')
```

---

optimize_torch *Main optimization function for continuous outcomes*

Description

This function is used to fit a HIP model given X and Y training data, a set of lambda values, and gamma values. This largely exists as a helper function, as the user will more often use select_lambda or fixed_lambda to fit HIP models (which then access optimize_torch). However, the function is available if the user needs to access the optimization function directly.

Usage

```
optimze_torch(X, Y, lambda_xi, lambda_g, gamma, family, K=NULL, k_thresh=0.2,
update_thresh=10^-5, epsilon=10^-4, max_iter=50, standardize='subgroup',
std_type='scale_center', std_x=TRUE, std_y=TRUE, verbose=FALSE)
```

Arguments

Y	matrix list - list of Y^s matrices containing outcomes
lambda_xi	double - value of lambda_xi in penalty term
lambda_g	double - value of lambda_g in penalty term
gamma	list - indicators of whether to penalize each data view; should be length D
family	string - family of outcome; options are 'gaussian', 'multiclass', 'poisson' or 'zip'
K	int - number of latent components; will use 'select_K_simple' to choose if not provided
k_thresh	double - threshold to use for 'select_K_simple' if K not provided; default = 0.2
update_thresh	double - criteria for convergence in Z, G, Xi, and theta optimization functions; default = 10^-5
epsilon	double - criteria for outer loop convergence; default = 10^-4
max_iter	int - maximum number of outer loop iterations allowed; default = 50
standardize	string - One of "all", "subgroup", or "none"; default = "subgroup"
std_type	string - One of "scale_center", "center", or "norm"; default = "scale_center"
std_x	boolean - indicates whether to standardize X; default = True
std_y	boolean - indicates whether to standardize Y; default = True
verbose	boolean - indicates whether to print additional info during run; default = False
matrix	list - list of X^d,s matrices containing covariates

Value

Returns a list with the possible following elements based on status:

theta	tensor - estimate of theta
beta	tensor - estimate of beta
B	list<tensor> - estimates of each B^d,s
G	list<tensor> - estimates of each G^d
Xi	list<tensor> - estimates of each Xi^d,s
Z	list<tensor> - estimates of each Z^s
Lambda	tuple - values of lambda_xi and lambda_g used
BIC	double - calculated BIC
AIC	double - calculated AIC
pred	double - prediction loss evaluated at final estimates
train_err	nested list - list returned from function to calculate training error
message	string - message with the status of the result; "Converged" if converged successfully, "MAX ITERS" if algorithm reached max_iter without converging.
paths	list - history of losses until convergence
iter	int - number of iterations to converge
iter_time	double - time to find solution in seconds
conv_criterion	double - value of last convergence criterion
std_x	boolean - whether X was standardized
std_y	boolean - whether Y was standardized

Examples

```
# Generating data
dat_gen <- generate_data()

# Run the optimization algorithm given the data, lambdas, and gamma
optimization_out <- optimize_torch(dat_gen$Y, dat_gen$X, 0.5, 0.5, c(1,1), 'gaussian', K=3)
```

`select_lambda` *Fit HIP models across multiple lambda values*

Description

`select_lambda` searches across lambda values (either grid search or random) and fits HIP models given training data. The best model is chosen based on selection criteria such as cross-validation, BIC, or AIC. Results can then be used for model evaluation, prediction, and plotting.

Usage

```
select_lambda(X, Y, gamma, family, topn, ncore=NA, K=NULL, k_thresh=0.2, update_thresh=10^-5,
              epsilon=10^-4, max_iter=50, search="random", xi_range=c(0,2),
              g_range=c(0,2), num_steps=c(8,8), standardize="subgroup",
              std_type="scale_center", std_x=TRUE, std_y=TRUE, verbose=FALSE,
              selection_criterion="eBIC_0", folds=5)
```

Arguments

`X`	matrix list - list of X^d,s matrices containing covariates
`Y`	matrix list - list of Y^s matrices containing outcomes
`gamma`	list - indicators of whether to penalize each data view; should be length D
`family`	string - family of outcome; options are 'gaussian', 'multiclass', 'poisson' or 'zip'
`topn`	int or list - number of variables to retain; different values may be specified for each view using a list of length D
`ncore`	int - number of cores to use in parallel processing; default is half of available cores
`K`	int - number of latent components; will use 'select_K_simple' to choose if not provided
`k_thresh`	• double - threshold to use for 'select_K_simple' if K not provided; default = 0.2
`epsilon`	double - threshold to use for overall convergence criterion; default = 10**-4
`max_iter`	int - maximum number of outer loop iterations; default = 50
`search`	string - what type of search to perform for lambda parameter; default = "random" • "random" - tries a random selection of rand_prop*num_steps lambda values in grid • "grid" - tries all lambda values in grid

xi_range	list - minimum and maximum values to consider for lambda_xi; default = c(0.0, 2.0)
g_range	list - minimum and maximum values to consider for lambda_g; default = c(0.0, 2.0)
num_steps	list - list of two integers; the first is the number of steps for lambda_xi (default = 8) and the second is the number of steps for lambda_g. Together these define the number of steps to use in lambda grid
standardize	string - One of "all", "subgroup", or "none"; default = "subgroup"
std_type	string - One of "scale_center", "center", or "norm"; default = "scale_center"
std_x	boolean - indicates whether to standardize X; default = True
std_y	boolean - indicates whether to standardize Y; default = True
verbose	whether to print additional information during optimization; default = False
selection_criterion	string - criterion to use for selecting the best model; one of 'CV', 'BIC', 'AIC', 'eBIC_0', 'eBIC_5', or 'eBIC_1'. eBIC_0 by default.
folds	int - number of folds to use if CV is selected as the selection criterion; default = 5

Value

The output is quite large, but most items do not need to be accessed directly by the user and instead are accessed by functions such as HIP_train_eval or HIP_pred. First, select_lambda returns a nested list called out which contains the following:

search_results	list - list with results returned from the optimize_torch Python function for each lambda value tried (see below for detailed output information)
total_time	double - time to complete entire search in seconds
xi_range	list - minimum and maximum values considered for lambda_xi
g_range	list - minimum and maximum values considered for lambda_g
num_steps	dict - number of steps used in lambda grid
search	string - type of search performed for selecting lambda parameters

select_lambda also returns the following:

best_index	int - index of the best model chosen by the selection criterion
criterion	string - criterion for model selection: one of 'CV', 'BIC', 'AIC', 'eBIC_0', 'eBIC_5', or 'eBIC_1'. eBIC_0 by default.
topn	int or list - number of variables retained
standardize	string - stores option used to standardize data
family	string - stores family label for outcome data

search_results is a large list containing results from fit models. First, it contains multiple fit models which have the following items:

full	list - results returned from optimize_torch on full data
include	list - list of length D with a 1 indicating the variable was included in subset fit and 0 indicating not included in subset fit
subset	list - results returned from optimize_torch on subset of variables

`full` and `subset` are large lists which contain the following:

`theta`	tensor - estimate of theta
`beta`	tensor - estimate of beta
`B`	list<tensor> - estimates of each B^d,s
`G`	list<tensor> - estimates of each G^d
`Xi`	list<tensor> - estimates of each Xi^d,s
`Z`	list<tensor> - estimates of each Z^s
`Lambda`	tuple - values of lambda_xi and lambda_g used
`BIC`	double - calculated BIC
`AIC`	double - calculated AIC
`pred`	double - prediction loss evaluated at final estimates
`train_err`	nested list - list returned from function to calculate training error
`message`	string - message with the status of the result; "Converged" if converged successfully, "MAX ITERS" if algorithm reached max_iter without converging.
`paths`	list - history of losses until convergence
`iter`	int - number of iterations to converge
`iter_time`	double - time to find solution in seconds
`conv_criterion`	double - value of last convergence criterion
`std_x`	boolean - whether X was standardized
`std_y`	boolean - whether Y was standardized

Examples

```
dat_gen <- generate_data()

res <- select_lambda(dat_gen$X, dat_gen$Y, c(1,1), 'gaussian', 50)
```

`standardize_dat` *Standardizes input data*

Description

Helper function used to standardize testing data from training data.

Usage

```
standardize_dat(
  standardize,
  std_type,
  X = NULL,
  Y = NULL,
  X_train = NULL,
  Y_train = NULL,
  std_x = TRUE,
  std_y = TRUE,
  verbose = TRUE
)
```

Arguments

standardize	string - One of "all", "subgroup", or "none"
std_type	string - One of "scale_center", "center", or "norm"
X	matrix list - list of X^d,s matrices containing covariates
Y	matrix list - list of Y^s matrices containing outcomes
X_train	matrix list - list of X^d,s matrices whose mean and sd will be used to standardize X
Y_train	matrix list - list of Y^s matrices whose mean andd sd will be used to standardize Y
std_x	boolean - indicates whether to standardize X; default = True
std_y	boolean - indicates whether to standardize Y; default = True

Value

Returns a nested list with the following elements:

X	matrix list - standardized X^d,s if standardization was requested
Y	matrix list - standardized Y^s if standardization was requested

---

variable_plots *Variable importance plots*

Description

Outputs plots showing the highest ranking variables contributing to model performance. Plots are shown for each data view and subgroup.

Usage

```
variable_plots(data, res, fix_or_search, top_plotted=15, compact=TRUE,
output_table=FALSE, file_path=NA)
```

Arguments

data	data object from generate_data or format_data
res	output from select_lambda or fixed_lambda
fix_or_search	string - 'fixed' if results obtained from fixed_lambda, 'search' otherwise
top_plotted	int - number of variables to include in each variable importance plot; 15 by default
compact	boolean - if TRUE (default), only plots top_plotted variables; if set to FALSE, plots as many as possible
output_table	boolean - set to TRUE to write variable importance results to an Excel file; FALSE by default
file_path	string - if output_table is TRUE, you may also specify a file path and file name; make sure to add the .xlsx extension to your file name. By default, it will be written to your current working directory with the name "variable_importance_table.xlsx"

Value

Returns the variable importance plot as a ggplot object

Examples

```
# Generate data
dat_gen <- generate_data()

# Get results from select_lambda
res <- select_lambda(dat_gen$X, dat_gen$Y, c(1,1), 'gaussian', 50)

# Variable importance plots
variable_plots(dat_gen, res, 'search', output_table=TRUE)
```

Index