Package ‘sMTL’

February 6, 2023

Title Sparse Multi-Task Learning

Version 0.1.0

Description Implements L0-constrained Multi-Task Learning and domain generalization algorithms. The algorithms are coded in Julia allowing for fast implementations of the coordinate descent and local combinatorial search algorithms. For more details, see a preprint of the paper: Loewinger et al., (2022) <arXiv:2212.08697>.

URL `https://github.com/gloewing/sMTL`,
`https://rpubs.com/gloewinger/996629`

BugReports `https://github.com/gloewing/sMTL/issues`

Maintainer Gabriel Loewinger <`gloewinger@gmail.com`>

Depends R (>= 3.5.0)

License MIT + file LICENSE

Encoding UTF-8

Imports glmnet, JuliaCall, JuliaConnectoR, caret, dplyr

RoxygenNote 7.2.1

Suggests knitr, rmarkdown

NeedsCompilation no

Author Gabriel Loewinger [aut, cre] (<`https://orcid.org/0000-0002-0755-8520`>),
Kayhan Behdin [aut],
Giovanni Parmigiani [aut],
Rahul Mazumder [aut],
National Science Foundation Grant DMS1810829 [fnd],
National Science Foundation Grant DMS2113707 [fnd],
National Science Foundation Grant NSF-IIS1718258, [fnd],
Office of Naval Research Grant ONR N000142112841 [fnd],
National Institute on Drug Abuse (NIH) Grant F31DA052153 [fnd]

Repository CRAN

Date/Publication 2023-02-06 11:20:02 UTC

R topics documented:

---

cv.smtl *cv.smtl: cross-validation function*

Description

cv.smtl: cross-validation function

Usage

```
cv.smtl(
  y,
  X,
  study = NA,
  grid = NA,
  nfolds = NA,
  commonSupp = FALSE,
  multiTask = TRUE,
  lambda_1 = TRUE,
  lambda_2 = FALSE,
  lambda_z = TRUE,
  maxIter = 2500,
  LocSrch_skip = 1,
  LocSrch_maxIter = 10,
  messageInd = FALSE,
  independent.regs = FALSE
)
```

Arguments

`y` A numeric outcome vector or matrix (for multi-label problems)

`X` A design (feature) matrix

`study` An integer vector specifying the task ID

`grid` A dataframe with column names "s", "lambda_1", "lambda_2" and "lambda_z" (if commonSupp = FALSE) with tuning values

`nfolds` An integer specifying number of CV folds

`commonSupp` A boolean specifying whether the task models should have the same support

`multiTask` A boolean only used if study/task indices are provided: used to distinguish between a Multi-Task Learning Tuning (TRUE) or Domain Generalization Tuning (FALSE)

`lambda_1` An optional boolean: if a grid is not provided, then set to TRUE if you want an automatic grid to be generated with non-zero values for this hyperparameter

`lambda_2` An optional boolean: if a grid is not provided, then set to TRUE if you want an automatic grid to be generated with non-zero values for this hyperparameter

`lambda_z` An optional boolean: if a grid is not provided, then set to TRUE if you want an automatic grid to be generated with non-zero values for this hyperparameter

`maxIter` An integer specifying the maximum number of coordinate descent iterations

`LocSrch_skip` An integer specifying whether to use local search at every tuning value (set to 1), every other value (set to 2), every third (set to 3),...

`LocSrch_maxIter`
An integer specifying the maximum number of local search iterations

`messageInd` A boolean (verbose) of whether to print messages

`independent.regs`
A boolean of whether models are completely indpendent (only set to TRUE for benchmarks)

Value

A list

Examples

```
#####################################################################################
##### simulate data
#####################################################################################
set.seed(1) # fix the seed to get a reproducible result
K <- 4 # number of datasets
p <- 100 # covariate dimension
s <- 5 # support size
q <- 7 # size of subset of covariates that can be non-zero for any task
n_k <- 50 # task sample size
N <- n_k * p # full dataset samplesize
X <- matrix( rnorm(N * p), nrow = N, ncol=p) # full design matrix
```

```
B <- matrix(1 + rnorm(K * (p+1) ), nrow = p + 1, ncol = K) # betas before making sparse
Z <- matrix(0, nrow = p, ncol = K) # matrix of supports
y <- vector(length = N) # outcome vector

# randomly sample support to make betas sparse
for(j in 1:K)     Z[1:q, j] <- sample( c( rep(1,s), rep(0, q - s) ), q, replace = FALSE )
B[-1,] <- B[-1,] * Z # make betas sparse and ensure all models have an intercept

task <- rep(1:K, each = n_k) # vector of task labels (indices)

# iterate through and make each task specific dataset
for(j in 1:K){
    indx <- which(task == j) # indices of task
    e <- rnorm(n_k)
    y[indx] <- B[1, j] + X[indx,] %*% B[-1,j] + e
    }

colnames(B) <- paste0("beta_", 1:K)
rownames(B) <- paste0("X_", 1:(p+1))

print("Betas")
print(round(B[1:8,],2))

    ###########################
    # custom tuning grid
    ###########################
    grid <- data.frame(s = c(4, 4, 5, 5),
                  lambda_1 = c(0.01, 0.1, 0.01, 0.1),
                  lambda_2 = rep(0, 4),
                  lambda_z = c(0.01, 0.1, 0.01, 0.1))

    #################################################
    # cross validation with custom tuning grid
    ##################################################
## Not run:

if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## The examples are quite time consuming
## Do initiation for and automatic installation if necessary

    tn <- cv.smtl(y = y,
                  X = X,
                  study = task,
                  commonSupp = FALSE,
                  grid = grid,
                  nfolds = 5,
                  multiTask = FALSE)

     # model fitting
     mod <- sMTL::smtl(y = y,
                   X = X,
                   study = task,
                   s = tn$best.1se$s,
                   commonSupp = TRUE,
                   lambda_1 = tn$best.1se$lambda_1,
                   lambda_z = tn$best.1se$lambda_z)

    ######################################################
    # cross validation with automatically generated grid
    #######################################################
    tn <- cv.smtl(y = y,
                  X = X,
                  study = task,
                  commonSupp = FALSE,
                  lambda_1 = TRUE,
                  lambda_w = FALSE,
                  lambda_z = TRUE,
                  nfolds = 5,
                  multiTask = FALSE)

     # model fitting
     mod <- sMTL::smtl(y = y,
                   X = X,
                   study = task,
                   s = tn$best.1se$s,
                   commonSupp = TRUE,
                   lambda_1 = tn$best.1se$lambda_1,
                   lambda_z = tn$best.1se$lambda_z)

     print(round(mod$beta[1:8,],2))
                   }

## End(Not run)
```

grid.gen

grid.gen: generate grid for cross-validation function. For internal package use only.

Description

grid.gen: generate grid for cross-validation function. For internal package use only.

Usage

```
grid.gen(
  y,
  p,
  study = NA,
  lambda_1 = TRUE,
  lambda_2 = FALSE,
  lambda_z = TRUE,
  commonSupp = FALSE,
  multiTask = TRUE
)
```

Arguments

`y`	A numeric vector or matrix of outcomes
`p`	An integer of covariate dimension
`study`	An integer vector of task IDs
`lambda_1`	A boolean
`lambda_2`	A boolean
`lambda_z`	A boolean
`commonSupp`	A boolean
`multiTask`	A boolean

Value

A dataframe

`maxEigen` *maxEigen: maximum eigenvalue wrapper for Julia TSVD package. internal package use only*

Description

maxEigen: maximum eigenvalue wrapper for Julia TSVD package. internal package use only

Usage

```
maxEigen(X, intercept = TRUE)
```

Arguments

`X`	A matrix.
`intercept`	A boolean.

Value

A numeric scalar of the maximum eigenvalue of provided matrix, X.

method_nm *methods names: give name for printing. Internal package use only.*

Description

methods names: give name for printing. Internal package use only.

Usage

```
method_nm(method, multiLabel = TRUE)
```

Arguments

`method`	A string
`multiLabel`	A boolean

Value

A string indicating what type of multi-task learning problem is being fit.

multiTaskRmse *multiTaskRmse: RMSE for multi-task problems (averaged across tasks)*

Description

multiTaskRmse: RMSE for multi-task problems (averaged across tasks)

Usage

```
multiTaskRmse(data, beta)
```

Arguments

`data`	A matrix including outcome vector/matrix and design matrix to test RMSE on
`beta`	A matrix of estimated beta coefficients where each task is in a different column

Value

Returns a scalar of average (across tasks) RMSE for predictions on data provided

`multiTaskRmse_MT` *multiTaskRmse: calculate average (across tasks) RMSE for multi-label prediction problems*

Description

multiTaskRmse: calculate average (across tasks) RMSE for multi-label prediction problems

Usage

```
multiTaskRmse_MT(data, K = NA, beta)
```

Arguments

`data`	A matrix including outcome vector/matrix and design matrix to test RMSE on
`K`	An integer of number of studies/tasks
`beta`	A matrix of estimated beta coefficients where each task is in a different column

Value

Returns a scalar of average (across tasks) RMSE for predictions on data provided

`predict` *predict: predict on smtl model object*

Description

predict: predict on smtl model object

Usage

```
predict(model, X, lambda_1 = NA, lambda_2 = NA, lambda_z = NA, stack = FALSE)
```

Arguments

`model`	An sMTL model object returned from the smtl() function
`X`	A matrix of deatures
`lambda_1`	A optional numeric scalar specifying which lambda_1 to use for prediction. Only needed if the model object is fit on a path (multiple hyperparameterr values)
`lambda_2`	A optional numeric scalar specifying which lambda_2 to use for prediction. Only needed if the model object is fit on a path (multiple hyperparameterr values)

`lambda_z` A optional numeric scalar specifying which lambda_2 to use for prediction. Only needed if the model object is fit on a path (multiple hyperparameterr values)

`stack` An optional boolean specifying whether to calculate and apply stacking weights (only for Domain Generalization problems).

Value

A matrix of task-specific predictions for multi-task/multi-label or for Domain Generalization problems, average and multi-study stacking predictions.

Examples

```
#####################################################################################
##### First Time Loading, Julia is Installed and Julia Path is Known ######
#####################################################################################
# fit model
## Not run:

if (identical(Sys.getenv(”AUTO_JULIA_INSTALL”), ”true”)) { ## The examples are quite time consuming
## Do initiation for and automatic installation if necessary
mod <- smtl(y = y,
            X = X,
            study = task,
            s = 5,
            commonSupp = FALSE,
            lambda_1 = c(0.1, 0.2, 0.3),
            lambda_z = c(0.01, 0.05, 0.1))

# make predictions
preds <- sMTL::predict.smtl(model = mod,
                       X = X,
                       lambda_1 = 0.1,
                       lambda_z = 0.01) }

## End(Not run)
```

reName_cv *reName_cv: rename output from CV. For internal package use only.*

Description

reName_cv: rename output from CV. For internal package use only.

Usage

```
reName_cv(x)
```

Arguments

`x` A list (S3 class) supplied from internal sMTL functions

Value

A list (S3 class) with elements renamed.

`best` A list (S3 class) with hyperparameters that achieve lowest average RMSE.

`best.1se` A list (S3 class) with hyperparameters associated with lowest sparsity level within 1 standard deviation of hyperparameters that achieve lowest average RMSE.

`lambda_1` Numeric hyperparameter for L2 (ridge penalty).

`lambda_2` Numeric hyperparameter for betabar penalty.

`rho` Integer specifying sparsity level (s).

`rhoScale` *rhoScale: scale lambda_z depending on magnitude. For internal package use only.*

Description

rhoScale: scale lambda_z depending on magnitude. For internal package use only.

Usage

```
rhoScale(K, p, rhoVec, itrs = 10000)
```

Arguments

`K` An integer - number of tasks

`p` An integer - dimension of covariates

`rhoVec` A vector of integers

`itrs` An integer

Value

A matrix or datafame with lambda_z hyperparameter scaled appropriately depending on sparsity level.

seReturn — *seReturn: find smallest rho within 1 se of smallest cv error. For internal package use.*

Description

seReturn: find smallest rho within 1 se of smallest cv error. For internal package use.

Usage

```
seReturn(x)
```

Arguments

x	dataframe

Value

Returns a dataframe that includes summary statistics to choose the best sparsity level (s) according to the 1-standard deviation rule.

smtl — *smtl: make model-fitting function*

Description

smtl: make model-fitting function

Usage

```
smtl(
  y,
  X,
  study = NA,
  s,
  commonSupp = FALSE,
  warmStart = TRUE,
  lambda_1 = 0,
  lambda_2 = 0,
  lambda_z = 0,
  scale = TRUE,
  maxIter = 10000,
  LocSrch_maxIter = 50,
  messageInd = TRUE,
  model = TRUE,
  independent.regs = FALSE
)
```

Arguments

y
A numeric outcome vector (for multi-task/domain generalization problems) or a numeric outcome matrix (for multi-label problems)

X
A matrix of covariates

study
A vector of integers specifying task (or study/domain) ID. This should be set to NA for Multi-Label problems, but is required for Multi-Task and Domain Generalization problems.

s
An integer specifying the sparsity level

commonSupp
A boolean specifying whether to constrain solutions to have a common support

warmStart
A boolean specifying whether a warm start model is fit internally before the final model. Warm starts improve solution quality but will be slower.

lambda_1
A numeric vector of ridge penalty hyperparameter values

lambda_2
A numeric vector of betaBar (to borrow strength across coefficient values) penalty hperparameter values

lambda_z
A numeric vector zBar (to borrow strength across coefficient supports) penalty hperparameter values

scale
A boolean specifying whether to center and scale covariates before model fitting (either way coefficient estimates are returned on original scale before centering/scaling)

maxIter
An integer specifying the maximum number of coordinate descent iterations before

LocSrch_maxIter
An integer specifying the number of maximum local search iterations

messageInd
A boolean specifying whether to include messages (verbose)

model
A boolean indicating whether to return design matrix and outcome vector

independent.regs
A boolean specifying whether to fit independent regressions (instead of multi-task). This ensures there is NO information sharing via active sets or penalties

Value

A list (object of S3 class).

beta
Matrix with coefficient estimates where column j are estimates from task j.

reg_type
String specifying whether model is "multiStudy" denoting that there is a separate design matrix for each task, "multiLabel" where the design matrix is the same across tasks and "L0" indicating a single-task regression.

K
Integer that indicates number of tasks.

s
An integer that indicates sparsity level.

commonSupp
Boolean indicating of supports are common across tasks.

warmStart
A Boolean indicating whether to fit a MTL model as a warm start.

grid
A dataframe including grid of hyperparameters that model is fit on.

`maxIter` An integer specifying the maximum number of iterations of block CD.

`LocSrch_maxIter`
An integer specify the maximum number of iterations of local search.

`independent.regs`
A boolean indicating whether to make each task independent of each other (no shared active sets).

`AS_multiplier` An integer specifying the active set multiplier.

`X_train` A Matrix: the design matrix (row concatenated across tasks).

`y_train` The outcome vector or matrix.

Examples

```
## Not run:

if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## The examples are quite time consuming
## Do initiation for and automatic installation if necessary

# load package
library(sMTL)
smtl_setup()

######################################################################################
##### simulate data
######################################################################################
set.seed(1) # fix the seed to get a reproducible result
K <- 4 # number of datasets
p <- 100 # covariate dimension
s <- 5 # support size
q <- 7 # size of subset of covariates that can be non-zero for any task
n_k <- 50 # task sample size
N <- n_k * p # full dataset samplesize
X <- matrix( rnorm(N * p), nrow = N, ncol=p) # full design matrix
B <- matrix(1 + rnorm(K * (p+1) ), nrow = p + 1, ncol = K) # betas before making sparse
Z <- matrix(0, nrow = p, ncol = K) # matrix of supports
y <- vector(length = N) # outcome vector

# randomly sample support to make betas sparse
for(j in 1:K)     Z[1:q, j] <- sample( c( rep(1,s), rep(0, q - s) ), q, replace = FALSE )
B[-1,] <- B[-1,] * Z # make betas sparse and ensure all models have an intercept

task <- rep(1:K, each = n_k) # vector of task labels (indices)

# iterate through and make each task specific dataset
for(j in 1:K){
    indx <- which(task == j) # indices of task
    e <- rnorm(n_k)
    y[indx] <- B[1, j] + X[indx,] %*% B[-1,j] + e
    }
    colnames(B) <- paste0("beta_", 1:K)
    rownames(B) <- paste0("X_", 1:(p+1))
    print("Betas")
    print(round(B[1:8,],2))

#####################################################################################
##### fit Multi-Task Learning Model for Heterogeneous Support
#####################################################################################

    mod <- sMTL::smtl(y = y,
                      X = X,
                      study = task,
                      s = 5,
                      commonSupp = FALSE,
                      lambda_1 = 0.001,
                      lambda_2 = 0,
                      lambda_z = 0.25)

    print(round(mod$beta[1:8,],2))

    # make predictions
    preds <- sMTL::predict(model = mod, X = X[1:5,])

#####################################################################################
##### fit Multi-Task Learning Model for Common Support
#####################################################################################
    library(sMTL)
    sMTL::smtl_setup(path = "/Applications/Julia-1.5.app/Contents/Resources/julia/bin")
    mod <- sMTL::smtl(y = y,
                      X = X,
                      study = task,
                      s = 5,
                      commonSupp = TRUE,
                      lambda_1 = 0.001,
                      lambda_2 = 0.5)

    print(round(mod$beta[1:8,],2))
    }

## End(Not run)
```

smtl_setup *smtl_setup: setup Julia path and/or install Julia or Julia packages using functions based on external package JuliaCall::julia_setup().*

Description

smtl_setup: setup Julia path and/or install Julia or Julia packages using functions based on external package JuliaCall::julia_setup().

Usage

```
smtl_setup(path = NULL, installJulia = FALSE, installPackages = FALSE)
```

Arguments

`path` A string

`installJulia` A boolean.

`installPackages` A boolean.

Value

A message indicating either Julia language or package installation status or the path of Julia Binary on your computer. See vignette if you have problems specifying the path of Julia binary correctly.

Examples

```
## Not run:

if (identical(Sys.getenv("AUTO_JULIA_INSTALL"), "true")) { ## The examples are quite time consuming
## Do initiation for and automatic installation if necessary
##################################################################
# First Time Loading, Julia is Installed and Julia Path is Known
##################################################################
smtl_setup(path = "/Applications/Julia-1.5.app/Contents/Resources/julia/bin",
           installJulia = FALSE,
           installPackages = FALSE)

#####################################################################################
# If you have run smtl_setup() before, then path specification shouldn't be necessary
#####################################################################################
smtl_setup(path = NULL, installJulia = FALSE, installPackages = FALSE)

#####################################################################################
##### First Time Loading, Julia is Not Installed   ######
#####################################################################################
smtl_setup(path = NULL, installJulia = TRUE, installPackages = FALSE)

#####################################################################################
##### First Time Loading, Julia is Installed But Packages NEED INSTALLATION  ######
#####################################################################################
smtl_setup(path = "/Applications/Julia-1.5.app/Contents/Resources/julia/bin",
           installJulia = TRUE,
           installPackages = TRUE)

           }


## End(Not run)
```

sparseCV *sparseCV: cross-validation functions. For internal package use only.*

Description

sparseCV: cross-validation functions. For internal package use only.

Usage

```
sparseCV(
  data,
  tune.grid,
  hoso = "hoso",
  method = "L0",
  nfolds = "K",
  juliaFnPath = NA,
  messageInd = FALSE,
  LSitr = 50,
  LSspc = 1,
  maxIter = 2500
)
```

Arguments

`data`	Matrix with outcome and design matrix
`tune.grid`	A data.frame of tuning values
`hoso`	String specifying tuning type
`method`	Sting specifying regression method
`nfolds`	String or integer specifying number of folds
`juliaFnPath`	String specifying path to Julia binary
`messageInd`	Boolean for message printing
`LSitr`	Integer specifying do <LSitr> local search iterations on parameter values where we do actually do LS; NA does no local search
`LSspc`	Integer specifying number of hyperparameters to conduct local search: conduct local search every <LSspc>^th iteration. NA does no local search
`maxIter`	Integer specifying max iterations of coordinate descent

Value

A list (S3 class) with elements used for cross validation.

`best`	A dataframe with the hyperparameters associated with the best prediction performance and summary statistics of performance.
`best.1se`	A dataframe including optimal hyperparameters according to 1-standard deviation rule.

`rmse`	A dataframe with prediction performance for hyperparamters in tuning grid for all folds.
`avg`	A dataframe with average performance at each of the hyperparameters in tuning grid (averaged across tasks).

`sparseCV_MT` *sparseCV_MT: internal cross-validation functions. For internal package use only.*

Description

sparseCV_MT: internal cross-validation functions. For internal package use only.

Usage

```
sparseCV_MT(
  data,
  tune.grid,
  hoso = "hoso",
  method = "L0",
  nfolds = "K",
  juliaFnPath = NA,
  messageInd = FALSE,
  LSitr = 50,
  LSspc = 1,
  maxIter = 2500
)
```

Arguments

`data`	Matrix with outcome and design matrix
`tune.grid`	A data.frame of tuning values
`hoso`	String specifying tuning type
`method`	Sting specifying regression method
`nfolds`	String or integer specifying number of folds
`juliaFnPath`	String specifying path to Julia binary
`messageInd`	Boolean for message printing
`LSitr`	Integer specifying do <LSitr> local search iterations on parameter values where we do actually do LS; NA does no local search
`LSspc`	Integer specifying number of hyperparameters to conduct local search: conduct local search every <LSspc>^th iteration. NA does no local search
`maxIter`	Integer specifying max iterations of coordinate descent

Value

A list (S3 class) with elements used for cross validation.

`best`	A dataframe with the hyperparameters associated with the best prediction performance and summary statistics of performance.
`best.1se`	A dataframe including optimal hyperparameters according to 1-standard deviation rule.
`rmse`	A dataframe with prediction performance for hyperparamters in tuning grid for all folds.
`avg`	A dataframe with average performance at each of the hyperparameters in tuning grid (averaged across tasks).

sparseL0Tn_iht — *sparseCV_L0: cross-validation functions. For internal package use only.*

Description

sparseCV_L0: cross-validation functions. For internal package use only.

Usage

```
sparseL0Tn_iht(
  data,
  tune.grid,
  hoso = "hoso",
  nfolds = "K",
  juliaFnPath = "/Users/gabeloewinger/Desktop/Research Final/Sparse Multi-Study/",
  trainingStudy = NA,
  messageInd = FALSE,
  LSitr = 50,
  LSspc = 1,
  maxIter = 2500
)
```

Arguments

`data`	Matrix with outcome and design matrix
`tune.grid`	A data.frame of tuning values
`hoso`	String specifying tuning type
`nfolds`	String or integer specifying number of folds
`juliaFnPath`	String specifying path to Julia binary
`trainingStudy`	Integer specifying index of training study
`messageInd`	Boolean for message printing

`LSitr`	Integer specifying do <LSitr> local search iterations on parameter values where we do actually do LS; NA does no local search
`LSspc`	Integer specifying number of hyperparameters to conduct local search: conduct local search every <LSspc>^th iteration. NA does no local search
`maxIter`	Integer specifying max iterations of coordinate descent

Value

A list (S3 class) with elements used for cross validation.

`best`	A dataframe with the hyperparameters associated with the best prediction performance and summary statistics of performance.
`best.1se`	A dataframe including optimal hyperparameters according to 1-standard deviation rule.
`rmse`	A dataframe with prediction performance for hyperparamters in tuning grid for all folds.
`avg`	A dataframe with average performance at each of the hyperparameters in tuning grid (averaged across tasks).

`tuneZscale` *tuneZscale: scale lambda_z depending on magnitude. For internal package use only.*

Description

tuneZscale: scale lambda_z depending on magnitude. For internal package use only.

Usage

```
tuneZscale(tune.grid, rhoScale)
```

Arguments

`tune.grid`	A dataframe
`rhoScale`	A dataframe

Value

A dataframe that includes tuning grid with the lambda_z hyperparameter re-scaled appropriately for sparsity levels (s).

Index