Package ‘vennLasso’

June 3, 2020

Type Package
Title Variable Selection for Heterogeneous Populations
Version 0.1.6
Description Provides variable selection and estimation routines for models with main effects stratified on multiple binary factors. The ‘vennLasso’ package is an implementation of the method introduced in Huling, et al. (2017) <doi:10.1111/biom.12769>.

URL https://github.com/jaredhuling/vennLasso
BugReports https://github.com/jaredhuling/vennLasso/issues
License GPL (>= 2)
Encoding UTF-8
LazyData TRUE
Depends R (>= 3.2.0)
Imports Rcpp (>= 0.11.0), foreach, survival, MASS, Matrix, VennDiagram, visNetwork, igraph, methods
LinkingTo Rcpp, RcppEigen, RcppNumerical
RoxygenNote 7.1.0
Suggests knitr, rmarkdown
VignetteBuilder knitr
NeedsCompilation yes
Author Jared Huling [aut, cre] (<https://orcid.org/0000-0003-0670-4845>), Muxuan Liang [ctb], Yixuan Qiu [cph], Gael Guennebaud [cph], Ray Gardner [cph], Jitse Niesen [cph]
Maintainer Jared Huling <jaredhuling@gmail.com>
Repository CRAN
Date/Publication 2020-06-03 08:50:02 UTC
cv.vennLasso

Cross Validation for the vennLasso

Description
Cross Validation for the vennLasso

Usage
cv.vennLasso(
x,
y,
groups,
lambda = NULL,
compute.se = FALSE,
conf.int = NULL,
type.measure = c("mse", "deviance", "class", "auc", "mae", "brier"),
nfolds = 10,
foldid,
grouped = TRUE,
keep = FALSE,
parallel = FALSE,
...)

Arguments
x input matrix or SparseMatrix of dimension nob x nvars. Each row is an observation, each column corresponds to a covariate
y numeric response vector of length nob
groups  A list of length equal to the number of groups containing vectors of integers indicating the variable IDs for each group. For example, `groups=list(c(1,2), c(2,3), c(3,4,5))` specifies that Group 1 contains variables 1 and 2, Group 2 contains variables 2 and 3, and Group 3 contains variables 3, 4, and 5. Can also be a matrix of 0s and 1s with the number of columns equal to the number of groups and the number of rows equal to the number of variables. A value of 1 in row i and column j indicates that variable i is in group j and 0 indicates that variable i is not in group j.

lambda  A user-specified sequence of lambda values. Left unspecified, the a sequence of lambda values is automatically computed, ranging uniformly on the log scale over the relevant range of lambda values.

compute.se  logical flag. If TRUE, standard errors will be computed, otherwise if FALSE they will not

conf.int  value between 0 and 1 indicating the level of the confidence intervals to be computed. For example if conf.int = 0.95, 95 percent confidence intervals will be computed.

type.measure  One of c("mse", "deviance", "class", "auc", "mae", "brier") indicating measure to evaluate for cross-validation. The default is type.measure = "deviance", which uses squared-error for gaussian models (a.k.a type.measure = "mse" there), deviance for logistic regression. type.measure = "class" applies to binomial only. type.measure = "auc" is for two-class logistic regression only. type.measure = "mse" or type.measure = "mae" (mean absolute error) can be used by all models; they measure the deviation from the fitted mean to the response. type.measure = "brier" is for models with family = "coxph" and will compute the Brier score.

nfolds  number of folds for cross-validation. default is 10. 3 is smallest value allowed.

foldid  an optional vector of values between 1 and nfold specifying which fold each observation belongs to.

grouped  Like in glmnet, this is an experimental argument, with default TRUE, and can be ignored by most users. For all models, this refers to computing nfolds separate statistics, and then using their mean and estimated standard error to describe the CV curve. If grouped = FALSE, an error matrix is built up at the observation level from the predictions from the nfold fits, and then summarized (does not apply to type.measure = "auc").

keep  If keep = TRUE, a prevalidated list of array is returned containing fitted values for each observation and each value of lambda for each model. This means these fits are computed with this observation and the rest of its fold omitted. The fold vector is also returned. Default is keep = FALSE

parallel  If TRUE, use parallel foreach to fit each fold. Must register parallel before hand, such as doMC.

...  parameters to be passed to vennLasso

Value

An object with S3 class "cv.vennLasso"
Examples

library(Matrix)

set.seed(123)
n.obs <- 150
n.vars <- 25

true.beta.mat <- array(NA, dim = c(3, n.vars))
true.beta.mat[1,] <- c(-0.5, -1, 0, 2, rep(0, n.vars - 5))
true.beta.mat[2,] <- c(0.5, 0.5, -0.5, -0.5, 1, -1, rep(0, n.vars - 6))
true.beta.mat[3,] <- c(0, 0, 1, 1, -1, rep(0, n.vars - 5))
rownames(true.beta.mat) <- c("1,0", "1,1", "0,1")
true.beta <- as.vector(t(true.beta.mat))

x.sub1 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub2 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub3 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x <- as.matrix(rbind(x.sub1, x.sub2, x.sub3))

conditions <- as.matrix(cbind(c(rep(1, 2 * n.obs), rep(0, n.obs)),
                               c(rep(0, n.obs), rep(1, 2 * n.obs))))

y <- rnorm(n.obs * 3, sd = 3) + drop(as.matrix(bdiag(x.sub1, x.sub2, x.sub3)) %*% true.beta)

fit <- cv.vennLasso(x = x, y = y, groups = conditions, nfolds = 3)

fitted.coef <- predict(fit$vennLasso.fit, type = "coefficients", s = fit$lambda.min)

(true.coef <- true.beta.mat[match(dimnames(fit$vennLasso.fit$beta)[[1]],
                                  rownames(true.beta.mat))])

round(fitted.coef, 2)
## effects smaller for logistic regression
## Not run:
true.beta.mat <- true.beta.mat / 2
true.beta <- true.beta / 2
# logistic regression example#

y <- rbinom(n.obs * 3, 1, 
            prob = 1 / (1 + exp(-drop(as.matrix(bdiag(x.sub1, x.sub2, x.sub3)) %*% true.beta))))

bfit <- cv.vennLasso(x = x, y = y, groups = conditions, family = "binomial",
                     nfolds = 3)

fitted.coef <- predict(bfit$vennLasso.fit, type = "coefficients", s = bfit$lambda.min)

(true.coef <- true.beta.mat[match(dimnames(bfit$vennLasso.fit$beta)[[1]],
                                   rownames(true.beta.mat))])

round(fitted.coef, 2)
## End(Not run)
**estimate.hier.sparsity.param**

function to estimate the hierarchical sparsity parameter for a desired level of sparsity for simulated hierarchical coefficients

**Description**

function to estimate the hierarchical sparsity parameter for a desired level of sparsity for simulated hierarchical coefficients

**Usage**

```r
estimate.hier.sparsity.param(
  ncats,
  nvars,
  avg.hier.zeros = 0.3,
  nsims = 150,
  effect.size.max = 0.5,
  misspecification.prop = 0
)
```

**Arguments**

- `ncats`: number of categories to stratify on
- `nvars`: number of variables
- `avg.hier.zeros`: desired percent of zero variables among the variables with hierarchical zero patterns.
- `nsims`: number of simulations to estimate the average sparsity. A larger number will be more accurate but take much longer.
- `effect.size.max`: maximum magnitude of the true effect sizes
- `misspecification.prop`: proportion of variables with hierarchical missingness misspecified

**Examples**

```r
set.seed(123)

# estimate hier.sparsity.param for 0.15 total proportion of nonzero variables
# among vars with hierarchical zero patterns
## Not run:
  hsp <- estimate.hier.sparsity.param(ncats = 3, nvars = 25, avg.hier.zeros = 0.15, nsims = 100)
## End(Not run)

# the above results in the following value
hsp <- 0.6341772
```
# check that this does indeed achieve the desired level of sparsity
mean(replicate(100, mean(genHierSparseBeta(ncats = 3, nvars = 25, hier.sparsity.param = hsp) != 0) ))

sparseBeta <- genHierSparseBeta(ncats = 3, nvars = 25, hier.sparsity.param = hsp)

## Not run:
hsp2 <- estimate.hier.sparsity.param(ncats = 2, nvars = 100, avg.hier.zeros = 0.30, nsims = 50) # 0.5778425
hsp3 <- estimate.hier.sparsity.param(ncats = 3, nvars = 100, avg.hier.zeros = 0.30, nsims = 50) # 0.4336312
hsp4 <- estimate.hier.sparsity.param(ncats = 4, nvars = 100, avg.hier.zeros = 0.30, nsims = 50) # 0.2670061
hsp5 <- estimate.hier.sparsity.param(ncats = 5, nvars = 100, avg.hier.zeros = 0.30, nsims = 50) # 0.146682

## End(Not run)
# 0.07551241 for hsp6

genHierSparseBeta  
function to generate coefficient matrix with hierarchical sparsity

**Description**

function to generate coefficient matrix with hierarchical sparsity

**Usage**

```r
genHierSparseBeta(
  ncats,
  nvars,
  hier.sparsity.param = 0.5,
  avg.hier.zeros = NULL,
  effect.size.max = 0.5,
  misspecification.prop = 0
)
```

**Arguments**

- `ncats` number of categories to stratify on
- `nvars` number of variables
- `hier.sparsity.param` parameter between 0 and 1 which determines how much hierarchical sparsity there is. To achieve a desired total level of sparsity among the variables with hierarchical sparsity, this parameter can be estimated using the function 'estimate.hier.sparsity.param'
genHierSparseData

function to generate data with hierarchical sparsity

Description

function to generate data with hierarchical sparsity

Usage

genHierSparseData(
  ncats,
  nvars,
  nobs,
  nobs.test = 100,
  hier.sparsity.param = 0.5,
  avg.hier.zeros = NULL,
  prop.zero.vars = 0.5,
  effect.size.max = 0.5,
  misspecification.prop = 0,
  family = c("gaussian", "binomial", "coxph"),
  sd = 1,
  snr = NULL,  

Examples

set.seed(123)

# estimate hier.sparsity.param for 0.15 total proportion of nonzero variables
# among vars with hierarchical zero patterns
# NOT RUN: Takes a long time
# hsp <- estimate.hier.sparsity.param(ncats = 3, nvars = 25, avg.hier.zeros = 0.15, nsims = 100)
# the above results in the following value
hsp <- 0.6341772

# check that this does indeed achieve the desired level of sparsity
mean(replicate(100, mean(genHierSparseBeta(ncats = 3, nvars = 25, hier.sparsity.param = hsp) != 0)))
sparseBeta <- genHierSparseBeta(ncats = 3, nvars = 25, hier.sparsity.param = hsp)
beta = NULL,
    tau = 10,
    covar = 0
)

Arguments

ncats       number of categories to stratify on
nvars       number of variables
nobs        number of observations per strata to simulate
nobs.test   number of independent test observations per strata to simulate
hier.sparsity.param       parameter between 0 and 1 which determines how much hierarchical sparsity there is. To achieve a desired total level of sparsity among the variables with hierarchical sparsity, this parameter can be estimated using the function `estimate.hier.sparsity.param`

avg.hier.zeros    desired percent of zero variables among the variables with hierarchical zero patterns. If this is specified, it will override the given hier.sparsity.param value and estimate it. This takes a while

prop.zero.vars    proportion of all variables that will be zero across all strata
effect.size.max    maximum magnitude of the true effect sizes
misspecification.prop    proportion of variables with hierarchical missingness misspecified

family       family for the response variable
sd           standard deviation for gaussian simulations
snr          signal-to-noise ratio (only used for family = "gaussian")
beta         a matrix of true beta values. If given, then no beta will be created and data will be simulated from the given beta
tau          rate parameter for rexp() for generating time-to-event outcomes
covar        scalar, pairwise covariance term for covariates

Examples

set.seed(123)

dat.sim <- genHierSparseData(ncats = 3, nvars = 100, nobs = 200)

# estimate hier.sparsity.param for 0.15 total proportion of nonzero variables
# among vars with hierarchical zero patterns
## Not run:
hsp <- estimate.hier.sparsity.param(ncats = 3, nvars = 50, avg.hier.zeros = 0.15, nsims = 100)

## End(Not run)
# the above results in the following value
hsp <- 0.6270698
# check that this does indeed achieve the desired level of sparsity
mean(replicate(50, mean(genHierSparseBeta(ncats = 3,
nvars = 50, hier.sparsity.param = hsp) != 0) ))

dat.sim2 <- genHierSparseData(ncats = 3, nvars = 100, nobs = 200, hier.sparsity.param = hsp)
sparseBeta <- genHierSparseBeta(ncats = 3, nvars = 100, hier.sparsity.param = hsp)

## generate data with already generated beta
dat.sim3 <- genHierSparseData(ncats = 3, nvars = 100, nobs = 200, beta = sparseBeta)

## complete example:
## 50% sparsity:
hsp <- 0.2626451
dat.sim <- genHierSparseData(ncats = 3, nvars = 25,
nobs = 150, nobs.test = 1000,
hier.sparsity.param = hsp,
prop.zero.vars = 0.5,
effect.size.max = 0.25,
family = "gaussian")

x <- dat.sim$x
x.test <- dat.sim$x.test
y <- dat.sim$y
y.test <- dat.sim$y.test
grp <- dat.sim$group.ind
grp.test <- dat.sim$group.ind.test

fit.adapt <- cv.vennLasso(x, y,
grp,
adaptive.lasso = TRUE,
nlambda = 25,
family = "gaussian",
abs.tol = 1e-5,
rel.tol = 1e-5,
maxit = 1000,
irls.maxit = 15L,
gamma = 0.2,
standardize = FALSE,
intercept = TRUE,
nfolds = 3,
model.matrix = TRUE)

preds.a <- predict(fit.adapt$vennLasso.fit, x.test, grp.test, s = fit.adapt$lambda.min,
type = 'response')
Description

log likelihood function for fitted vennLasso objects

Usage

## S3 method for class 'vennLasso'
logLik(object, ...)

Arguments

object fitted "vennLasso" model object.
...
not used

Examples

library(Matrix)
set.seed(123)
n.obs <- 200
n.vars <- 50
true.beta.mat <- array(NA, dim = c(3, n.vars))
true.beta.mat[1,] <- c(-0.5, -1, 0, 0, 2, rep(0, n.vars - 5))
true.beta.mat[2,] <- c(0.5, 0.5, -0.5, -0.5, 1, -1, rep(0, n.vars - 6))
true.beta.mat[3,] <- c(0, 0, 1, 1, -1, rep(0, n.vars - 5))
rownames(true.beta.mat) <- c("1,0", "1,1", "0,1")
true.beta <- as.vector(t(true.beta.mat))
x.sub1 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub2 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub3 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x <- as.matrix(rbind(x.sub1, x.sub2, x.sub3))
conditions <- as.matrix(cbind(c(rep(1, 2 * n.obs), rep(0, n.obs)),
c(rep(0, n.obs), rep(1, 2 * n.obs))))
y <- rnorm(n.obs * 3, sd = 3) + drop(as.matrix(bdiag(x.sub1, x.sub2, x.sub3))) %*% true.beta
fit <- vennLasso(x = x, y = y, groups = conditions)
logLik(fit)
Overlapping Group Lasso (OGLasso)

Usage

\[
\text{oglasso}(x, y, \delta = \text{NULL}, \text{group}, \text{fused} = \text{NULL}, \text{family} = \text{c("gaussian", "binomial", "coxph")}, \text{nlambda} = 100L, \text{lambda} = \text{NULL}, \text{lambda.min.ratio} = \text{NULL}, \text{lambda.fused} = 0, \text{alpha} = \text{NULL}, \text{group.weights} = \text{NULL}, \text{adaptive.lasso} = \text{FALSE}, \text{adaptive.fused} = \text{FALSE}, \text{penalty.factor} = \text{NULL}, \text{penalty.factor.fused} = \text{NULL}, \gamma = 1, \text{standardize} = \text{TRUE}, \text{intercept} = \text{TRUE}, \text{compute.se} = \text{FALSE}, \text{rho} = \text{NULL}, \text{dynamic.rho} = \text{TRUE}, \text{maxit} = 500L, \text{abs.tol} = 1e-05, \text{rel.tol} = 1e-05, \text{irls.tol} = 1e-05, \text{irls.maxit} = 100L)
\]

Arguments

- **x**: input matrix of dimension nobs by nvars. Each row is an observation, each column corresponds to a covariate
- **y**: numeric response vector of length nobs
- **delta**: vector of length equal to the number of observations with values in 1 and 0, where a 1 indicates the observed time is a death and a 0 indicates the observed time is a censoring event
group
A list of length equal to the number of groups containing vectors of integers indicating the variable IDs for each group. For example, `group = list(c(1,2),c(2,3),c(3,4,5))` specifies that Group 1 contains variables 1 and 2, Group 2 contains variables 2 and 3, and Group 3 contains variables 3, 4, and 5. Can also be a matrix of 0s and 1s with the number of columns equal to the number of groups and the number of rows equal to the number of variables. A value of 1 in row i and column j indicates that variable i is in group j and 0 indicates that variable i is not in group j.

fused
matrix specifying generalized lasso penalty formulation. Each column corresponds to each variable and each row corresponds to a new penalty term, ie if row 1 has the first entry of 1 and the second entry of -1, then the penalty term `lambda.fused * |beta_1 - beta_2|` will be added. Not available now.

family
"gaussian" for least squares problems, "binomial" for binary response

nlambda
The number of lambda values. Default is 100.

lambda
A user-specified sequence of lambda values. Left unspecified, the a sequence of lambda values is automatically computed, ranging uniformly on the log scale over the relevant range of lambda values.

lambda.min.ratio
Smallest value for lambda, as a fraction of lambda.max, the (data derived) entry value (i.e. the smallest value for which all parameter estimates are zero). The default depends on the sample size `nobs` relative to the number of variables `nvars`. If `nobs > nvars`, the default is 0.0001, close to zero. If `nobs < nvars`, the default is 0.01. A very small value of `lambda.min.ratio` will lead to a saturated fit in the `nobs < nvars` case.

lambda.fused
tuning parameter for fused (generalized) lasso penalty

alpha
currently not used. Will be used later for fused lasso

group.weights
A vector of values representing multiplicative factors by which each group’s penalty is to be multiplied. Often, this is a function (such as the square root) of the number of predictors in each group. The default is to use the square root of group size for the group selection methods.

adaptive.lasso
Flag indicating whether or not to use adaptive lasso weights. If set to `TRUE` and `group.weights` is unspecified, then this will override `group.weights`. If a vector is supplied to `group.weights`, then the adaptive.lasso weights will be multiplied by the `group.weights` vector.

adaptive.fused
Flag indicating whether or not to use adaptive fused lasso weights.

penalty.factor
vector of weights to be multiplied to the tuning parameter for the group lasso penalty. A vector of length equal to the number of groups

penalty.factor.fused
vector of weights to be multiplied to the tuning parameter for the fused lasso penalty. A vector of length equal to the number of variables. mostly for internal usage

gamma
power to raise the MLE estimated weights by for the adaptive lasso. defaults to 1
standardize Logical flag for x variable standardization, prior to fitting the models. The coefficients are always returned on the original scale. Default is standardize = TRUE. If variables are in the same units already, you might not wish to standardize.

intercept Should intercept(s) be fitted (default = TRUE) or set to zero (FALSE)

compute.se Should standard errors be computed? If TRUE, then models are re-fit with no penalization and the standard errors are computed from the re-fit models. These standard errors are only theoretically valid for the adaptive lasso (when adaptive.lasso is set to TRUE)

rho ADMM parameter. must be a strictly positive value. By default, an appropriate value is automatically chosen

dynamic.rho TRUE/FALSE indicating whether or not the rho value should be updated throughout the course of the ADMM iterations

maxit integer. Maximum number of ADMM iterations. Default is 500.

abs.tol absolute convergence tolerance for ADMM iterations for the relative dual and primal residuals. Default is $10^{-5}$, which is typically adequate.

rel.tol relative convergence tolerance for ADMM iterations for the relative dual and primal residuals. Default is $10^{-5}$, which is typically adequate.

irls.tol convergence tolerance for IRLS iterations. Only used if family != "gaussian". Default is $10^{-5}$.

irls.maxit integer. Maximum number of IRLS iterations. Only used if family != "gaussian". Default is 100.

Value
An object with S3 class "oglasso"

Examples

library(vennLasso)

set.seed(123)
n.obs <- 1e3
n.vars <- 50

true.beta <- c(rep(0,2), 1, -1, rep(0, 8), 0.5, -0.5, 1, rep(0, 35))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + drop(x %*% true.beta)

groups <- c(list(c(1,2), c(2,3), c(3,4,5), 5:10, 6:12, 7:15), lapply(16:50, function(x) x))

## Not run:
fit <- oglasso(x = x, y = y, group = groups)

## End(Not run)
Description

Plot method for cv.vennLasso fitted objects
Plotting method for vennLasso fitted objects

Usage

## S3 method for class 'cv.vennLasso'
plot(x, sign.lambda = 1, ...)

## S3 method for class 'vennLasso'
plot(
  x,
  which.subpop = 1,
  xvar = c("norm", "lambda", "loglambda", "dev"),
  xlab = iname,
  ylab = "Coefficients",
  ...
)

Arguments

x fitted vennLasso or cv.vennLasso model object
sign.lambda Either plot against log(lambda) (default) or its negative if sign.lambda = -1.
... other graphical parameters for the plot
which.subpop which row in the coefficient matrix should be plotting? Each row corresponds to a particular combination of the specified stratifying variables
xvar What is on the X-axis. "norm" plots against the L1-norm of the coefficients, "lambda" against the log-lambda sequence, and "dev" against the percent deviance explained.
xlab character value supplied for x-axis label
ylab character value supplied for y-axis label

Examples

set.seed(123)

dat.sim <- genHierSparseData(ncats = 3, nvars = 25,
  nob = 100,
  hier.sparsity.param = 0.5,
  prop.zero.vars = 0.5,
  effect.size.max = 0.25,
  family = "gaussian")
```r
x <- dat.sim$x
x.test <- dat.sim$x.test
y <- dat.sim$y
y.test <- dat.sim$y.test
grp <- dat.sim$group.ind
grp.test <- dat.sim$group.ind.test

fit.adapt <- cv.vennLasso(x, y,
                        grp,
                        adaptive.lasso = TRUE,
                        nlambda = 25,
                        nfolds = 4)

plot(fit.adapt)

library(Matrix)

set.seed(123)
n.obs <- 200
n.vars <- 50

true.beta.mat <- array(NA, dim = c(3, n.vars))
true.beta.mat[1,] <- c(-0.5, -1, 0, 0, 2, rep(0, n.vars - 5))
true.beta.mat[2,] <- c(0.5, 0.5, -0.5, -0.5, 1, -1, rep(0, n.vars - 6))
true.beta.mat[3,] <- c(0, 0, 1, 1, -1, rep(0, n.vars - 5))
rownames(true.beta.mat) <- c("1,0", "1,1", "0,1")
true.beta <- as.vector(t(true.beta.mat))

x.sub1 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub2 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub3 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x <- as.matrix(rbind(x.sub1, x.sub2, x.sub3))

conditions <- as.matrix(cbind(c(rep(1, 2 * n.obs), rep(0, n.obs)),
                                c(rep(0, n.obs), rep(1, 2 * n.obs))))

y <- rnorm(n.obs * 3, sd = 3) + drop(as.matrix(bdiag(x.sub1, x.sub2, x.sub3)) %*% true.beta)

fit <- vennLasso(x = x, y = y, groups = conditions)

layout(matrix(1:3, ncol = 3))
plot(fit, which.subpop = 1)
plot(fit, which.subpop = 2)
plot(fit, which.subpop = 3)
```

---

**plotCoefs**

plotting function to investigate estimated coefficients
Description
plotting function to investigate estimated coefficients

Usage
plotCoefs(object, s = NULL, ...)

Arguments
object fitted vennLasso object
s lambda value for the predictions. Only one can be specified at a time
... other graphical parameters for the plot

Examples
set.seed(123)

dat.sim <- genHierSparseData(ncats = 3, nvars = 25, nobs = 200)

fit <- vennLasso(x = dat.sim$x, y = dat.sim$y, groups = dat.sim$group.ind)

plotCoefs(fit, s = fit$lambda[22])

plotSelections plotting function to investigate hierarchical structure of selection

Description
plotting function to investigate hierarchical structure of selection

Usage
plotSelections(object, s = NULL, type = c("d3.tree"), ...)

Arguments
object fitted vennLasso object
s lambda value for the predictions. Only one can be specified at a time
type type of plot to make. Currently only "d3.tree" and "igraph.tree" available
... other graphical parameters for the plot
plotVenn

Examples

```r
set.seed(123)

dat.sim <- genHierSparseData(ncats = 3, nvars = 25, nob = 200)

fit <- vennLasso(x = dat.sim$x, y = dat.sim$y, groups = dat.sim$group.ind)

plotSelections(fit, s = fit$lambda[32])
```

plotVenn  
plotting function for venn diagrams of overlapping conditions

Description
plotting function for venn diagrams of overlapping conditions

Usage

```r
plotVenn(
  conditions,
  condition.names = NULL,
  lty = "blank",
  fill.colors = c("royalblue1", "goldenrod1", "mediumvioletred", "turquoise3",
                  "firebrick1"),
  ...
)
```

Arguments

- **conditions**: condition matrix such as the one given to vennLasso() function. It can have up to 5 conditions
- **condition.names**: names of the conditions (equal to the number of columns of conditions)
- **lty**: standard 'lty' graphical parameter for line type around circles. default is no lines
- **fill.colors**: vector of colors for plotting. Set fill.colors = NULL for no colors
- **...**: other graphical parameters for the plot

Examples

```r
library(Matrix)

set.seed(123)
n.obs <- 200
n.vars <- 50
```
true.beta.mat <- array(NA, dim = c(3, n.vars))
true.beta.mat[1,] <- c(-0.5, -1, 0, 0, 2, rep(0, n.vars - 5))
true.beta.mat[2,] <- c(0.5, 0.5, -0.5, -0.5, 1, -1, rep(0, n.vars - 6))
true.beta.mat[3,] <- c(0, 0, 1, 1, -1, rep(0, n.vars - 5))
rownames(true.beta.mat) <- c("1,0", "1,1", "0,1")
true.beta <- as.vector(t(true.beta.mat))

x.sub1 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub2 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x.sub3 <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
x <- as.matrix(rbind(x.sub1, x.sub2, x.sub3))

conditions <- as.matrix(cbind(c(rep(1, 2 * n.obs), rep(0, n.obs)),
                               c(rep(0, n.obs), rep(1, 2 * n.obs))))

y <- rnorm(n.obs * 3, sd = 3) + drop(as.matrix(bdiag(x.sub1, x.sub2, x.sub3)) %*% true.beta)

fit <- vennLasso(x = x, y = y, groups = conditions)
vennobj <- plotVenn(conditions)

---

**predict.cv.vennLasso**

**Prediction for Cross Validation Hierarchical Lasso Object**

**Description**

Prediction for Cross Validation Hierarchical Lasso Object

**Usage**

```r
## S3 method for class 'cv.vennLasso'
predict(object, newx, group.mat, s = c("lambda.min"), use.refit = FALSE, ...)
```

**Arguments**

- **object**: fitted cv.vennLasso object
- **newx**: new matrix for predictions
- **group.mat**: A matrix of the group memberships for now. Ignore the rest: A list of length equal to the number of groups containing vectors of integers indicating the variable IDs for each group. For example, groups=list(c(1,2), c(2,3), c(3,4,5)) specifies that Group 1 contains variables 1 and 2, Group 2 contains variables 2 and 3, and Group 3 contains variables 3, 4, and 5. Can also be a matrix of 0s and 1s with the number of columns equal to the number of groups and the number of rows equal to the number of variables. A value of 1 in row i and column j indicates that variable i is in group j and 0 indicates that variable i is not in group j.
predict.vennLasso

s

lambda value for the predictions. defaults to all values computed in the vennLasso object

use.refit

Should the refitted beta estimates be used for prediction? Defaults to FALSE. If TRUE then the beta estimates from the model refit on just the selected covariates are used

... parameters to be passed to predict.vennLasso

Value

predictions or coefficients

Predictions for Hierarchical Shared Lasso

Description

Prediction for Hierarchical Shared Lasso

Usage

## S3 method for class 'vennLasso'
predict(
  object,
  newx,
  group.mat,
  s = NULL,
  use.refit = FALSE,
  type = c("link", "response", "coefficients", "nonzero", "class", "nvars", "median", "survival"),
  ...
)

Arguments

object fitted vennLasso object

newx new matrix for predictions

group.mat A matrix of the group memberships for now. Ignore the rest: A list of length equal to the number of groups containing vectors of integers indicating the variable IDs for each group. For example, groups=list(c(1,2), c(2,3), c(3,4,5)) specifies that Group 1 contains variables 1 and 2, Group 2 contains variables 2 and 3, and Group 3 contains variables 3, 4, and 5. Can also be a matrix of 0s and 1s with the number of columns equal to the number of groups and the number of rows equal to the number of variables. A value of 1 in row i and column j indicates that variable i is in group j and 0 indicates that variable i is not in group j.
lambda value for the predictions. Defaults to all values computed in the vennLasso object

Should the refitted beta estimates be used for prediction? Defaults to FALSE. If TRUE then the beta estimates from the model refit on just the selected covariates are used

type of predictions to be made. type = "median" is for the median survival time and type = "survival" is for the predicted hazard function

parameters to be passed to vennLasso

predictions or coefficients

Fitting vennLasso models

Fitting vennLasso models

Usage

vennLasso(
  x,
  y,
  groups,
  family = c("gaussian", "binomial"),
  nlambdas = 100L,
  lambda = NULL,
  lambda.min.ratio = NULL,
  lambda.fused = NULL,
  penalty.factor = NULL,
  group.weights = NULL,
  adaptive.lasso = FALSE,
  adaptive.fused = FALSE,
  gamma = 1,
  standardize = FALSE,
  intercept = TRUE,
  one.intercept = FALSE,
  compute.se = FALSE,
  conf.int = NULL,
  rho = NULL,
  dynamic.rho = TRUE,
  maxit = 500L,
  abs.tol = 1e-05,
  rel.tol = 1e-05,
irls.tol = 1e-05,
irls.maxit = 100L,
model.matrix = FALSE,
...)

Arguments

x input matrix of dimension nobs by nvars. Each row is an observation, each
column corresponds to a covariate

y numeric response vector of length nobs

groups A list of length equal to the number of groups containing vectors of integers indi-
cating the variable IDs for each group. For example, groups = list(c(1, 2), c(2, 3), c(3, 4, 5))
specifies that Group 1 contains variables 1 and 2, Group 2 contains variables 2
and 3, and Group 3 contains variables 3, 4, and 5. Can also be a matrix of 0s and
1s with the number of columns equal to the number of groups and the number
of rows equal to the number of variables. A value of 1 in row i and column j
indicates that variable i is in group j and 0 indicates that variable i is not in group
j.

family "gaussian" for least squares problems, "binomial" for binary response, and
"coxph" for time-to-event outcomes (not yet available)

nlambda The number of lambda values. Default is 100.

lambda A user-specified sequence of lambda values. Left unspecified, the a sequence
of lambda values is automatically computed, ranging uniformly on the log scale
over the relevant range of lambda values.

lambda.min.ratio Smallest value for lambda, as a fraction of lambda.max, the (data derived) entry
value (i.e. the smallest value for which all parameter estimates are zero). The
default depends on the sample size nobs relative to the number of variables
nvars. If nobs > nvars, the default is 0.0001, close to zero. If nobs < nvars,
the default is 0.01. A very small value of lambda.min.ratio can lead to a
saturated fit when nobs < nvars.

lambda.fused tuning parameter for fused lasso penalty

penalty.factor vector of weights to be multiplied to the tuning parameter for the group lasso
penalty. A vector of length equal to the number of groups

group.weights A vector of values representing multiplicative factors by which each group’s
penalty is to be multiplied. Often, this is a function (such as the square root) of
the number of predictors in each group. The default is to use the square root of
group size for the group selection methods.

adaptive.lasso Flag indicating whether or not to use adaptive lasso weights. If set to TRUE
and group.weights is unspecified, then this will override group.weights. If a
vector is supplied to group.weights, then the adaptive.lasso weights will be
multiplied by the group.weights vector

adaptive.fused Flag indicating whether or not to use adaptive fused lasso weights.

gamma power to raise the MLE estimated weights by for the adaptive lasso. defaults to 1
standardize Should the data be standardized? Defaults to FALSE.
intercept Should an intercept be fit? Defaults to TRUE.
one.intercept Should a single intercept be fit for all subpopulations instead of one for each subpopulation? Defaults to FALSE.
compute.se Should standard errors be computed? If TRUE, then models are re-fit with no penalization and the standard errors are computed from the refit models. These standard errors are only theoretically valid for the adaptive lasso (when adaptive.lasso is set to TRUE).
conf.int level for confidence intervals. Defaults to NULL (no confidence intervals). Should be a value between 0 and 1. If confidence intervals are to be computed, compute.se will be automatically set to TRUE.
rho ADMM parameter. must be a strictly positive value. By default, an appropriate value is automatically chosen.
dynamic.rho TRUE/FALSE indicating whether or not the rho value should be updated throughout the course of the ADMM iterations.
maxit integer. Maximum number of ADMM iterations. Default is 500.
abs.tol absolute convergence tolerance for ADMM iterations for the relative dual and primal residuals. Default is \(10^{-5}\), which is typically adequate.
rel.tol relative convergence tolerance for ADMM iterations for the relative dual and primal residuals. Default is \(10^{-5}\), which is typically adequate.
irls.tol convergence tolerance for IRLS iterations. Only used if family != "gaussian". Default is \(10^{-5}\).
irls.maxit integer. Maximum number of IRLS iterations. Only used if family != "gaussian". Default is 100.
model.matrix logical flag. Should the design matrix used be returned?

Value

An object with S3 class "vennLasso"

Examples

library(Matrix)

# first simulate heterogeneous data using
genHierSparseData
set.seed(123)
dat.sim <- genHierSparseData(ncats = 2, nvars = 25,
nobs = 200,
hier.sparsity.param = 0.5,
prop.zero.vars = 0.5,
family = "gaussian")

x <- dat.sim$x
conditions <- dat.sim$group.ind
y <- dat.sim$y
true.beta.mat <- dat.sim$beta.mat

fit <- vennLasso(x = x, y = y, groups = conditions)

(true.coef <- true.beta.mat[match(dimnames(fit$beta)[[1]], rownames(true.beta.mat)),])
round(fit$beta[,21], 2)

## fit adaptive version and compute confidence intervals
afit <- vennLasso(x = x, y = y, groups = conditions, conf.int = 0.95, adaptive.lasso = TRUE)

(true.coef <- true.beta.mat[match(dimnames(fit$beta)[[1]], rownames(true.beta.mat)),][,1:10])
round(afit$beta[,1:10,28], 2)
round(afit$lower.ci[,1:10,28], 2)
round(afit$upper.ci[,1:10,28], 2)

aic.idx <- which.min(afit$aic)
bic.idx <- which.min(afit$bic)

# actual coverage
mean(true.coef[afit$beta[,-1,aic.idx] != 0] >=
    afit$lower.ci[,-1,aic.idx][afit$beta[,-1,aic.idx] != 0] &
    true.coef[afit$beta[,-1,aic.idx] != 0] <=
    afit$upper.ci[,-1,aic.idx][afit$beta[,-1,aic.idx] != 0])

(covered <- true.coef >= afit$lower.ci[,1:aic.idx] & true.coef <= afit$upper.ci[,1:aic.idx])
mean(covered)

# logistic regression example
## Not run:
set.seed(123)
dat.sim <- genHierSparseData(ncats = 2, nvars = 25,
nobs = 200,
hier.sparsity.param = 0.5,
prop.zero.vars = 0.5,
family = "binomial",
effect.size.max = 0.5) # don't make any
# coefficients too big

x <- dat.sim$x
conditions <- dat.sim$group.ind
y <- dat.sim$y
true.beta.b <- dat.sim$beta.mat

bfit <- vennLasso(x = x, y = y, groups = conditions, family = "binomial")

(true.coef.b <- -true.beta.b[match(dimnames(fit$beta)[[1]], rownames(true.beta.b)),])
round(bfit$beta[,20], 2)

## End(Not run)
Index

cv.vennLasso, 2
estimate.hier.sparsity.param, 5
genHierSparseBeta, 6
genHierSparseData, 7
logLik.vennLasso, 10
oglasso, 11
plot.cv.vennLasso, 14
plot.vennLasso(plot.cv.vennLasso), 14
plotCoefs, 15
plotSelections, 16
plotVenn, 17
predict.cv.vennLasso, 18
predict.vennLasso, 19
vennLasso, 20