Package 'pensim'

July 23, 2025

Type Package

Title Simulation of High-Dimensional Data and Parallelized Repeated Penalized Regression

Version 1.3.6

Author Levi Waldron <lwaldron.research@gmail.com>

Maintainer Levi Waldron <lwaldron.research@gmail.com>

Depends R (>= 2.10.0)

Imports methods, parallel, penalized, MASS

Suggests survivalROC, survival, rmarkdown, knitr

Description Simulation of continuous, correlated high-dimensional data with time to event or binary response, and parallelized functions for Lasso, Ridge, and Elastic Net penalized regression with repeated starts and two-dimensional tuning of the Elastic Net.

License GPL (>= 2)

LazyLoad yes

VignetteBuilder knitr

URL https://waldronlab.io/pensim/

BugReports https://github.com/waldronlab/pensim/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2022-12-09 00:10:02 UTC

Contents

`pensim-package` *Functions and data for simulation of high-dimensional data and parallelized repeated penalized regression*

Description

Simulation of continuous, correlated high-dimensional data with time-to-event or binary response, and parallelized functions for Lasso, Ridge, and Elastic Net penalized regression model training and validation by split-sample or nested cross-validation. See the help page for opt.nested.crossval() for the most extensive usage examples.

Details

Package:	pensim
Type:	Package
License:	GPL (>=2)
LazyLoad:	yes

Model training and validation by Lasso, Ridge, and Elastic Net penalized regression. This package also contains a function for simulation of correlated high-dimensional data with binary or time-to-event response.

Author(s)

Levi Waldron

Maintainer: Levi Waldron <lwaldron.research@gmail.com>

References

Waldron L, Pintilie M, Tsao M-S, Shepherd FA, Huttenhower C*, Jurisica I*: Optimized application of penalized regression methods to diverse genomic data. Bioinformatics 2011, 27:3399-3406. (*equal contribution)

See Also

penalized-package

Examples

```
set.seed(9)
##
## create some data,  with one of a group of five correlated variables
## having an association with the binary outcome:
##
x <- create.data(
  nvars = c(10, 3),
  cors = c(0, 0.8),
  associations = c(0, 2),
  firstonly = c(TRUE, TRUE),
  nsamples = 50,
  response = "binary",
  logisticintercept = 0.5
)
x$summary
##
##predictor data frame and binary response vector
##
pen.data <- x$data[, -match("outcome", colnames(x$data))]
response <- x$data[, match("outcome", colnames(x$data))]
## lasso regression.  Note that epsilon=1e-2 is passed onto optL1,  and
## reduces the precision of the tuning compared to the default 1e-10.
output <-
  opt1D(
    nsim = 1,
    nprocessors = 1,
    penalized = pen.data,
    response = response,
    epsilon = 1e-2
  )
cc <-
  output[which.max(output[, "cvl"]), -1:-3]  ##non-zero b.* are true positives
```

`beer.exprs` *Lung adenocarcinoma microarray expression data of Beer et al. (2002)*

Description

Lung adenocarcinomas were profiled by Beer et al. (2002) using Affymetrix hu6800 microarrays. The data here were normalized from raw .CEL files by RMAExpress (v0.3). The expression matrix contains expression data for 86 patients with 7,129 probe sets.

Usage

```
data(beer.exprs)
```

Format

A data frame with 7129 probe sets (rows) for 86 patients (columns)

Source

Beer DG, Kardia SL, Huang C, Giordano TJ, Levin AM, Misek DE, Lin L, Chen G, Gharib TG, Thomas DG, Lizyness ML, Kuick R, Hayasaka S, Taylor JM, Iannettoni MD, Orringer MB, Hanash S: Gene-expression profiles predict survival of patients with lung adenocarcinoma. Nat Med 2002, 8:816-824.

References

Irizarry, R.A., et al. (2003) Summaries of Affymetrix GeneChip probe level data, Nucl. Acids Res., 31, e15+-e15+.

Examples

```
data(beer.exprs)
mysd <- apply(beer.exprs, 1, sd)
beer.subset <- as.matrix(beer.exprs[rank(-mysd) <= 100, ])
heatmap(beer.subset)
```

---

beer.survival *Survival data for Beer et al. (2002) lung adenocarcinoma study*

Description

Overall survival time for 86 lung adenocarcinoma patients, with 62 of the 86 events being censored.

Usage

```
data(beer.survival)
```

Format

A data frame with 86 observations on the following 2 variables.

`status` a numeric vector

`os` a numeric vector

Source

Beer DG, Kardia SL, Huang C, Giordano TJ, Levin AM, Misek DE, Lin L, Chen G, Gharib TG, Thomas DG, Lizyness ML, Kuick R, Hayasaka S, Taylor JM, Iannettoni MD, Orringer MB, Hanash S: Gene-expression pr ofiles predict survival of patients with lung adenocarcinoma. Nat Med 2002, 8:816-824.

Examples

```
data(beer.survival)
library(survival)
surv.obj <- with(beer.survival, Surv(os, status))
surv.obj.rev <- with(beer.survival, Surv(os, 1-status))
survfit(surv.obj.rev~1)  #reverse KM estimate of follow-up time (months)
(my.survfit <- survfit(surv.obj~1))  ##KM estimate of survival
plot(my.survfit, xlab="Time (months)",
     ylab="KM estimate of overall survival")
legend("bottomright", lty=c(1, 2), pch=-1,
       legend=c("KM estimate", "95 percent confidence interval"))
```

`create.data` *simulate correlated predictors with time-to-event or binary outcome*

Description

This function creates multiple groups of predictor variables which may be correlated within each group, and binary or survival time (without censoring) response according to specified weights of the predictors.

Usage

```
create.data(nvars = c(100, 100, 100, 100, 600),
            cors = c(0.8, 0, 0.8, 0, 0),
            associations = c(0.5, 0.5, 0.3, 0.3, 0),
            firstonly = c(TRUE, FALSE, TRUE, FALSE, FALSE),
            nsamples = 100,
            censoring = "none",
            labelswapprob = 0,
            response = "timetoevent",
            basehaz = 0.2,
            logisticintercept = 0)
```

Arguments

`nvars`	integer vector giving the number of variables of each variable type. The number of variable types is equal to the length of this vector.
`cors`	integer vector of the same length as nvars, giving the population pairwise Pearson correlation within each group.
`associations`	integer vector of the same length as nvars, giving the associations of each type with outcome
`firstonly`	logical vector of the same length as nvars, specifying whether only the first variable of each type is associated with outcome (TRUE) or all variables of that type (FALSE)
`nsamples`	an integer giving the number of observations

censoring	"none" for no censoring, or a vector of length two c(a,b) for uniform U(a,b) censoring.
labelswapprob	This provides an option to add uncertainty to binary outcomes by randomly switching labels with probability labelswapprob. The probability of a label being swapped is independent for each observation. The value is ignored if response is "timetoevent"
response	either "timetoevent" or "binary"
basehaz	baseline hazard, used for "timetoevent"
logisticintercept	intercept which is added to X%*%Beta for "binary"

Details

This function simulates "predictor" variables in one or more groups, which are standard normally distributed. The user can specify the population correlation within each variable group, the association of each variable group to outcome, and whether the first or all variables of that type should be associated with outcome. The simulated response variable can be time to event with an exponential distribution, or binary survival with a logistic distribution.

Value

Returns a list with items:

summary	a summary of the variable types produced
associations	weights of each variable in computing the outcome
covariance	covariance matrix used for generating potentially correlated random predictors
data	dataframe containing the predictors and response. Response is the last column for binary outcome ("outcome"), and the last two columns for timetoevent outcome ("time" and "cens")

Note

Depends on the MASS package for correlated random number generation

Author(s)

Levi Waldron et al.

References

Waldron L., Pintilie M., Tsao M.-S., Shepherd F. A., Huttenhower C.*, and Jurisica I.* Optimized application of penalized regression methods to diverse genomic data. (2010). Under review. (*equal contribution)

Examples

```
##binary outcome example
set.seed(9)
x <-
  create.data(
    nvars = c(15, 5),
    cors = c(0, 0.8),
    associations = c(0, 2),
    firstonly = c(TRUE, TRUE),
    nsamples = 50,
    response = "binary",
    logisticintercept = 0.5
  )
summary(x)
x$summary
model <- glm(outcome ~ .,  data = x$data, family = binomial)
summary(model)
dat <- t(as.matrix(x$data[, -match("outcome", colnames(x$data))]))
heatmap(dat, ColSideColors = ifelse(x$data$outcome == 0, "black", "white"))

##censored survival outcome example:
set.seed(1)
x <- create.data(
  nvars = c(15, 5),
  cors = c(0, 0.8),
  associations = c(0, 2),
  firstonly = c(TRUE, TRUE),
  nsamples = 50,
  censoring = c(2, 10),
  response = "timetoevent"
)
sum(x$data$cens == 0) / nrow(x$data)  #34 percent censoring

library(survival)
surv.obj <- Surv(x$data$time, x$data$cens)
plot(survfit(surv.obj ~ 1), ylab = "Survival probability", xlab = "time")
```

`opt.nested.crossval` *Parallelized calculation of cross-validated risk score predictions from L1/L2/Elastic Net penalized regression.*

Description

calculates risk score predictions by a nested cross-validation, using the optL1 and optL2 functions of the penalized R package for regression. In the outer level of cross-validation, samples are split into training and test samples. Model parameters are tuned by cross-validation within training samples only.

By setting nprocessors > 1, the outer cross-validation is split between multiple processors.

The functions support z-score scaling of training data, and application of these scaling and shifting coefficients to the test data. It also supports repeated tuning of the penalty parameters and selection of the model with greatest cross-validated likelihood.

Usage

```
opt.nested.crossval(outerfold=10, nprocessors=1, cl=NULL, ...)
```

Arguments

`outerfold`	number of folds in outer cross-validation (the level used for validation)
`nprocessors`	An integer number of processors to use. If specified in opt.nested.crossval, iterations of the outer cross-validation are sent to different processors. If specified in opt.splitval, repeated starts for the penalty tuning are sent to different processors.
`cl`	Optional cluster object created with the makeCluster() function of the parallel package. If this is not set, pensim calls makeCluster(nprocessors, type="SOCK"). Setting this parameter can enable parallelization in more diverse scenarios than multi-core desktops; see the documentation for the parallel package. Note that if cl is user-defined, this function will not automatically run parallel::stopCluster() to shut down the cluster.
`...`	optFUN (either "opt1D" or "opt2D"), scaling (TRUE to z-score training data then apply the same shift and scale factors to test data, FALSE for no scaling) are passed onto the opt.splitval function. Additional arguments are required, to be passed to the optL1 or optL2 function of the penalized R package. See those help pages, and it may be desirable to test these arguments directly on optL1 or optL2 before using this more CPU-consuming and complex function.

Details

This function calculates cross-validated risk score predictions, tuning a penalized regression model using the optL1 or optL2 functions of the penalized R package, for each iteration of the cross-validation. Tuning is done by cross-validation in the training samples only. Test samples are scaled using the shift and scale factors determined from the training samples. parameter. If nprocessors > 1, it uses the SNOW package for parallelization, dividing the iterations of the outer cross-validation among the specified number of processors.

Some arguments MUST be passed (through the ... arguments) but which are documented for the functions in which they are used. These include, from the opt.splitval function:

optFUN="opt1D" for Lasso or Ridge regression, or "opt2D" for Elastic Net. See the help pages for opt1D and opt2D for additional arguments associated with these functions.

scaling=TRUE to scale each feature (column) of the training sample to z-scores. These same scaling and shifting factors are applied to the test data. If FALSE, no scaling is done. Note that only data in the penalized argument are scaled, not the optional unpenalized argument (see documentation for opt1D, opt2D, or cvl from the penalized package for descriptions of the penalized and unpenalized arguments). Alternatively, the standardize=TRUE argument to the penalized package functions can be used to do scaling internally.

nsim=50 this number specifies the number of times to repeat tuning of the penalty parameters on different data foldings for the cross-validation.

setpen="L1" or "L2" : if optFUN="opt1D", this sets regression type to LASSO or Ridge, respectively. See ?opt1D.

L1range, L2range, dofirst, L1gridsize, L2gridsize: options for Elastic Net regression if optFUN="opt2D". See ?opt2D.

Value

Returns a vector of cross-validated continuous risk score predictions.

Note

Depends on the R packages: penalized, parallel

Author(s)

Levi Waldron et al.

References

Waldron L, Pintilie M, Tsao M-S, Shepherd FA, Huttenhower C*, Jurisica I*: Optimized application of penalized regression methods to diverse genomic data. Bioinformatics 2011, 27:3399-3406. (*equal contribution)

See Also

opt.splitval

Examples

```
data(beer.exprs)
data(beer.survival)

##select just 100 genes to speed computation:
set.seed(1)
beer.exprs.sample <- beer.exprs[sample(1:nrow(beer.exprs), 100), ]

gene.quant <- apply(beer.exprs.sample, 1, quantile, probs = 0.75)
dat.filt <- beer.exprs.sample[gene.quant > log2(100), ]
gene.iqr <- apply(dat.filt, 1, IQR)
dat.filt <- as.matrix(dat.filt[gene.iqr > 0.5, ])
dat.filt <- t(dat.filt)
dat.filt <- data.frame(dat.filt)

library(survival)
surv.obj <- Surv(beer.survival$os, beer.survival$status)

## First, test the regression arguments using functions from
## the penalized package.  I use maxlambda1=5 here to ensure at least
## one non-zero coefficient.
testfit <- penalized::optL1(
  response = surv.obj,
```

```
  maxlambda1 = 3,
  penalized = dat.filt,
  fold = 2,
  positive = FALSE,
  standardize = TRUE,
  trace = TRUE
)

## Now pass these arguments to opt.nested.splitval() for cross-validated
## calculation and assessment of risk scores, with the additional
## arguments:
##    outerfold and nprocessors (?opt.nested.crossval)
##    optFUN and scaling (?opt.splitval)
##    setpen and nsim (?opt1D)

## Ideally nsim would be 50, and outerfold and fold would be 10, but the
## values below speed computation 200x compared to these recommended
## values.  Note that here we are using the standardize=TRUE argument of
## optL1 rather than the scaling=TRUE argument of opt.splitval.  These
## two approaches to scaling are roughly equivalent, but the scaling
## approaches are not the same (scaling=TRUE does z-score,
## standardize=TRUE scales to unit central L2 norm), and results will
## not be identical.  Also, using standardize=TRUE scales variables but
## provides coeffients for the original scale, whereas using
## scaling=TRUE scales variables in the training set then applies the
## same scales to the test set.
set.seed(1)
## In this example I use two processors:
preds <-
  pensim::opt.nested.crossval(
    outerfold = 2,
    nprocessors = 1,
    #opt.nested.crossval arguments
    optFUN = "opt1D",
    scaling = FALSE,
    #opt.splitval arguments
    setpen = "L1",
    nsim = 1,
    #opt1D arguments
    response = surv.obj,
    #rest are penalized::optl1 arguments
    penalized = dat.filt,
    fold = 2,
    maxlambda1 = 5,
    positive = FALSE,
    standardize = TRUE,
    trace = FALSE
  )

## We probably also want the coefficients from the model fit on all the
## data, for future use:
beer.coefs <- pensim::opt1D(
  setpen = "L1",
```

```
  nsim = 1,
  maxlambda1 = 5,
  response = surv.obj,
  penalized = dat.filt,
  fold = 2,
  positive = FALSE,
  standardize = TRUE,
  trace = FALSE
)

## We can also include unpenalized covariates, if desired.
## Note that when keeping only one variable for a penalized or
## unpenalized covariate, indexing a dataframe like [1] instead of doing
## [,1] preserves the variable name.  With [,1] the variable name gets
## converted to "".

beer.coefs <- pensim::opt1D(
  setpen = "L1",
  nsim = 1,
  maxlambda1 = 5,
  response = surv.obj,
  penalized = dat.filt[-1],
  # This is equivalent to dat.filt[, -1]
  unpenalized = dat.filt[1],
  fold = 2,
  positive = FALSE,
  standardize = TRUE,
  trace = FALSE
)
## (note the non-zero first coefficient this time, due to it being unpenalized).

## Summarization and plotting.
preds.dichot <- preds > median(preds)

coxfit.continuous <- coxph(surv.obj ~ preds)
coxfit.dichot <- coxph(surv.obj ~ preds.dichot)
summary(coxfit.continuous)
summary(coxfit.dichot)

nobs <- length(preds)
cutoff <- 12
if (requireNamespace("survivalROC", quietly = TRUE)) {
preds.roc <-
  survivalROC::survivalROC(
    Stime = beer.survival$os,
    status = beer.survival$status,
    marker = preds,
    predict.time = cutoff,
    span = 0.25 * nobs ^ (-0.20)
  )
 plot(
  preds.roc$FP,
  preds.roc$TP,
```

```
 type = "l",
 xlim = c(0, 1),
 ylim = c(0, 1),
 lty = 2,
 xlab = paste("FP", "\n", "AUC = ", round(preds.roc$AUC, 3)),
 ylab = "TP",
 main = "LASSO predictions\n ROC curve at 12 months"
)
abline(0, 1)
}
```

`opt.splitval` *Parallelized calculation of split training/test set predictions from L1/L2/Elastic Net penalized regression.*

Description

uses a single training/test split to train a penalized regression model in the training samples, then use the model to calculate values of the linear risk score in the test samples. This function is used by opt.nested.crossval, but can also be used on its own.

This function support z-score scaling of training data, and application of these scaling and shifting coefficients to the test data. It also supports repeated tuning of the penalty parameters and selection of the model with greatest cross-validated likelihood.

Usage

```
opt.splitval(optFUN="opt1D",testset="equal",scaling=TRUE,...)
```

Arguments

`optFUN`	"opt1D" for Lasso or Ridge regression, "opt2D" for Elastic Net. See the help pages for these functions for additional arguments.
`testset`	For the opt.splitval function ONLY. "equal" for randomly assigned equal training and test sets, or an integer vector defining the positions of the test samples in the response, penalized, and unpenalized arguments which are passed to the optL1, optL2, or cvl functions of the penalized R package.
`scaling`	If TRUE, each feature (column) of the training samples (in matrix/dataframe specified by the penalized argument) are scaled to z-scores, then these scaling and shifting factors are applied to the test data. If FALSE, no scaling is done.
`...`	Additional arguments are required, to be passed to the optL1 or optL2 function of the penalized R package. See those help pages, and it may be desirable to test these arguments directly on optL1 or optL2 before using this more CPU-consuming and complex function.

Details

This function does split sample model training and testing for a single split of the data, using the optL1 or optL2 functions of the penalized R package, for each iteration of the cross-validation. Scaling of the test samples is done independently, using scale factors determined from the training samples. Repeated starts of model training can be parallelized as documented in the opt1D and opt2D functions. This function is used for nested cross-validation by the opt.nested.crossval function.

Value

Returns a vector of cross-validated continuous risk score predictions.

Note

Depends on the R packages: penalized, parallel, rlecuyer

Author(s)

Levi Waldron et al.

References

Waldron L, Pintilie M, Tsao M-S, Shepherd FA, Huttenhower C*, Jurisica I*: Optimized application of penalized regression methods to diverse genomic data. Bioinformatics 2011, 27:3399-3406. (*equal contribution)

See Also

opt1D, opt2D, opt.nested.crossval

Examples

```
data(beer.exprs)
data(beer.survival)

## select just 250 genes to speed computation:
set.seed(1)
beer.exprs.sample <- beer.exprs[sample(1:nrow(beer.exprs), 250), ]

gene.quant <- apply(beer.exprs.sample, 1, quantile, probs = 0.75)
dat.filt <- beer.exprs.sample[gene.quant > log2(100),]
gene.iqr <- apply(dat.filt, 1, IQR)
dat.filt <- as.matrix(dat.filt[gene.iqr > 0.5,])
dat.filt <- t(dat.filt)

library(survival)
surv.obj <- Surv(beer.survival$os, beer.survival$status)

## Single split training/test evaluation.  Ideally nsim would be 50 and
## fold=10, but this requires 100x more resources.
set.seed(1)
```

```
preds50 <- opt.splitval(
  optFUN = "opt1D",
  scaling = TRUE,
  testset = "equal",
  setpen = "L1",
  nsim = 1,
  nprocessors = 1,
  response = surv.obj,
  penalized = dat.filt,
  fold = 5,
  positive = FALSE,
  standardize = FALSE,
  trace = FALSE
)

preds50.dichot <- preds50 > median(preds50)

surv.obj.50 <-
  surv.obj[match(names(preds50), rownames(beer.survival))]
coxfit50.continuous <- coxph(surv.obj.50 ~ preds50)
coxfit50.dichot <- coxph(surv.obj.50 ~ preds50.dichot)
summary(coxfit50.continuous)
summary(coxfit50.dichot)
```

`opt1D` *Parallelized repeated tuning of Lasso or Ridge penalty parameter*

Description

This function is a wrapper to the optL1 and optL2 functions of the penalized R package, useful for parallelized repeated tuning of the penalty parameters.

Usage

```
opt1D(nsim = 50, nprocessors = 1, setpen = "L1", cl = NULL, ...)
```

Arguments

`nsim`	Number of times to repeat the simulation (around 50 is suggested)
`nprocessors`	An integer number of processors to use.
`setpen`	Either "L1" (Lasso) or "L2" (Ridge) penalty
`cl`	Optional cluster object created with the makeCluster() function of the parallel package. If this is not set, pensim calls makeCluster(nprocessors, type="SOCK"). Setting this parameter can enable parallelization in more diverse scenarios than multi-core desktops; see the documentation for the parallel package. Note that if cl is user-defined, this function will not automatically run parallel::stopCluster() to shut down the cluster.
`...`	arguments passed on to optL1 or optL2 function of the penalized R package

Details

This function sets up a SNOW (Simple Network of Workstations) "sock" cluster to parallelize the task of repeated tunings the L1 or L2 penalty parameter. Tuning of the penalty parameters is done by the optL1 or optL2 functions of the penalized R package.

Value

Returns a matrix with the following columns:

`L1 (or L2)`	optimized value of the penalty parameter
`cvl`	optimized cross-validated likelihood
`coef_1, coef_2, ..., coef_n`	argmax coefficients for the model with this value of the tuning parameter

The matrix contains one row for each repeat of the regression.

Note

Depends on the R packages: penalized, parallel, rlecuyer

Author(s)

Levi Waldron et al.

References

Waldron L, Pintilie M, Tsao M-S, Shepherd FA, Huttenhower C*, Jurisica I*: Optimized application of penalized regression methods to diverse genomic data. Bioinformatics 2011, 27:3399-3406. (*equal contribution)

See Also

optL1, optL2

Examples

```
data(beer.exprs)
data(beer.survival)

##select just 100 genes to speed computation:
set.seed(1)
beer.exprs.sample <- beer.exprs[sample(1:nrow(beer.exprs), 100),]

gene.quant <- apply(beer.exprs.sample, 1, quantile, probs = 0.75)
dat.filt <- beer.exprs.sample[gene.quant > log2(100),]
gene.iqr <- apply(dat.filt, 1, IQR)
dat.filt <- as.matrix(dat.filt[gene.iqr > 0.5,])
dat.filt <- t(dat.filt)

##define training and test sets
set.seed(1)
```

```
trainingset <- sample(rownames(dat.filt), round(nrow(dat.filt) / 2))
testset <-
  rownames(dat.filt)[!rownames(dat.filt) %in% trainingset]

dat.training <- data.frame(dat.filt[trainingset, ])
pheno.training <- beer.survival[trainingset, ]

library(survival)
surv.training <- Surv(pheno.training$os, pheno.training$status)

dat.test <- data.frame(dat.filt[testset, ])
all.equal(colnames(dat.training), colnames(dat.test))
pheno.test <- beer.survival[testset, ]
surv.test <- Surv(pheno.test$os, pheno.test$status)

##ideally nsim should be on the order of 50,  but this slows computation
##50x without parallelization.
set.seed(1)
output <-
  pensim::opt1D(
    nsim = 1,
    nprocessors = 1,
    setpen = "L2",
    response = surv.training,
    penalized = dat.training,
    fold = 3,
    positive = FALSE,
    standardize = TRUE,
    minlambda2 = 1,
    maxlambda2 = 100
  )

cc <- output[which.max(output[, "cvl"]),-(1:2)]  #coefficients
sum(abs(cc) > 0)  #count non-zero coefficients

preds.training <- as.matrix(dat.training) %*% cc
preds.training.median <- median(preds.training)
preds.training.dichot <-
  ifelse(preds.training > preds.training.median, "high risk", "low risk")
preds.training.dichot <-
  factor(preds.training.dichot[, 1], levels = c("low risk", "high risk"))
preds.test <- as.matrix(dat.test) %*% cc
preds.test.dichot <-
  ifelse(preds.test > preds.training.median, "high risk", "low risk")
preds.test.dichot <-
  factor(preds.test.dichot[, 1], levels = c("low risk", "high risk"))

coxphfit.training <- coxph(surv.training ~ preds.training.dichot)
survfit.training <- survfit(surv.training ~ preds.training.dichot)
summary(coxphfit.training)
coxphfit.test <- coxph(surv.test ~ preds.test.dichot)
survfit.test <- survfit(surv.test ~ preds.test.dichot)
summary(coxphfit.test)
```

```
(p.training <-
    signif(summary(coxphfit.training)$logtest[3], 2))  #likelihood ratio test
(hr.training <- signif(summary(coxphfit.training)$conf.int[1], 2))
(hr.lower.training <- summary(coxphfit.training)$conf.int[3])
(hr.upper.training <- summary(coxphfit.training)$conf.int[4])
par(mfrow = c(1, 2))
plot(
  survfit.training,
  col = c("black", "red"),
  conf.int = FALSE,
  xlab = "Months",
  main = "TRAINING",
  ylab = "Overall survival"
)
xmax <- par("usr")[2] - 50
text(
  x = xmax,
  y = 0.4,
  lab = paste("HR=", hr.training),
  pos = 2
)
text(
  x = xmax,
  y = 0.3,
  lab = paste("p=", p.training, "", sep = ""),
  pos = 2
)
tmp <- summary(preds.training.dichot)
text(
  x = c(xmax, xmax),
  y = c(0.2, 0.1),
  lab = paste(tmp, names(tmp)),
  col = 1:2,
  pos = 2
)
(p.test <-
    signif(summary(coxphfit.test)$logtest[3], 2))  #likelihood ratio test
(hr.test <- signif(summary(coxphfit.test)$conf.int[1], 2))
(hr.lower.test <- summary(coxphfit.test)$conf.int[3])
(hr.upper.test <- summary(coxphfit.test)$conf.int[4])
plot(
  survfit.test,
  col = c("black", "red"),
  conf.int = FALSE,
  xlab = "Months",
  main = "TEST"
)
text(
  x = xmax,
  y = 0.4,
  lab = paste("HR=", hr.test),
  pos = 2
```

```
)
text(
  x = xmax,
  y = 0.3,
  lab = paste("p=", p.test, "", sep = ""),
  pos = 2
)
tmp <- summary(preds.test.dichot)
text(
  x = c(xmax, xmax),
  y = c(0.2, 0.1),
  lab = paste(tmp, names(tmp)),
  col = 1:2,
  pos = 2
)
```

---

opt2D *Parallelized, two-dimensional tuning of Elastic Net L1/L2 penalties*

Description

This function implements parallelized two-dimensional optimization of Elastic Net penalty parameters. This is accomplished by scanning a regular grid of L1/L2 penalties, then using the top five CVL penalty combinations from this grid as starting points for the convex optimization problem.

Usage

```
opt2D(nsim,
      L1range = c(0.001, 100),
      L2range = c(0.001, 100),
      dofirst = "both",
      nprocessors = 1,
      L1gridsize = 10, L2gridsize = 10,
      cl = NULL,
      ...)
```

Arguments

nsim	Number of times to repeat the simulation (around 50 is suggested)
L1range	numeric vector of length two, giving minimum and maximum constraints on the L1 penalty
L2range	numeric vector of length two, giving minimum and maximum constraints on the L2 penalty
dofirst	"L1" to optimize L1 followed by L2, "L2" to optimize L2 followed by L1, or "both" to optimize both simultaneously in a two-dimensional optimization.
nprocessors	An integer number of processors to use.
L1gridsize	Number of values of the L1 penalty in the regular grid of L1/L2 penalties

L2gridsize	Number of values of the L2 penalty in the regular grid of L1/L2 penalties
cl	Optional cluster object created with the makeCluster() function of the parallel package. If this is not set, pensim calls makeCluster(nprocessors, type="SOCK"). Setting this parameter can enable parallelization in more diverse scenarios than multi-core desktops; see the documentation for the parallel package. Note that if cl is user-defined, this function will not automatically run parallel::stopCluster() to shut down the cluster.
...	arguments passed on to optL1 and optL2 (dofirst="L1" or "L2"), or cvl (dofirst="both") functions of the penalized R package

Details

This function sets up a SNOW (Simple Network of Workstations) "sock" cluster to parallelize the task of repeated tunings the Elastic Net penalty parameters. Three methods are implemented, as described by Waldron et al. (2011): lambda1 followed by lambda2 (lambda1-lambda2), lambda2 followed by lambda1 (lambda2-lambda1), and lambda1 with lambda2 simultaneously (lambda1+lambda2). Tuning of the penalty parameters is done by the optL1 or optL2 functions of the penalized R package.

Value

Returns a matrix with the following columns:

L1	optimized value of the L1 penalty parameter
L2	optimized value of the L2 penalty parameter
cvl	optimized cross-validated likelihood
convergence	0 if the optimization converged, non-zero otherwise (see stats:optim for details)
fncalls	number of calls to cvl function during optimization
coef_1, coef_2, ..., coef_n	argmax coefficients for the model with this value of the tuning parameter

The matrix contains one row for each repeat of the regression.

Note

Depends on the R packages: penalized, parallel, rlecuyer

Author(s)

Levi Waldron et al.

References

Waldron L, Pintilie M, Tsao M-S, Shepherd FA, Huttenhower C*, Jurisica I*: Optimized application of penalized regression methods to diverse genomic data. Bioinformatics 2011, 27:3399-3406. (*equal contribution)

See Also

optL1, optL2, cvl

Examples

```
data(beer.exprs)
data(beer.survival)

## Select just 100 genes to speed computation:
set.seed(1)
beer.exprs.sample <- beer.exprs[sample(1:nrow(beer.exprs), 100),]

## Apply an unreasonably strict gene filter here to speed computation
## time for the Elastic Net example.
gene.quant <- apply(beer.exprs.sample, 1, quantile, probs = 0.75)
dat.filt <- beer.exprs.sample[gene.quant > log2(150),]
gene.iqr <- apply(dat.filt, 1, IQR)
dat.filt <- as.matrix(dat.filt[gene.iqr > 1,])
dat.filt <- t(dat.filt)

## Define training and test sets
set.seed(9)
trainingset <- sample(rownames(dat.filt), round(nrow(dat.filt) / 2))
testset <-
  rownames(dat.filt)[!rownames(dat.filt) %in% trainingset]

dat.training <- data.frame(dat.filt[trainingset,])
pheno.training <- beer.survival[trainingset,]

library(survival)
surv.training <- Surv(pheno.training$os, pheno.training$status)

dat.test <- data.frame(dat.filt[testset,])
all.equal(colnames(dat.training), colnames(dat.test))
pheno.test <- beer.survival[testset,]
surv.test <- Surv(pheno.test$os, pheno.test$status)

set.seed(1)
##ideally set nsim=50, fold=10, but this takes 100x longer.
system.time(
  output <- opt2D(
    nsim = 1,
    L1range = c(0.1, 1),
    L2range = c(20, 1000),
    dofirst = "both",
    nprocessors = 1,
    response = surv.training,
    penalized = dat.training,
    fold = 5,
    positive = FALSE,
    standardize = TRUE
  )
```

```
)

cc <- output[which.max(output[, "cvl"]),-1:-5]
output[which.max(output[, "cvl"]), 1:5]  #small L1, large L2
sum(abs(cc) > 0)  #number of non-zero coefficients

preds.training <- as.matrix(dat.training) %*% cc
preds.training.median <- median(preds.training)
preds.training.dichot <-
  ifelse(preds.training > preds.training.median, "high risk", "low risk")
preds.training.dichot <-
  factor(preds.training.dichot[, 1], levels = c("low risk", "high risk"))
preds.test <- as.matrix(dat.test) %*% cc
preds.test.dichot <-
  ifelse(preds.test > preds.training.median, "high risk", "low risk")
preds.test.dichot <-
  factor(preds.test.dichot[, 1], levels = c("low risk", "high risk"))

coxphfit.training <- coxph(surv.training ~ preds.training.dichot)
survfit.training <- survfit(surv.training ~ preds.training.dichot)
summary(coxphfit.training)
coxphfit.test <- coxph(surv.test ~ preds.test.dichot)
survfit.test <- survfit(surv.test ~ preds.test.dichot)
summary(coxphfit.test)

(p.training <-
    signif(summary(coxphfit.training)$logtest[3], 2))  #likelihood ratio test
(hr.training <- signif(summary(coxphfit.training)$conf.int[1], 2))
(hr.lower.training <- summary(coxphfit.training)$conf.int[3])
(hr.upper.training <- summary(coxphfit.training)$conf.int[4])
par(mfrow = c(1, 2))
plot(
  survfit.training,
  col = c("black", "red"),
  conf.int = FALSE,
  xlab = "Months",
  main = "TRAINING",
  ylab = "Overall survival"
)
xmax <- par("usr")[2] - 50
text(
  x = xmax,
  y = 0.4,
  lab = paste("HR=", hr.training),
  pos = 2
)
text(
  x = xmax,
  y = 0.3,
  lab = paste("p=", p.training, "", sep = ""),
  pos = 2
)
tmp <- summary(preds.training.dichot)
```

```
text(
  x = xmax,
  y = c(0.2, 0.1),
  lab = paste(tmp, names(tmp)),
  col = 1:2,
  pos = 2
)
## Now the test set.
## in the test set,  HR=1.7 is not significant - not surprising with the
## overly strict non-specific pre-filter (IQR>1,  75th percentile > log2(150)
(p.test <-
    signif(summary(coxphfit.test)$logtest[3], 2))  #likelihood ratio test
(hr.test <- signif(summary(coxphfit.test)$conf.int[1], 2))
(hr.lower.test <- summary(coxphfit.test)$conf.int[3])
(hr.upper.test <- summary(coxphfit.test)$conf.int[4])
plot(
  survfit.test,
  col = c("black",  "red"),
  conf.int = FALSE,
  xlab = "Months",
  main = "TEST"
)
text(
  x = xmax,
  y = 0.4,
  lab = paste("HR=", hr.test),
  pos = 2
)
text(
  x = xmax,
  y = 0.3,
  lab = paste("p=", p.test, "", sep = ""),
  pos = 2
)
tmp <- summary(preds.test.dichot)
text(
  x = xmax,
  y = c(0.2, 0.1),
  lab = paste(tmp, names(tmp)),
  col = 1:2,
  pos = 2
)
```

`scan.l1l2` *Function calculate cross-validated likelihood on a regular grid of L1/L2 penalties*

Description

This function generates a grid of values of L1/L2 penalties, then calculated cross-validated likelihood at each point on the grid. The grid can be regular (linear progression of the penalty values), or

polynomial (finer grid for small penalty values, and coarser grid for larger penalty values).

Usage

```
scan.l1l2(L1range = c(0.1, 100.1),
          L2range = c(0.1, 100.1),
          L1.ngrid = 50,
          L2.ngrid = 50,
          nprocessors = 1,
          polydegree = 1,
          cl = NULL,
          ...)
```

Arguments

L1range	numeric vector of length two, giving minimum and maximum constraints on the L1 penalty
L2range	numeric vector of length two, giving minimum and maximum constraints on the L2 penalty
L1.ngrid	Number of values of the L1 penalty in the regular grid of L1/L2 penalties
L2.ngrid	Number of values of the L2 penalty in the regular grid of L1/L2 penalties
nprocessors	An integer number of processors to use.
polydegree	power of the polynomial on which the L1/L2 penalty values are fit. ie if polydegree=2, penalty values could be y=x^2, x=1,2,3,..., so y=1,4,9,...
cl	Optional cluster object created with the makeCluster() function of the parallel package. If this is not set, pensim calls makeCluster(nprocessors, type="SOCK"). Setting this parameter can enable parallelization in more diverse scenarios than multi-core desktops; see the documentation for the parallel package. Note that if cl is user-defined, this function will not automatically run parallel::stopCluster() to shut down the cluster.
...	arguments passed on to cvl function of the penalized R package

Details

This function sets up a SNOW (Simple Network of Workstations) "sock" cluster to parallelize the task of scanning a grid of penalty values to search for suitable starting values for two-dimensional optimization of the Elastic Net.

Value

cvl	matrix of cvl values along the grid
L1range	range of L1 penalties to scan
L2range	range of L2 penalties to scan
xlab	A text string indicating the range of L1 penalties
ylab	A text string giving the range of L2 penalties
zlab	A text string giving the range of cvl values

note	A note to the user that rows of cvl correspond to values of lambda1, columns to lambda2

Note

Depends on the R packages: penalized, parallel, rlecuyer

Author(s)

Levi Waldron et al.

References

Waldron L, Pintilie M, Tsao M-S, Shepherd FA, Huttenhower C*, Jurisica I*: Optimized application of penalized regression methods to diverse genomic data. Bioinformatics 2011, 27:3399-3406. (*equal contribution)

See Also

cvl

Examples

```
data(beer.exprs)
data(beer.survival)

##select just 250 genes to speed computation:
set.seed(1)
beer.exprs.sample <- beer.exprs[sample(1:nrow(beer.exprs), 250), ]

gene.quant <- apply(beer.exprs.sample, 1, quantile, probs = 0.75)
dat.filt <- beer.exprs.sample[gene.quant > log2(150), ]
gene.iqr <- apply(dat.filt, 1, IQR)
dat.filt <- as.matrix(dat.filt[gene.iqr > 1, ])
dat.filt <- t(dat.filt)

## Define training and test sets
set.seed(9)
trainingset <- sample(rownames(dat.filt), round(nrow(dat.filt) / 2))
testset <- rownames(dat.filt)[!rownames(dat.filt) %in% trainingset]

dat.training <- data.frame(dat.filt[trainingset, ])
pheno.training <- beer.survival[trainingset, ]

library(survival)
surv.training <- Surv(pheno.training$os, pheno.training$status)

dat.test <- data.frame(dat.filt[testset, ])
all.equal(colnames(dat.training), colnames(dat.test))
pheno.test <- beer.survival[testset, ]
surv.test <- Surv(pheno.test$os, pheno.test$status)
```

```
set.seed(9)
system.time(
  output <- scan.l1l2(
    L1range = c(0.2, 3.2),
    L2range = c(2, 30),
    L1.ngrid = 10,
    L2.ngrid = 10,
    polydegree = 1,
    nprocessors = 1,
    response = surv.training,
    penalized = dat.training,
    fold = 4,
    positive = FALSE,
    standardize = TRUE
  )
)

##Note that the cvl surface is not smooth because a different folding of
##the data was used for each cvl calculation
image(
  x = seq(output$L1range[1], output$L1range[2], length.out = nrow(output$cvl)),
  y = seq(output$L2range[1], output$L2range[2], length.out = ncol(output$cvl)),
  z = output$cvl,
  xlab = "lambda1",
  ylab = "lambda2",
  main = "red is higher cross-validated likelihood"
)
```

Index