Package ‘aucm’

December 2, 2019

LazyLoad yes
LazyData yes
Version 2019.12-1
Title AUC Maximization
Maintainer Youyi Fong <youyifong@gmail.com>
Depends R (>= 3.1.3), kyotil
Suggests RUnit, mvtnorm
Description Implements methods for identifying linear and nonlinear marker combinations that maximizes the Area Under the AUC Curve (AUC).
License GPL-2
NeedsCompilation yes
Author Youyi Fong [cre],
  Krisztian Sebestyen [aut],
  Shuxin Yin [aut],
  Ying Huang [aut],
  Alexandros Karatzoglou [ctb],
  Alex Smola [ctb],
  Kurt Hornik [ctb]
Repository CRAN
Date/Publication 2019-12-02 09:20:02 UTC

R topics documented:

  auc ......................................................... 2
  aucm ....................................................... 3
  bupa ....................................................... 3
  cleveland ............................................... 4
  control.minQuad ................................. 5
  dcsauc ............................................... 7
  get.X.diff ............................................. 9
  getQ .................................................... 10
  grid.auc .............................................. 11
AUC methods.

Usage

## S3 method for class 'auc'
coef(object, ...)
## S3 method for class 'auc'
predict(object, newdata, case.percentage = NULL, ...)
## S3 method for class 'auc'
print(x, ...)
## S3 method for class 'auc'
summary(object, ...)
## S3 method for class 'auc'
trainauc(fit, training.data = NULL, ...)
## S3 method for class 'auc'
ratio(fit)

## S3 method for class 'glm'
trainauc(fit, ...)
## S3 method for class 'glm'
ratio(fit)

Arguments

fit an object that inherits from class 'auc' such as 'rauc' or 'sauc'
object an object that inherits from class 'auc' such as 'rauc' or 'sauc'
x an object that inherits from class 'auc' such as rauc, sauc or sauc.dca.
newdata data at which to predict
case.percentage

used for class prediction, defaults to NULL

training.data

data frame used to compute auc based on a fit obtained by a call to rauc, sauc
or sauc.dca

... arguments passed to or from methods

Author(s)

Youyi Fong <youyifong@gmail.com>
Krisztian Sebestyen <>

bupa

Bupa Dataset

Description

Bupa Dataset

Usage

data(bupa)

Format

A data frame with 345 observations on the following 14 variables.

Case  a numeric vector
V1   a numeric vector
V2   a numeric vector
V3   a numeric vector
V4   a numeric vector
V5   a numeric vector
V6   a numeric vector
y    a numeric vector
V1.2 a numeric vector
Examples
data(bupa)
### maybe str(bupa) ; plot(bupa) ...

cleveland  Cleveland Dataset

Description
cleveland Dataset

Usage
data(cleveland)

Format
A data frame with 297 observations on the following 15 variables.
V1  a numeric vector
V2  a numeric vector
V3  a numeric vector
V4  a numeric vector
V5  a numeric vector
V6  a numeric vector
V7  a numeric vector
V8  a numeric vector
V9  a numeric vector
V10  a numeric vector
V11  a numeric vector
V12  a numeric vector
V13  a numeric vector
case  a logical vector
y  a numeric vector

Examples
data(cleveland)
### maybe str(cleveland) ; plot(cleveland) ...
Description

Control function to `minQuad`

Usage

```
control.minQuad(
    maxit = 1e4, tol = 1e-04,
    q = 0,
    ws = c("v","v2","greedy","rv2wg","rvwg","rv","rv2"),
    method = c("default","tron","loqo","exhaustive","x"),
    optim.control = list(),
    rank = 0,
    DUP = FALSE,
    NAOK = FALSE,
    verbose = FALSE,
    ret.ws = FALSE,
    ret.data = FALSE
)
```

Arguments

- `rank` a nonnegative integer indicating the 'rank' of the design matrix, only used by by method 'exhaustive' or 'x'. if zero it is estimated by the singular value decomposition of each 'sub-matrix' associated with the sub-optimization problem in the decomposition method.
- `method` a character string (first letter is sufficient) indicating which quadratic optimizer to use, defaults to 'default'. See details.
- `optim.control` a list of control parameters to methods 'tron' or 'loqo'; 'tron' : list(maxfev = 1000,fatol = tol,frtol = tol,ctol=tol,gtol=tol,fmin = - .Machine$double.xmax), 'loqo' : list(bound = 10,margin=0.05,maxiter=40,sigfig = 7,inf = 1e6)
- `q` size of the working set, will be set to 2 for all methods except for method = 'tron' when it defaults to NULL. In that case workings set size is automatically chosen to be sqrt(#violators) at each iteration.
- `ws` a character string indicating the strategy of how to select the working set, de-faults to "rv2wg", see details.
- `maxit` maximum number of iterations whose `typeof` returns "integer".
- `tol` tolerance for termination criterion whose `typeof` returns "double".
control.minQuad

DUP should arguments be passed by reference? defaults to FALSE.

NAOK should NA’s,NaN’s be allowed to be passed to C code (no checking)? defaults to FALSE.

verbose some output at each iteration, possible values are FALSE/TRUE or an integer if more details are wanted, defaults to FALSE.

ret.ws defaults to FALSE, indicates whether to return the working set selected at each iteration.

ret.data defaults to FALSE, indicates whether to return the data passed to minQuad.

Details

Four quadratic optimizers are available within minQuad: "default", "tron", "loqo" and "exhaustive" (optimizer 'x' is a slightly faster implementation of the exhaustive method). For working set size q = 2, the 'default' option is a fast implementation that loosely minimizes the quadratic objective function, which is often sufficient to achieve convergence in the DCA-loop in rauc. For working set size q = 2, the 'default' option minimizes the quadratic objective function by "solving" an associated equation at each data point. The "exhaustive" method is a brute-force method that gives an exact solution to each quadratic sub-problem in minQuad and should probably not be used beyond working set size q = 8,10 on most computers. Method 'tron' is a positive semidefinite quadratic optimizer and thus well suited for low-rank problems - for this method 'q' can be larger, ~100 or perhaps even ~1000. Method 'loqo' is a positive definite quadratic optimizer that accepts 'm' constraints specified by (m x n) matrix A in the form v <= A*x <= v+r with both v and r finite. The default value of the size of the working set 'q' is 0. This means that if 'method' is 'tron' then 'q' is automatically set to the sqrt(no. of violators) at the current iteration in minQuad (rounded up). Otherwise 'q' defaults to 2 but may be set to any nonzero integer that is greater than 1. The "ws" argument sets the type of strategy to select the working set. Denote the two sets of violators as V0 = {1 if (a[p] > 0.0), (df[p] > 0.0), 0 ow.}, VC = {1 if (a[p] < C), (df[p] < 0.0), 0 ow.} where "df[a]" stands for the gradient of the objective function at 'a'. "greedy" selects the extremes pairs (max, min) from the two sets (M, m) where M = {-df[i], a[i] < C} and m = {-df[j], |a[j]| > 0}. "v" selects from violators V = V0 U VC ranked by |df|. "v2" selects separately from V0 and VC separately, ranked by |df|. "rv" selects without replacement (WOR) from all violators. "rvwg" selects WOR from all violators V with probability ~ |df[V]|. "rv2wg" selects WOR from the two sets of violators V0 and VC with probability ~ |df[V]|.

Value

A list with the following elements:

convergence 0 if converged, 1 if maximum iteration is reached.
alpha estimated vector of coefficients.
value value of the objective function.
iterations number of iterations until convergence or "maxit" reached.
epsilon stopping rule value to be compared to "tol".

n
nSV #{0 < a}, no. of support vectors.
nBSV #{a==C}, no. of bounded support vectors.
**dcssauc**  

nFSV  #\{0 < \alpha < C\}, no. of unbounded support vectors.

control  the control argument.

ws  if requested, the working set selected at current iteration.

**Author(s)**

Krisztian Sebestyen <ksebestyen@gmail.com>
Youyi Fong <youyifong@gmail.com>
Shuxin Yin <>

**References**

*Combining Biomarkers Nonlinearily for Classification Using the Area Under the ROC Curve* Y. FONG, S. YIN, Y. HUANG Biometrika (2012), pp 1-28


**See Also**

rauc, minQuad

---

**Description**

optimizes SAUC using a smoothed DCA algorithm.

**Usage**

```R
dcssauc (formula, data, ...)  
srauc (formula, data, ...)  
auc.dca (formula, data,  
  type="srauc",  
  kernel="linear", para=NULL,  
  lambda=.1, zeta=.1, b=10, s=1, epsilon=1e-3,  
  method="tron", decomposition=TRUE,  
  dca.control = list(maxit=1e3, abstol=1e-5, coef.init=NULL, lincomb.init=NULL),  
  tron.control = list(q=50, maxfev=1e3, gtol=1e-2, frtol=1e-12, K.thresh=1, verbose=0),  
  return.K=FALSE, verbose = FALSE  
)
```
Arguments

- **formula**: formula, e.g. `y~x1+x2`
- **data**: a data frame
- **type**: string. Either srauc or dcsauc
- **kernel**: See `getK` for more details
- **para**: See `getK` for more details
- **lambda**: scale parameter of the penalty function, defaults to 1
- **zeta**: parameter (\( \rightarrow 0^+ \)) in writing sigmoid function as difference of two convex functions.
- **b**: 'decay rate' parameter in sigmoid function \( \frac{1}{\exp(bx)} \)
- **s**: the parameter in `rauc`
- **epsilon**: the parameter in the approximation of a hinge function
- **method**: the optimizer to use, "tron", or an `optim` method
- **decomposition**: Boolean. If TRUE, decomposition strategy is used if tron is the method
- **dca.control**: list of control parameters for the DCA algorithm
- **tron.control**: list of control parameters to 'tron' optimizer
- **return.K**: logical, whether to return the Kernel matrix
- **verbose**: logical, whether to print info as alg. progresses
- **...**: parameters passed to auc.dca

Details

dcsauc and srauc pass directly to auc.dca with the name-sake type.

Examples

```r
# # dat = sim.dat.1(n=100, seed=1)
# dat.test = sim.dat.1(n=1e3, seed=1000)
#
# t.1 = system.time(
#   fit.1 = sauc.dca(y~x1+x2, dat, zeta=.1)
# )
#
# t.2 = system.time(
#   fit.2 = sauc.dca(y~x1+x2, dat, zeta=1)
# )
#
# ## compare time
# rbind(t.1, t.2)[,3]
# ## compare performance
# RUnit::checkEqualsNumeric(
#   c(fit.1$train.auc, fit.2$train.auc)
```
get.X.diff

#, c(0.7291917, 0.7282913), tolerance=1e-6)
#

get.X.diff    get.X.diff

Description

computes X.diff matrix

Usage

get.X.diff (x1,...)
## Default S3 method:
get.X.diff(x1,x2,...)
## S3 method for class 'formula'
get.X.diff(formula, data,...)

Arguments

x1      data matrix from the case group, dimension n1 x d
x2      data matrix from the non-case group, dimension n2 x d
formula a formula
data      a data frame
...      arguments passed 'to' or 'from' methods

Details

In get.X.diff.formula, x is the case predictors and x2 control.

Value

A (n1*n2) x d matrix

Author(s)

Shuxin Yin <>
Youyi Fong <youyifong@gmail.com>
Krisztian Sebestyen <>
Examples

```
dat = sim.dat.1(n=100,seed=1)
X1 = as.matrix(subset(dat, y==0, select=c(x1,x2)))
X2 = as.matrix(subset(dat, y==1, select=c(x1,x2)))
X.diff = get.X.diff (X1, X2)
dim(X1)
dim(X2)
dim(X.diff)
```

Description

getQ calculates Q or Q.pred matrix depending on the value of do.pred.

Usage

```
getQ (K,n1,n2,call.C=TRUE,do.pred=FALSE)
```

Arguments

- `K`: kernel matrix of dimension \((n1+n2)\) by \((n1+n2)\) or \(n.pred\) by \((n1+n2)\). The \((n1+n2)\) observations must be ordered case followed by non-case.
- `n1`: number of cases
- `n2`: number of non-cases
- `call.C`: boolean. If TRUE, make \(\cdot C\) call, otherwise compute Q in R.
- `do.pred`: boolean. If TRUE, K is a \(n.pred\) by \((n1+n2)\) matrix; otherwise, it is a \((n1+n2)\) by \((n1+n2)\) matrix.

Value

A \(n1\times n2\) by \(n1\times n2\) matrix if do.pred is FALSE, or \(n.pred\) by \(n1\times n2\) matrix if do.pred is TRUE

Author(s)

Shuxin Yin <>
Youyi Fong <youyifong@gmail.com>
Krisztian Sebestyen <>
Examples

dat = sim.dat.1(n=100,seed=1)
dat=rbind(subset(dat, y==1), subset(dat, y==0))
X = as.matrix(subset(dat, select=c(x1,x2)))
n1=sum(dat$y)
n2=sum(1-dat$y)

K = kyotil::getK(X,"linear", 1)
Q1 = getQ(K,n1=n1,n2=n2,call.C=FALSE)
Q2 = getQ(K,n1=n1,n2=n2,call.C=TRUE)
all(Q2-Q1<1e-6)

# compare to a direct computation
X.diff=get.X.diff(X[1:n1,], X[1:n2+n1,])
Q3 = tcrossprod(X.diff, X.diff)
all(Q3-Q1<1e-6)

# two printouts of Q2 should not be different
Q2[1:3,1:3]
K = kyotil::getK(X,"rbf", 1)
Q4 = getQ(K,n1=n1,n2=n2,call.C=TRUE)
Q2[1:3,1:3]
Q4[1:3,1:3]

K = kyotil::getK(X[1:10,],"linear", 1, X2=X)
Q5 = getQ(K,n1=n1,n2=n2,call.C=FALSE,do.pred=TRUE)
Q6 = getQ(K,n1=n1,n2=n2,call.C=TRUE,do.pred=TRUE)
dim(Q5)
dim(Q6)
all(Q5-Q6<1e-6)

Description

grid search for beta that maximize (penalized, partial) auc/sauc/rauc eauc maximizes empirical AUC, but only works with two covariates

Usage

grid.auc (formula, dat, beta, approx.type=NULL, approx.param=1, lambda=0, loss=TRUE, t0=NULL, t1=NULL,ret.vcov = FALSE)
eauc (formula, dat,t0 = NULL, t1 = NULL)
Arguments

- **formula**: a formula
- **dat**: a data frame
- **beta**: a matrix of coefficients
- **approx.type**: a string. If NULL, AUC is computed. If "phi", normal CDF approximation SAUC is computed. If "logistic", logistic approximation SAUC is computed. If "rauc", ramp AUC approximation is computed. Defaults to NULL
- **approx.param**: 's' for rauc, 'h' for sauc
- **loss**: a boolean. TRUE is default and means 1-(p)RAUC is computed. If lambda is not 0, loss is forced to be TRUE internally.
- **t0**: a number between 0 and 1 that is the lower boundary of pAUC
- **t1**: a number between 0 and 1 that is the upper boundary of pAUC
- **lambda**: a number that scales the L2 penalty, default to 0, meaning no penalty. If lambda is not 0, loss is forced to be TRUE.
- **ret.vcov**: logical, whether to return an estimate of the covariance matrix of 'beta' for normal or logistic sigmoid functions.

Details

eauc is a shortcut for grid.auc when empirical AUC is the objective function. When loss is FALSE, the criterion function is mean_i_j(loss) When loss is TRUE, including when lambda is not 0, the criterion function is sum_i_j(loss) + 0.5 * lambda * pen, i.e. the penalty is added to the sum of penalty and not mean of the penalty.

Value

A n x n matrix

Author(s)

- Shuxin Yin <>
- Youyi Fong <youyifong@gmail.com>
- Krisztian Sebestyen <>

Examples

```r
library(aucm)

dat = sim.dat.1(n=200,seed=1)
beta=cbind(4, 4*seq(-1,0,length=100))
dim(beta)

fit = eauc(y~x1+x2, dat)
## Not run:
```
kyphosis

# not run due to r cmd check requirements

par(mfrow=c(3,2))

out1 = grid.auc (y~x1+x2, dat, beta, approx.param=1, loss=FALSE, t0=NULL, t1=NULL, lambda=0,
approx.type="rauc")
plot(out1$par[2,]/out1$par[1,],out1$vals,type="l",xlab=expression(beta[2]/beta[1]),main="RAUC")

# penalized RAUC
out2 = grid.auc (y~x1+x2, dat, beta, approx.param=1, loss=TRUE, t0=NULL, t1=NULL, lambda=0,
approx.type="rauc")
out3 = grid.auc (y~x1+x2, dat, beta, approx.param=1, loss=TRUE, t0=NULL, t1=NULL, lambda=30,
approx.type="rauc")
plot(out2$par[2,]/out2$par[1,],out2$vals,type="l",xlab=expression(beta[2]/beta[1]),main="penalized RAUC loss")
lines(out3$par[2,]/out3$par[1,],out3$vals,type="l",col=2)
out2$par
out3$par

# pRAUC
out4 = grid.auc (y~x1+x2, dat, beta, approx.param=1, loss=FALSE, t0=0, t1=0.5, lambda=0,
approx.type="rauc")
out5 = grid.auc (y~x1+x2, dat, beta, approx.param=1, loss=FALSE, t0=0.5, t1=1, lambda=0,
approx.type="rauc")
plot(out4$par[2,]/out4$par[1,],out4$vals,type="l",xlab=expression(beta[2]/beta[1]),main="pRAUC")
plot(out5$par[2,]/out5$par[1,],out5$vals,type="l",xlab=expression(beta[2]/beta[1]),main="pRAUC")
out4$par
out5$par

# penalized pRAUC
out6 = grid.auc (y~x1+x2, dat, beta, approx.param=1, loss=TRUE, t0=0, t1=0.5, lambda=0,
approx.type="rauc")
out7 = grid.auc (y~x1+x2, dat, beta, approx.param=1, loss=TRUE, t0=0, t1=0.5, lambda=10,
approx.type="rauc")
plot(out6$par[2,]/out6$par[1,],out6$vals,type="l",xlab=expression(beta[2]/beta[1]),
main="penalized pRAUC loss")
lines(out7$par[2,]/out7$par[1,],out7$vals,type="l",col=2)
out3$par
out7$par

## End(Not run)

---

kyphosis

Kyphosis Data

Description

Kyphosis Data
Usage

   data(kyphosis)

Format

   A data frame with 81 observations on the following 5 variables.

   Kyphosis a factor with levels absent present
   Age   a numeric vector
   Number a numeric vector
   Start  a numeric vector
   y      a numeric vector

Examples

   data(kyphosis)
   ## maybe str(kyphosis) ; plot(kyphosis) ...

minQuad

Description

   minimizes the objective function 0.5a'Qa + b'a with respect to "a" subject to 0 <= a <= C, or to the
   constraints lower <= a <= upper, v <= Ax <= v + r.

Usage

   minQuad(H,b,C = 1.0,n1=0,n2=0,
       mem.efficient = FALSE,alpha = NULL,
       lower = NULL,upper = NULL,mat.constr = NULL, lhs.constr = NULL,rhs.constr = NULL,
       control = list(DUP = TRUE,maxit = 1e4, tol = 1e-04,
                       verbose = FALSE, ret.ws = FALSE,ret.data = FALSE,
                       rank = 0,
                       method = c("default","tron","loqo","exhaustive","x"),
                       optim.control = list(),
                       q = 2,
       ws = c("v","v2","greedy","rv2wg","rvwg","rv","rv2")
   )
)
Arguments

H
A symmetric matrix whose typeof returns "double" of dimension (n x n). If mem.efficient = FALSE n = n1*n2 matches the length of the vector 'b' else n = n1 + n2, see details, defaults to NULL.

b
a numeric vector of length 'n' whose typeof returns "double".

C
a numeric variable whose typeof returns "double", defaults to 1.0. It is the upper bound on alpha's, where the lower bound is 0.0.

n1, n2
integer variables giving the specific values for n1 = #{diseased}, n2 = #{non-diseased} subjects if mem.efficient = TRUE.

mem.efficient
logical, if FALSE then 'H' is represented by the (n1n2 x n1n2) matrix 'Q' else by the (n1+n2 x n1+n2) matrix 'K', defaults to FALSE.

alpha
a length-n1n2 vector vector of initial values for "alpha", whose typeof returns "double", defaults to NULL, in which case it is set to 0.5C.

ccontrol
a list with control parameters. See control.minQuad

mat.constr
m x n constraint matrix for loqo optimizer

lhs.constr
numeric of length 'm', the left hand side constraints for loqo optimizer

rhs.constr
numeric of length 'm', the left hand side for constraints for loqo optimizer

lower
numeric of length 'n', the lower bounds on primal variables for loqo optimizer

upper
numeric of length 'n', the upper bounds on primal variables for loqo optimizer

Details

The function minQuad passes its arguments by "reference" via the .C function call to C if DUP = FALSE to avoid copying the large matrix "Q". When 'H' = 'Q', 'Q' is a symmetric matrix and should have numeric type "double", be of type "matrix" not of "data.frame": is.matrix(.) should return "TRUE". We do not make an extra copy by tranposing Q but access the 'flattened' vector in C directly since 'Q' is symmetric. When 'mem.efficient' = TRUE 'H' = K_{(n1+n2 x n1+n2)} and may be obtained by the function getK. 'K' is relevant to AUC estimation, see rauc for more details. The "ws" argument sets the type of strategy to select the working set. Denote the two sets of violators as V0 = {1 if (a[p] > 0.0),(df[p] > 0.0), 0 ow.}, VC = {1 if (a[p] < C),(df[p] < 0.0), 0 ow.} where "df[a]" stands for the gradient of the objective function at 'a'. "greedy" selects the extremes pairs (max,min) from the two sets (M,m) where M = {-df[i], a[i] < C} and m = {-df[j] | a[j] > 0}. "v" selects from violators V = V0 U VC ranked by ldf. "v2" selects separately from V0 and VC separately, ranked by ldf. "rv" selects without replacement (WOR) from all violators. "rvwg" selects WOR from all violators V with probability ~ |df[V]|. "rv2wg" selects WOR from the two sets of violators V0 and VC with probability ~ |df[V]|. Three methods are available, "tron", "hideo" and "exhaustive". Optimizer 'x' is a slightly faster implementation of the exhaustive method, whereas 'default' is a fast implementation for q = 2 only. The "exhaustive" method should probably not be used beyond working set size q = 8,10 on most computers. The 'loqo' optimizer accepts constraints of the form v <= A*x <= v + r, lower <= x <= upper. v = lhs.constr and r = rhs.constr - lhs.constr. The entries in 'v', 'r' and 'A' must be finite. When verbose is TRUE, each DCA iteration prints one line. Delta means, for the linear kernel, max(abs((beta.new-beta.init)/beta.init)), and for the nonlinear kernel the difference in penalized RAUC loss. epsilon is the KKT criterion of minQuad.
Value

A list with the following elements:

- convergence: 0 if converged, 1 if maximum iteration is reached.
- alpha: estimated vector of coefficients.
- value: value of the objective function.
- iterations: number of iterations until convergence or "maxit" reached.
- epsilon: stopping rule value to be compared to "tol".
- n SV: #{0 < a}, no. of support vectors.
- n BSV: #{a==C}, no. of bounded support vectors.
- n FSV: #{0 < alpha < C}, no. of unbounded support vectors.
- control: the control argument.
- ws: if requested, the working set selected at current iteration.

Author(s)

Krisztian Sebestyen <ksebestyen@gmail.com>
Youyi Fong <youyifong@gmail.com>
Shuxin Yin <>

References

Combining Biomarkers Nonlinearly for Classification Using the Area Under the ROC Curve Y. FONG, S. YIN, Y. HUANG Biometrika (2012), pp 1-28


See Also

control.minQuad, rauc
Description
computes ramp function value from paired difference of linear combinations

Usage
ramp.f (eta,s,loss=TRUE)

Arguments
eta a vector of paired difference of linear combinations
s absolute value of the slope parameter
loss a boolean. If TRUE, return loss function i.e. 1 - RAUC. If FALSE, return RAUC. Default to TRUE, because we minimize loss.

Value
A vector of same size as eta

Author(s)
Shuxin Yin <>
Youyi Fong <youyifong@gmail.com>
Krisztian Sebestyen <>

Examples

dat = sim.dat.1(n=100,seed=1)
X1 = as.matrix(subset(dat, y==0, select=c(x1,x2)))
X2 = as.matrix(subset(dat, y==1, select=c(x1,x2)))
X.diff = get.X.diff (X1, X2)
dim(X1)
dim(X2)
dim(X.diff)
aux = ramp.f(X.diff %*% c(1,1), s=1)
length(aux)
mean(aux)
aux = ramp.f(X.diff %*% c(1,1), s=1, loss=FALSE)
length(aux)
mean(aux)
rauc

Description
minimizes 1 - (p)AUC plus a penalty

Usage
rauc(formula, dat, s = 1, lambda = 1, kernel = "linear", para = NULL, start.method = "rlogit", eta0.init = NULL, beta.init = NULL, eta.diff.init = NULL, maxit = 50, tol = 1e-5, minQuad.control = control.minQuad(), init.alpha.from.previous = TRUE, mem.efficient = TRUE, ret.vcov = FALSE, garbage.collection = TRUE, verbose = FALSE, ...)

Arguments

formula      formula, e.g. y~x1+x2
dat          Data frame
s             absolute value of the slope, default to 1 - REMOVE THIS, the pair (s,lambda) is redundant
lambda       scale parameter in front of the penalty function, default to 1
kernel       See getK for more details
para         See getK for more details
start.method a string. When kernel is linear: If "rlogit", robust logistic fit is used as beta.init. If "1", a vector of 1 is used as beta.init. If "0", a vector of 0 is used as beta.init.
eta0.init    a vector of the same length as the number of rows in dat
beta.init    a vector of length equal to no. of covariates (without intercept) of initial values for linear kernel.
eta.diff.init a vector of the same length as the number of rows in dat
maxit        maximum number of iterations in the DCA algorithm
tol          absolute tolerance in RAUC if kernel is not linear, relative tolerance in coefficients if kernel is linear.
minQuad.control control parameters passed to method minQuad, please see minQuad.
init.alpha.from.previous defaults to TRUE, if TRUE then after the first iteration minQuad receives as the initial "alpha" the estimate of "alpha" from the previous iteration in dca algorithm.
mem.efficient if TRUE, the small matrix 'K' instead of 'Q' is used in computations, defaults to TRUE.
ret.vcov  logical, whether to return an estimate of the covariance matrix of 'beta' for normal or logistic sigmoid functions.

garbage.collection  logical, whether to call gc at end of each DCA iteration

verbose  prints information at each iteration, defaults to FALSE

Value

A list with the following elements:

convergence  0 if converged, 1 if maximum iteration is reached.

value  value of the objective function.

iterations  number of iterations until convergence or 'maxit' reached.

Author(s)

Shuxin Yin <>
Youyi Fong <youyifong@gmail.com>
Krisztian Sebestyen <ksebestyen@gmail.com>

Examples

## Not run:

# options(path.svml = '\':/downloaded_scientific_programs/svmlight')
# options(path.svml = '~/bin/svmlight')

###########################################################
# a linear example

dat = sim.dat.1(n=200,seed=1)

# convergence takes long, to pass CRAN check, set maxit=1

fit1 = rauc(y~x1+x2, dat, lambda=2, kernel="linear", maxit=2)

#fit2 = rauc.linear (y~x1+x2, dat, lambda=2, verbose=TRUE)
#aux2=fit2$X %*% fit2$coefficients
#all(fit1$linear.combination-aux2<1e-2)

fit1$train.auc # 0.7206015

fit3 = rauc (y~x1+x2, dat, lambda=2, kernel="rbf", para=1, verbose=TRUE)

fit3$train.auc # 0.7773434

fit4 = svml (y~x1+x2, dat, kernel="r", fitted=FALSE, cost=1e4)
fast.auc(predict(fit4, dat)$posterior[,1], dat$y) # 0.7921805
tune.svml(y~x1+x2, dat, kernel="r")
# 1 10 100 1000 10000 1e+05
# 0.7027569 0.7254135 0.7517794 0.7653133 0.7921805 0.6674687

# glm derived score for comparision
fit.glm=glm(y~x1+x2, dat, family="binomial")
fast.auc(fit1$X %*% fit.glm$coef[-1], fit1$y) #

# add outliers
dat = sim.dat.1(n=200,seed=1, add.outliers=TRUE)
fit3 = rauc (y~x1+x2, dat, lambda=2, kernel="rbf", para=1, verbose=TRUE)
fit3$train.auc # 0.7066667
fit4 = svml (y~x1+x2, dat, kernel="r", fitted=FALSE, cost=1e4)
fast.auc(predict(fit4, dat)$posterior[,1], dat$y) # 0.6910101
tune.svml(y~x1+x2, dat, kernel="r")
# 1 10 100 1000 10000 1e+05
# 0.6485859 0.6705051 0.6722222 0.6767677 0.6910101 0.5007071

###############################################################
# a nonlinear example
###############################################################

# a nonlinear example
dat=skin.orange (n=100,seed=1,noise=FALSE)
dim(dat)

# nonlinear kernel fit
fit1 = rauc (y~x1+x2+x3+x4, dat, lambda=2, kernel="rbf", para=1, verbose=TRUE)
# glm fit
fit.glm=glm(y~x1+x2+x3+x4, dat, family="binomial")
# linear kernel fit
fit2 = rauc (y~x1+x2+x3+x4, dat, lambda=2, kernel="linear", start.method = "rlogit", verbose=TRUE)

# training data prediction
fast.auc(fit1$linear.combination, fit1$y)
fast.auc(fit1$X %*% fit.glm$coef[-1], fit1$y)
fast.auc(fit2$linear.combination, fit2$y)

# test data prediction
newdata=skin.orange (n=1000,seed=2,noise=FALSE)
fast.auc(predict(fit1, newdata), newdata$y)
fast.auc(as.matrix(subset(newdata, select=c(x1,x2,x3,x4))) %*% fit.glm$coef[-1], newdata$y)
fast.auc(predict(fit2, newdata), newdata$y)

######## IMPROVEMENTS ################################################################

## rank = 2 problem
dat = sim.dat.1(n=300,seed=1,add.outliers = TRUE,std.dev = 1.0); fm = y~x1+x2

## linear kernel and random working set selection - low rank (2) problem
## setting initial alpha (to be passed to minQuad at each iteration in dca-loop)
## to estimate from previous dca() iteration
## size of working set is automatically set
set.seed(100)
fit.lin = rauc (fm, dat,lambda=.1,kernel="linear",
verbose=TRUE,maxit = 100,tol = 1e-5,
init.alpha.from.previous = TRUE,mem.efficient = TRUE,
minQuad.control = control.minQuad(
  verbose = 1,maxit = 1e6,tol = 1e-4,
  method = "tron",
  working.set= "rv2wg")
)

## 'rbf' kernel and random working set selection
## low rank mapped to possibly infinite rank problem try larger working set 'q' set.seed(100)
## size of working set is set to q = 100
fit.rbf = rauc (fm, dat,lambda=.1,kernel="rbf",para = 1, verbose=TRUE,maxit = 100,tol = 1e-5,
init.alpha.from.previous = TRUE,mem.efficient = TRUE,
minQuad.control = control.minQuad(
  verbose = 1,maxit = 1e6,tol = 1e-4,
  q = 100,
  method = "tron",
  working.set= "rv2wg")
)

## End(Not run)

---

**Description**

Robust logistic regression estimator of Bianco and Yohai

**Usage**

```r
rlogit (formula, dat, const=0.5, kmax=1e3, maxhalf=10, verbose=FALSE)
## S3 method for class 'rlogit'
coef(object,...)
## S3 method for class 'rlogit'
trainauc(fit, training.data=NULL, ...)
## S3 method for class 'rlogit'
predict(object, newdata, ...)
## S3 method for class 'rlogit'
ratio(fit)
logistic.f(eta,h,loss=TRUE)
```
Arguments

- **formula**: a formula specifying the model to be fit.
- **dat**: a data frame containing the outcome and covariates in the model.
- **const**: tuning constant used in the computation of the estimator, defaults to 0.5.
- **kmax**: maximum number of iterations before convergence, defaults to 1000.
- **maxhalf**: max number of step-halving, defaults to 10.
- **verbose**: logical.
- **object**: an object of class `rlogit`.
- **fit**: an object that inherits from class `auc` such as `rauc` or `sauc`.
- **newdata**: data at which to predict.
- **training.data**: data frame used to compute auc based on a fit obtained by a call to `rauc`, `sauc` or `sauc.dca`.
- **eta,h**: logistic.f computes for loss = FALSE expit(eta/h) or expit(-eta/h) for loss = TRUE.
- **loss**: a boolean. if TRUE (default) logistic loss is assumed.
- **...**: arguments passed to or from methods.

Details

This program computes the estimator of Bianco and Yohai (1996) in logistic regression. By default, an intercept term is included and p parameters are estimated. The outcome is coded as a 0/1 binomial variable.

If initwml == TRUE, a weighted ML estimator is computed with weights derived from the MCD estimator computed on the explanatory variables. If initwml == FALSE, a classical ML fit is performed. When the explanatory variables contain binary observations, it is recommended to set initwml to FALSE or to modify the code of the algorithm to compute the weights only on the continuous variables.

Value

A list with the following components:

- **convergence**: logical, was convergence achieved.
- **objective**: value of the objective function at the minimum.
- **coef**: estimates for the parameters.
- **sterror**: standard errors of the parameters (if convergence is TRUE).

Author(s)

Christophe Croux, Gentiane Haesbroeck. Thanks to Kristel Joossens and Valentin Todorov for improving the code.
**References**

*Implementing the Bianco and Yohai estimator for Logistic Regression*
Computational Statistics and Data Analysis, 44, 273-295

**Examples**

```r
set.seed(1)
x0 <- matrix(rnorm(100,1))
y <- as.numeric(runif(100)>0.5)  # numeric(runif(100)>0.5)
dat=data.frame(y=y, x=x0)
rlogit(y~x, dat)
```

**roc**

ROC and AUC

**Description**

ROC/AUC methods. `fast.auc` calculates the AUC using a sort operation, instead of summing over pairwise differences in R.
`computeRoc` computes an ROC curve.
`plotRoc` plots an ROC curve.
`addRoc` adds an ROC curve to a plot.
`classification.error` computes classification error

**Usage**

```r
fast.auc(score, outcome, t0 = 0, t1 = 1, reverse.sign.if.nece = TRUE, quiet = FALSE)
computeRoc(score, outcome, reverse.sign.if.nece = TRUE, cutpoints = NULL)
plotRoc(x, add = FALSE, type = "l", diag.line=TRUE,...)
addRoc(x,...)
classification.error(score, outcome, threshold=NULL, verbose=FALSE)
```

**Arguments**

- **score**: a vector. Linear combination or score.
- **outcome**: a vector of 0 and 1. Outcome.
- **t0**: a number between 0 and 1 that is the lower boundary of pAUC
- **t1**: a number between 0 and 1 that is the upper boundary of pAUC
reverse.sign.if.nece
   a boolean. If TRUE, score is multiplied by -1 if AUC is less than 0.5.

x
   a list of two elements: sensitivity and specificity.

diag.line
   boolean. If TRUE, a diagonal line is plotted

add
   boolean. If TRUE, add to existing plot. If FALSE, create a new plot.

quiet
   boolean

cutpoints
   cutpoints

threshold
   threshold

verbose
   boolean

type
   line type for lines

... arguments passed to plot or lines

Details

These functions originally come from Thomas Lumley and Tianxi Cai et al.

Value

computeRoc returns a list of sensitivity and specificity.
plotRoc and addRoc plots ROC curves.

Author(s)

Shuxin Yin <>
Youyi Fong <youyifong@gmail.com>
Krisztian Sebestyen <>

Examples

n=1e2
score=c(rnorm(n/2,1), rnorm(n/2,0))
outcome=rep(1:0, each=n/2)
# cannot print due to r cmd check
#plotRoc(computeRoc(score, outcome))

# commented out b/c slower on pc and cause note when r cmd check
## test, fast.auc2 is a version without all the checking
#score=rnorm(1e5)
#outcome=rbinom(1e5,1,.5)
#system.time(for (i in 1:1e2) fast.auc(score,outcome)) # 4.9 sec
#system.time(for (i in 1:1e2) fast.auc2(score,outcome)) # 3.8 sec
Description

sample.for.cv does sampling for cross validation

Usage

sample.for.cv (dat, v, seed)

Arguments

dat a data frame. One of the columns must be named y and y should be 0/1 with 1 for case and 0 for control

v v-fold cross validation

seed seed for random number generators

Details

case and controls are sampled separately

Value

A list of two vector of integers: train and test, which refer to the rows of dat

Author(s)

Youyi Fong <youyifong@gmail.com>

Description

sauc.phi optimizes Normal CDF approximation of AUC using Newton Raphson

Usage

sauc.phi (formula, dat, constrain.method="L2", h.method="Lin", start.method="rlogit", opt.method = "Lin", upper = NULL, verbose = FALSE, ret.vcov = FALSE, truth = NULL, beta.init=NULL)
**Arguments**

- **formula**: a formula
- **dat**: a data frame
- **constrain.method**: a string. If "L2", L2 norm is constrained to 1. If "beta1", beta1 is fixed to 1. Default "L2".
- **h.method**: a string. If "Lin", Lin et al, data dependent. If "Vexler", \((n_1*n_2)^{-0.1}\) Vexler et al (2006). If "MH", Ma and Huang. Default "Lin".
- **start.method**: a string. If "rlogit", robust logistic fit is used as beta.init If "1", a vector of 1 is used as beta.init. Default "rlogit".
- **opt.method**: character string, possible values are "truth","YH","Lin", please see code for more details
- **upper**: required for opt.method = 'YH'
- **verbose**: logical
- **ret.vcov**: logical, whether to return an estimate of the covariance matrix of 'beta' for normal or logistic sigmoid functions.
- **truth**: numeric, it will be returned as the result of the fit, please see code for more details
- **beta.init**: vector. Initial values for coefficients.

**Details**

If an error happens during optimization (typically due to solve()), the errors are caught and NAs are returned.

**Author(s)**

Shuxin Yin <>
Ying Huang <>
Youyi Fong <youyifong@gmail.com>

**Examples**

```r
seed=26
seed=16
seed=3
dat.train = sim.dat.1(n=200, seed=seed, add.outliers=TRUE)
fits=list()
fits[[1]]=sauc.phi(y~x1+x2, dat.train, constrain.method="L2", h.method="Lin")
fits[[2]]=sauc.phi(y~x1+x2, dat.train, constrain.method="L2", h.method="MH")
fits[[3]]=sauc.phi(y~x1+x2, dat.train, constrain.method="beta1", h.method="Lin")
fits[[4]]=sauc.phi(y~x1+x2, dat.train, constrain.method="beta1", h.method="MH")
# not a good combination of constrain.method and h.method
sapply(fits, function(x) ratio(x)[2])
```
# explosion
seed=954
dat.train = sim.dat.1(n=200, seed=seed, add.outliers=TRUE)
fit.1 = sauc.phi(y~x1+x2, dat.train,constrain.method="L2",h.method="Lin")
ratio(fit.1)

---

**simulations**  
**Simulate datasets**

---

**Description**

sim.dat.1 simulates a dataset with two covariates to reproduce Pepe Figure 1. skin.orange simulates a skin of orange dataset as in Hastie et al.

**Usage**

sim.dat.1(n, seed, add.outliers=FALSE, std.dev = 0.2)

**Arguments**

- **n** sample size
- **seed** seed for random number generator
- **add.outliers** boolean. If TRUE, 10% of data are replaced by a contaminating distribution
- **std.dev** standard deviation in data generating process

**Value**

A data frame with n rows, and 4 columns: y, x1, x2, and eta, where eta is the linear combination X*beta.

**Author(s)**

- Shuxin Yin <>
- Youyi Fong <youyifong@gmail.com>
- Krisztian Sebestyen <>

**Examples**

dat = sim.dat.1(n=100,seed=1)
nrow(dat)

dat = sim.dat.1(n=100,seed=1,add.outliers=TRUE)
nrow(dat)
tune.it

Description

Tuning methods.

Usage

tune.it (formula, dat.train, dat.tune, method, kernel, verbose=TRUE, step.size = 2)
gamma0 (formula, dat.train)

Arguments

formula a formula object.
dat.train a data frame. Training data
dat.tune a data frame. Tuning data. If NULL, gacv is done.
method a string. "svm" or "rauc"
kernel a string. "rbf" or "linear"
verbose logical
step.size step size for lambda as in step.size^seq(-3,3,1), please see code for more details

Value

Tune.it returns a vector of one or two elements: lamcost and gamma, depending on the kernel.
gamma0 returns .5/quantile(dist., c(.9,.75,.5)), where dist. is the Euclidean distance between objects from two classes.

Author(s)

Shuxin Yin <>
Youyi Fong <youyifong@gmail.com>
Krisztian Sebestyen <ksebesty@gmail.com>
Index

*Topic area*
control.minQuad, 5
minQuad, 14
rauc, 18

*Topic auc*
control.minQuad, 5
minQuad, 14
rauc, 18

*Topic datasets*
bupa, 3
cleveland, 4
kyphosis, 13

*Topic distribution*
aucm, 3

*Topic receiver*
control.minQuad, 5
minQuad, 14
rauc, 18

*Topic roc*
control.minQuad, 5
minQuad, 14
rauc, 18
.C, 15

addRoc (roc), 23
auc, 2
auc.dca (dcsauc), 7
aucm, 3

bupa, 3
classification.error (roc), 23
cleveland, 4
coeff.auc (auc), 2
coeff.rlogit (rlogit), 21
computeRoc (roc), 23
control.minQuad, 5, 15, 16
dcsauc, 7
eauc (grid.auc), 11

fast.auc (roc), 23
gc, 19
get.gamma0 (tune.it), 28
get.X.diff, 9
getK, 8, 15, 18
getQ, 10
grid.auc, 11
kyphosis, 13
logistic.f (rlogit), 21
minQuad, 5–7, 14, 18
nsv (rauc), 18

phi.f (sauc.phi), 25
plot.gridsearch (grid.auc), 11
plot.rauc (rauc), 18
plotRoc (roc), 23
predict.auc (auc), 2
predict.rlogit (rlogit), 21
print.auc (auc), 2
print.minQuad (rauc), 18

ramp.f, 17
ratio (auc), 2
ratio.rlogit (rlogit), 21
rauc, 6, 7, 15, 16, 18
rlogit, 21
roc, 23

sample.for.cv, 25
sauc.phi, 25
sim.d20c (simulations), 27
sim.dat.1 (simulations), 27
sim.disc (simulations), 27
sim.easy (simulations), 27
sim.MH1 (simulations), 27
sim.MH11 (simulations), 27
sim.MH2 (simulations), 27
sim.mix (simulations), 27
sim.NL (simulations), 27
sim.NL1 (simulations), 27
sim.p1 (simulations), 27
sim.pepe.2 (simulations), 27
sim.ring (simulations), 27
sim.YH1 (simulations), 27
sim.YH2 (simulations), 27
simulations, 27
skin.orange (simulations), 27
smooth.rauc (dcsauc), 7
srauc (dcsauc), 7
summary.auc (auc), 2

trainauc (auc), 2
trainauc.rlogit (rlogit), 21
tune.it, 28
typeof, 5, 15

vcov.gridsearch (grid.auc), 11
vcov.rauc (rauc), 18
vcov.sauc (sauc.phi), 25