Package ‘bmrn’

February 19, 2015

Type Package
Title Bundle Methods for Regularized Risk Minimization Package
Version 3.0
Date 2015-01-13
Depends R (>= 3.0.2)
Imports lpSolve, LowRankQP
Author Julien Prados
Maintainer Julien Prados <julien.prados@unige.ch>
Copyright 2014, University of Geneva
Description Bundle methods for minimization of convex and non-convex risk under L1 or L2 regularization. Implements the algorithm proposed by Teo et al. (JMLR 2010) as well as the extension proposed by Do and Artieres (JMLR 2012). The package comes with lot of loss functions for machine learning which make it powerful for big data analysis. The applications includes: structured prediction, linear SVM, multi-class SVM, f-beta optimization, ROC optimization, ordinal regression, quantile regression, epsilon insensitive regression, least mean square, logistic regression, least absolute deviation regression (see package examples), etc... all with L1 and L2 regularization.
License GPL-3
NeedsCompilation no
Repository CRAN
Date/Publication 2015-01-15 16:59:17

R topics documented:

bmrn .......................................................... 2
costMatrix .................................................. 4
epsilonInsensitiveRegressionLoss ....................... 4
fbetaLoss .................................................... 5
gradient ...................................................... 6
**Description**

Implement Bundle Methods for Regularized Risk Minimization as described in Teo et. al 2007. Find \( w \) that minimize: \( \lambda \cdot \text{regularization\_norm}(w) + \text{lossfun}(w) \) where \( \text{regularization\_norm} \) is either L1 or L2.

**Usage**

```r
bmrm(lossfun, lambda = 1, max_iter = 100, epsilon_tol = 0.01,
      regfun = c("l1", "l2"), w0 = 0, verbose = TRUE)
```

**Arguments**

- **lossfun**: the loss function to use in the optimization (e.g.: `hingeLoss`, `softMarginVectorLoss`). The function must evaluate the loss value and its gradient for a given point vector \( w \).
- **lambda**: control the regularization strength in the optimization process. This is the value used as coefficient of the regularization term.
- **max_iter**: the maximum number of iteration to perform. The function stop with a warning message if the number of iteration exceed this value.
- **epsilon_tol**: control optimization stopping criteria: the optimization end when the optimization gap is below this threshold.
- **regfun**: type of regularization to consider in the optimization. It can either be the character string "l1" for L1-norm regularization, or "l2" (default) for L2-norm regularization.
- **w0**: initial weight vector where optimization start.
- **verbose**: a length one logical. Show progression of the convergence on stdout.

**Value**

a list of 2 fields: "w" the optimized weight vector; "log" a data.frame showing the trace of important values in the optimization process.
Author(s)
Julien Prados

References
Teo et al. A Scalable Modular Convex Solver for Regularized Risk Minimization. KDD 2007

See Also
hingeLoss softMarginVectorLoss

Examples
# -- Create a 2D dataset with the first 2 features of iris, with binary labels
x <- data.matrix(iris[1:2])
y <- c(-1,1,1)[iris$Species]

# -- Add a constant dimension to the dataset to learn the intercept
x <- cbind(x,1)

# -- train scalar prediction models with maxMarginLoss and fbetaLoss
models <- list(
    svm_L1 = bmrm(hingeLoss(x,y),LAMBDAR=0.1,regfun='l1',verbose=TRUE),
    svm_L2 = bmrm(hingeLoss(x,y),LAMBDAR=0.1,regfun='l2',verbose=TRUE),
    f1_L1 = bmrm(fbetaLoss(x,y),LAMBDAR=0.01,regfun='l1',verbose=TRUE)
)

# -- Plot the dataset and the predictions
plot(x,pch=20,y.main="dataset & hyperplanes")
legend('bottomright',legend=names(models),col=seq_along(models),lty=1,cex=0.75,lwd=3)
for(i in seq_along(models)) {
    m <- models[[i]]
    if (m$w[2]!={0}) abline(-m$w[3]/m$w[2],-m$w[1]/m$w[2],col=i,lwd=3)
}

rx <- range(na.rm=TRUE,1,col=seq_along(models),lty=1,cex=0.75,lwd=3)
plot(rx,ry,type="n",ylab="epsilon gap",xlab="iteration",main="evolution of the epsilon gap")
for(i in seq_along(models)) {
    m <- models[[i]]
    lines(m$log$epsilon,type="o",col=i,lwd=3)
}

# --
set.seed(123)
X <- matrix(rnorm(4000*200), 4000, 200)
beta <- c(rep(1,ncol(X)-4),0,0,0,0)
Y <- X%*%beta + rnorm(nrow(X))
model <- bmrm(ladRegressionLoss(X,Y),regfun="l2",LAMBDAR=100,MAX_ITER=150)
layout(1)
costMatrix

Compute or check the structure of a cost matrix

Description
Compute or check the structure of a cost matrix

Usage
`costMatrix(y, C = c("0/1", "linear"))`

Arguments
- `y`: a factor representing the labels of the instances
- `C`: either a cost matrix to check for consistency with labels in `y`, or a character string defining the standard matrix to compute. If a character string the accepted values are "0/1" for a 0-1 cost matrix or "linear" for linear cost.

Value
the cost matrix object

See Also
`bmrn, ordinalRegressionLoss`

epsilonInsensitiveRegressionLoss

The loss function to perform a epsilon-insensitive regression (Vapnik et al. 1997)

Description
The loss function to perform a epsilon-insensitive regression (Vapnik et al. 1997)

Usage
`epsilonInsensitiveRegressionLoss(x, y, epsilon)`

Arguments
- `x`: matrix of training instances (one instance by row)
- `y`: numeric vector of values representing the training labels for each instance in `x`
- `epsilon`: a numeric value setting tolerance of the epsilon-regression
fbetaLoss

Value

a function taking one argument w and computing the loss value and the gradient at point w

References

Teo et al. Bundle Methods for Regularized Risk Minimization JMLR 2010

See Also

bmrm

fbetaLoss  F beta score loss function

Description

F beta score loss function

Usage

fbetaLoss(x, y, beta = 1)

Arguments

x  matrix of training instances (one instance by row)
y  numeric vector of values in (-1,+1) representing the training labels for each instance in x
beta  a numeric value setting the beta parameter is the f-beta score

Value

a function taking one argument w and computing the loss value and the gradient at point w

References

Teo et al. A Scalable Modular Convex Solver for Regularized Risk Minimization. KDD 2007

See Also

bmrm
gradient

Return or set gradient attribute

Description
Return or set gradient attribute

Usage
gradient(x, ...)

## Default S3 method:
gradient(x, ...)

gradient(x, ...) <- value

## Default S3 replacement method:
gradient(x, ...) <- value

Arguments
x any R object
...
additional parameters
value new gradient value to set

Details
gradient attribute is used by loss/risk function to return the gradient of the function at a given point together with the function value

Value
attr(x,"gradient")

hingeLoss

Hinge Loss function for SVM

Description
Hinge Loss function for SVM

Usage
hingeLoss(x, y, loss.weights = 1)
**ladRegressionLoss**

Arguments

- **x**: matrix of training instances (one instance by row)
- **y**: numeric vector of values in (-1,+1) representing the training labels for each instance in x
- **loss.weights**: numeric vector of loss weights to incur for each instance of x in case of mis-prediction. Vector length should match length(y), but values are cycled if not of identical size. Default to 1 so we define a standard 0/1 loss for SVM classifier. The parameter might be useful to adapt SVM learning in case of unbalanced class distribution.

Value

a function taking one argument w and computing the loss value and the gradient at point w

References

Teo et al. A Scalable Modular Convex Solver for Regularized Risk Minimization. KDD 2007

See Also

- bmrm

---

**ladRegressionLoss**  
*The loss function to perform a least absolute deviation regression*

Description

The loss function to perform a least absolute deviation regression

Usage

ladRegressionLoss(x, y)

Arguments

- **x**: matrix of training instances (one instance by row)
- **y**: numeric vector of values representing the training labels for each instance in x

Value

a function taking one argument w and computing the loss value and the gradient at point w

References

Teo et al. Bundle Methods for Regularized Risk Minimization JMLR 2010

See Also

- bmrm
**lmsRegressionLoss**  
*The loss function to perform a least mean square regression*

**Description**

The loss function to perform a least mean square regression

**Usage**

lmsRegressionLoss(x, y)

**Arguments**

- **x**  
  matrix of training instances (one instance by row)
- **y**  
  numeric vector of values representing the training labels for each instance in x

**Value**

a function taking one argument w and computing the loss value and the gradient at point w

**References**

Teo et al. Bundle Methods for Regularized Risk Minimization JMLR 2010

**See Also**

bmr

---

**logisticRegressionLoss**  
*The loss function to perform a logistic regression*

**Description**

The loss function to perform a logistic regression

**Usage**

logisticRegressionLoss(x, y)

**Arguments**

- **x**  
  matrix of training instances (one instance by row)
- **y**  
  numeric vector of values representing the training labels for each instance in x
Value

a function taking one argument \( w \) and computing the loss value and the gradient at point \( w \)

References

Teo et al. Bundle Methods for Regularized Risk Minimization JMLR 2010

See Also

bmrm

nrbm

Convex and non-convex risk minimization with L2 regularization and limited memory

Description

Use algorithm of Do and Artieres, JMLR 2012 to find \( w \) minimizing:

\[
f(w) = 0.5 \times \text{LAMBDA} \times \text{l2norm}(w) + \text{riskFun}(w)
\]

where \( \text{riskFun} \) is either a convex or a non-convex risk function.

Usage

```r
nrbm(riskfunL, lambda = 1, max_iter = 1000L, EPSILON_TOL = 0.01, w0 = 0,
     maxCP = 100L, convexRisk = TRUE)
```

Arguments

- `riskFun`: the risk function to use in the optimization (e.g.: `hingeLoss`, `softMarginVectorLoss`). The function must evaluate the loss value and its gradient for a given point vector \( (w) \).
- `lambda`: control the regularization strength in the optimization process. This is the value used as coefficient of the regularization term.
- `max_iter`: the maximum number of iteration to perform. The function stop with a warning message if the number of iteration exceed this value.
- `EPSILON_TOL`: control optimization stoping criteria: the optimization end when the optimization gap is below this threshold.
- `w0`: initial weight vector where optimization start.
- `maxCP`: maximal number of cutting plane to use to limit memory footprint.
- `convexRisk`: a length 1 logical telling if the risk function `riskFun` is convex. If TRUE, use CRBM algorithm; if FALSE use NRBM algorithm from Do and Artieres, JMLR 2012.

Value

the optimal weight vector \( (w) \)
Examples

```r
set.seed(123)
X <- matrix(rnorm(4000*200), 4000, 200)
beta <- c(rep(1,nrow(X)),0,0,0,0)
Y <- X%*%beta + rnorm(nrow(X))
w <- nrbm(ladRegressionLoss(X/100,Y/100),maxCP=50)
layout(1)
barplot(w)
```

The loss function for ordinal regression

**Description**

The loss function for ordinal regression

**Usage**

```r
ordinalRegressionLoss(x, y, C = "0/1", impl = c("loglin", "quadratic"))
```

**Arguments**

- `x`: matrix of training instances (one instance by row)
- `y`: integer vector of positive values (>=1) representing the training labels for each instance in `x`
- `C`: the cost matrix to use, C[i,j] being the cost for predicting label i instead of label j.
- `impl`: either the string "loglin" or "quadratic", that define the implementation to use for the computation of the loss.

**Value**

a function taking one argument w and computing the loss value and the gradient at point w

**References**

Teo et al. Bundle Methods for Regularized Risk Minimization JMLR 2010

**See Also**

bmrn
quantileRegressionLoss

Examples

# -- Load the data
x <- data.matrix(iris[1:4])
y <- as.integer(iris$Species)

# -- Train the model
m <- bmrn(ordinalRegressionLoss(x), LAMBDA=0.001, EPSILON_TOL=0.0001)
m2 <- bmrn(ordinalRegressionLoss(x, y, impl="quadratic"), LAMBDA=0.001, EPSILON_TOL=0.0001)

# -- plot predictions
f <- x %*% m$w
f2 <- x %*% m2$w
layout(1:2)
plot(y, f)
plot(f, f2, main="compare predictions of quadratic and loglin implementations")

# -- Compute accuracy
ij <- expand.grid(i=seq(nrow(x)), j=seq(nrow(x)))
n <- tapply(f[ij$i] - f[ij$j] > 0, list(y[ij$i], y[ij$j]), sum)
N <- table(y[ij$i], y[ij$j])
print(n/N)

quantileRegressionLoss

The loss function to perform a quantile regression

Description

The loss function to perform a quantile regression

Usage

quantileRegressionLoss(x, y, q = 0.5)

Arguments

x                 matrix of training instances (one instance by row)
y                 numeric vector of values representing the training labels for each instance in x
q                 a numeric value in the range [0-1] defining quantile value to consider

Value

a function taking one argument w and computing the loss value and the gradient at point w

References

Teo et al. Bundle Methods for Regularized Risk Minimization JMLR 2010
rocLoss

The loss function to maximize area under the ROC curve

Description

The loss function to maximize area under the ROC curve

Usage

rocLoss(x, y)

Arguments

x  matrix of training instances (one instance by row)
y  numeric vector of values in (-1,+1) representing the training labels for each instance in x

Value

a function taking one argument w and computing the loss value and the gradient at point w

References

Teo et al. Bundle Methods for Regularized Risk Minimization JMLR 2010

See Also

bmrm

softMarginVectorLoss

Soft Margin Vector Loss function for multiclass SVM

Description

Soft Margin Vector Loss function for multiclass SVM

Usage

softMarginVectorLoss(x, y, l = "0/1")
softMarginVectorLoss

Arguments

- \( x \): instance matrix, where \( x(t,.) \) defines the features of instance \( t \)
- \( y \): target vector where \( y(t) \) is an integer encoding target of \( x(t,) \)
- \( l \): loss matrix. \( l(t,p(t)) \) must be the loss for predicting target \( p(t) \) instead of \( y(t) \) for instance \( t \). By default, the parameter is set to character value "0/1" so that the loss is set to a 0/1 loss matrix.

Value

a function taking one argument \( w \) and computing the loss value and the gradient at point \( w \)

References

Teo et al. A Scalable Modular Convex Solver for Regularized Risk Minimization. KDD 2007

Examples

```r
# -- Load the data
x <- data.matrix(iris[1:2])
y <- as.integer(iris$Species)

# -- Add a constant dimension to the dataset to learn the intercept
cst <- sqrt(max(rowSums(x*x)))
x <- cbind(x,cst)

# -- train a multi-class SVM & compute the predictions
train.multiclassSVM <- function(x,y,...) {
  m <- bmrn(softMarginVectorLoss(x,y),...)
  m$w <- matrix(m$w,ncol(x))
  m$f <- x %*% m$w
  m$y <- max.col(m$f)
  m$contingencyTable <- table(y,m$y)
  print(m$contingencyTable)
  return(m)
}

# # train a l1-regularized multi-class SVM
m <- train.multiclassSVM(x,y,MAX_ITER=50,regfun='l1',LAMBDAM=1)

# -- Plot the dataset and the decision boundaries
gx <- seq(min(x[,1]),max(x[,1]),length=200) # positions of the probes on x-axis
gy <- seq(min(x[,2]),max(x[,2]),length=200) # positions of the probes on y-axis
Y <- outer(gx,gy,function(a,b){
  max.col(cbind(a,b,cst) %*% m$w)
}) # matrix of predictions for all probes
layout(matrix(c(1,3,2,3),2,2))
image(gx,gy,Y,asp=1,main="dataset & decision boundaries",xlab=colnames(x)[1],ylab=colnames(x)[2])
points(x,pch=19)
pred <- m$log$epsilon,type="o",ylab="epsilon gap",xlab="iteration")
plot(row(m$f),m$y,pch=19+col(m$f),ylab="prediction values",xlab="sample")
```
Index

bmrn, 2

costMatrix, 4

epsilonInsensitiveRegressionLoss, 4

fbetaLoss, 5

gradient, 6
gradient<- (gradient), 6

hingeLoss, 3, 6

ladRegressionLoss, 7

lmsRegressionLoss, 8

logisticRegressionLoss, 8

nrbm, 9

ordinalRegressionLoss, 10

quantileRegressionLoss, 11

rocLoss, 12

softMarginVectorLoss, 3, 12